Georg-August-Universität
Göttingen
Zentrum für Informatik

ISSN 1612-6793
Nummer ZFI-BM-2004-23

Bachelorarbeit

im Studiengang „Angewandte Informatik“

Natürliche Sprachverarbeitung für das Expertensystem ScienceAtlas

Stefan E. Funk

am Lehrstuhl für

Informatik

Bachelor- und Masterarbeiten
des Zentrums für Informatik
an der Georg-August-Universität Göttingen

11. Oktober 2004

Georg-August-Universität Göttingen
Zentrum für Informatik

Lotzestraße 16-18
37083 Göttingen
Germany

Tel. +49 (5 51) 39-1 44 02
Fax +49 (5 51) 39-1 44 03
Email office@informatik.uni-goettingen.de
WWW www.informatik.uni-goettingen.de

Ich erkläre hiermit, daß ich die vorliegende Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Göttingen, den 11. Oktober 2004

Bachelorarbeit

Natürliche Sprachverarbeitung für das Expertensystem ScienceAtlas

Stefan E. Funk

11. Oktober 2004

Betreut durch Prof. Dr. Roland Potthast
Institut für Numerische und Angewandte Mathematik
Georg-August-Universität Göttingen

Inhaltsverzeichnis

1 Einleitung

In dieser Arbeit untersuche ich, welche Möglichkeiten bestehen, die natürlichsprachliche Analyse des Expertensystems *ScienceAtlas* zu erweitern. ScienceAtlas ist ein auf dem Medium Internet basierendes mathematisches Expertensystem, mit dem der Benutzer in einer Wissensbasis Begriffe nachschlagen oder Vorlesungsscripte einsehen kann. Außerdem kann mit Variablen, Zahlen, Brüchen, Matritzen, Funktionen und Scripten programmiert und gerechnet werden. Der Nutzer kann Fragen oder Aussagen in schriftlicher Form in das System eingeben. Die Verarbeitung natürlicher Sprache ist momentan durch eine Art Pattern Matching mit regulären Ausdrücken beschränkt, so daß festgelegte Muster erkannt und interpretiert werden können, jedoch nur sehr eingeschränkt auf die Eigenheiten der deutschen Sprache eingegangen werden kann.

Die Untersuchung hat folgende Ziele:

- Mit Hilfe eines natürlichsprachlichen Parsers werden die Eingaben der Benutzer analysiert. Dieser erkennt Syntax- und Semantikfehler in der Eingabe und ist in der Lage, diese sehr präzise zu benennen.
- Die Eingaben der Benutzer werden mit dem vorhandenen Wissen des Expertensystems kombiniert. Es werden alle vorhandenen Wissensdatenbanken genutzt: Die allgemeine Wissensbasis, frühere Eingaben des Nutzers und seine im System definierten Variablen.
- Schließlich soll der Benutzer auf seine Eingabe eine Antwort vom System bekommen. Diese Antwort soll im Rahmen der Möglichkeiten des ScienceAtlas-Universums die Erwartungen des Nutzers zufriedenstellen.

Die Analyse von Schriftsprache und die Kommunikation zwischen Mensch und Computer spielt in dieser Arbeit eine wesentliche Rolle. Ich erläutere die Grundlagen einer natürlichsprachlichen Analyse und untersuche die Möglichkeiten einer Antwortgebung seitens des Expertensystems. Ich werde versuchen, unter Verwendung der zur Verfügung stehenden Mittel von einer Nutzereingabe in das System zu einer für den Nutzer befriedigenden Ausgabe zu kommen. Dabei nutze ich eine natürlichsprachliche Grammatik mit ihrem Lexikon, um die Eingabe syntaktisch wie semantisch zu analysieren. Das Expertenwissen des Systems wie der Variablenraum und die vorherigen Eingaben des Nutzers sollen in die semantische Analyse mit einbezogen werden.

2 Sprache und Schrift

2.1 Natürliche Sprachen

Natürliche Sprachen sind Medien der Kommunikation, sie dienen dem Austausch von Gedanken, der Archivierung von Wissen, der Weitergabe von Werten, der Unterhaltung. Sie sind im Laufe der Jahrhunderte gewachsene Systeme, die sich durch fortwährende Benutzung weiterentwickeln und sich stetig verändern. Daraus resultieren Unregelmäßigkeiten, die sich in jeder Sprache anders zeigen und auch auf verschiedenen Ebenen auftauchen. So gibt es z.B. regionale Unterschiede, soziale Unterschiede, situationsbedingte Unterschiede sowie unterschiedliche Gewohnheiten verschiedener Sprecher. Natürliche Sprachen sind somit recht inhomogene Gebilde.

Die Schrift hat sich aus und auch mit der Sprache entwickelt. Sie kann als ein (unvollkommenes) Abbild der Sprache aufgefasst werden, denn eine gewisse Menge an Information geht vom Übergang der Sprache zur Schrift verloren. Es kommt darauf an, wer einen Satz ausspricht und auf welche Art und Weise dieser Satz betont wird. Gesprochene Sprache, die aus Worten besteht, transportiert sehr viel mehr als nur die Wörter, die aufgeschrieben werden können.

Es können also gesprochene Sprache als Worte und geschriebene Sprache als Wörter bezeichnet werden. Worte werden gesagt, können ausgesprochen sowie vernommen werden. Wörter kann man entziffern, erkennen oder malen und auch als Wort aussprechen. Worte können anrühren und verletzen, sie können aufrichtig oder gelogen, liebevoll oder haßerfüllt sein. Ein geschriebenes Wort hingegen kann nicht an sich liebevoll sein, sondern nur dessen Bedeutung in einem gewissen Zusammenhang.

Im Zusammenhang mit Schrift werden immer Zeichen benötigt, mit denen Sprache geschrieben wird. Zwei Arten von Zeichen können hier unterschieden werden. An erster Stelle möchte ich die *Buchstabenschriften* nennen, beispielsweise die Zeichen unseres Alphabets oder die des Griechischen. Auch die Zeichen der japanischen Silbenschriften, dem *Hiragana* und dem *Katakana*, zähle ich zu dieser ersten Kategorie. Das chinesiche *Han ze* und das japanische *Kanji* gehören in die Kategorie der *Symbolschriften*, wobei das Kanji genau genommen Zeichen des Han ze benutzt.[1] Han ze kann mit „chinesische Schrift“ übersetzt

[1] *Kanji* bedeutet auch „chinesische Zeichen“. *Kan* steht als japanische Lesart für die Han-Dynastie – also

werden, meint eigentlich jedoch als Plural die Gesamtheit der chinesischen Zeichen, die diese Schrift ausmachen. Jedes Zeichen hat eine bestimmte Bedeutung und steht nicht als abstrakter Ausdruck, als ein Wort oder ein Buchstabe, „hinter“ dem ein bestimmter Inhalt steht bzw. der einen bestimmten Laut bezeichnet. Weitere Informationen zu den japanischen Schriften finden sich im Internet, siehe [14] oder [2].

Bei überaus komplexen Gebilden, die natürlich gewachsene Sprachen darstellen, deren Struktur und Regelsysteme oft kompliziert, umfangreich und voller Ausnahmeregelungen sind, ist es sehr schwierig, eine Eingabe in einer natürlichen Sprache syntaktisch wie semantisch korrekt zu analysieren, um letztendlich zu einer befriedigenden Ausgabe innerhalb eines natürlichsprachlichen Systems zu kommen.

Die *syntaktische Analyse* stellt hohe Anforderungen an das Lexikon und an die verwendete Grammatik. Die Möglichkeiten sind hier zwar eingeschränkt, jedoch existieren Verfahren, die zu einem guten Ergebnis führen. Das Lexikon und die gewählte Grammatik begrenzen die Ergebnisse der syntaktischen Analyse.

Die *semantische Analyse* eines Satzes kann bis zu einem gewissen Grad mit der Syntaktischen einhergehen, wenn es z.B. um Sachverhalte geht, die innerhalb eines Satzes darstellbar sind oder die aus der Grammatik hervorgehen[2]. Kompliziertere Sachverhalte um das sogenannte *Weltwissen*[3] – das benötigt wird, um diese Sachverhalte zu behandeln – begrenzen das Ergebnis der semantischen Analyse. [13] [5] [28]

2.2 Künstliche Sprachen

Betrachtet man nicht nur die natürlichen Sprachen, sondern auch die künstlichen, wie zum Beispiel die Programmiersprachen, sieht man, daß diese weitaus weniger kompliziert sind. Sie sind, im Vergleich zu unseren natürlich gewachsenen Sprachen, in vergleichsweise kurzer Zeit und für einen sehr speziellen Zweck erdacht worden. Damit ein Compiler einen Programmtext einer „höheren“ Programmiersprache wie z.B. `Java`, `Prolog` oder `Basic` in eine „maschinennahe“ Sprache wie `Assembler` oder `Java Bytecode` übersetzen kann, muß er vorher entscheiden können, ob der Text der Quellsprache korrekt oder fehlerhaft ist. Die formale Struktur oder die Syntax der Quellsprache muß also *vollständig* und *eindeutig* formuliert sein, so daß eine zweifelsfreie Entscheidung möglich ist. [30]

für Zeichen aus China – und *ji* bedeutet „Zeichen“.

[2] z.B. „Suchender“ und „Gesuchter“, siehe *semantische Rollen*, Kapitel 3.1

[3] Mit *Weltwissen* bezeichne ich die Kenntnis von der Welt, die das natürlichsprachliche System bzw. das Expertensystem sein Eigen nennen kann. Dinge wie *Was kann gegeben werden?*, *Kann ich ein Auto essen?*, *Wurde bereits ein Graph erfolgreich gezeichnet?*, usw.

2.3 Formale Sprachen

Eine Klasse von Sprachen, die den oben genannten Forderungen nach Vollständigkeit und Eindeutigkeit gerecht wird, sind die *formalen Sprachen*. Alle Programmiersprachen sind Beispiele für formale Sprachen. Natürliche Sprachen können ebenfalls mit formalen Sprachen beschrieben werden, jedoch gibt es hier einige Einschränkungen, da bei natürlichen Sprachen die Semantik sehr viel umfangreicher ausfällt als bei Programmiersprachen – siehe Kapitel 3.2.

Eine formale Sprache ist eine endliche Menge $\mathcal{T}$ von *Symbolen*. Die Symbole können Buchstaben und Ziffern sein, sie repräsentieren jedoch meist abstrakte Begriffe wie z.B die Schlüsselwörter einer Sprache wie `Java` oder `C`, wobei jedes Schlüsselwort durch ein eigenes Symbol dargestellt wird. Ein solches Symbol wird bei den formalen Sprachen ähnlich benutzt wie ein Wort in einer natürlichen Sprache. Aus diesen Symbolen (oder Worten) können nun beliebig lange Symbolfolgen gebildet werden. Eine *Symbolfolge* der Länge n ist ein geordnetes n-Tupel

$$\mathcal{T}^n \quad = \quad \mathcal{T} \times \mathcal{T} \times \ldots \times \mathcal{T}$$

und entspricht der Menge aller Symbolfolgen der Länge n, die aus der Menge $\mathcal{T}$ aller Symbole gebildet werden können. Die Menge aller Symbolfolgen ist dann die Vereinigung

$$\mathcal{T}^* \quad = \quad \cup_{n \geq 0} \, \mathcal{T}^n$$

und $\mathcal{T}^0$ besteht aus der *leeren* Symbolfolge, die im Allgemeinen mit ϵ bezeichnet wird. Auf der Menge $\mathcal{T}^*$ gibt es die Operation der *Verkettung*, die aus den Symbolfolgen u und v die Symbolfolge uv bildet:

$$\begin{array}{rcl} \mathcal{T}^* \times \mathcal{T}^* & \rightarrow & \mathcal{T}^* \\ (u, v) & \mapsto & uv \end{array}$$

Das neutrale Element dieser Operation ist ϵ und die Verkettung ist *assoziativ*, denn es gilt

$$\begin{array}{rl} \epsilon x = x\epsilon = x & \forall x \in \mathcal{T}^* \\ u(vw) = (uv)w & \forall u, v, w \in \mathcal{T}^* \end{array}$$

Eine beliebige Sprache $\mathcal{L}$ über der Symbolmenge $\mathcal{T}$ ist eine beliebige Teilmenge von $\mathcal{T}^*$ und eine formale Sprache ist eine Teilmenge $\mathcal{L}$ von $\mathcal{T}^*$, die durch endlich viele *Produktionsregeln* beschrieben wird.

Für die Beschreibung der Produktionsregeln werden Symbole aus $\mathcal{T}$ benötigt, die *Terminale* (im Folgenden bezeichnet mit $a, b, c, \ldots$), Symbole aus $\mathcal{N}$, die *Nicht-Terminale*

(bezeichnet mit $A, B, C, \ldots$) und schließlich noch Symbolfolgen aus $\mathcal{T}^*$ oder $(\mathcal{T} \cup \mathcal{N})^*$ (bezeichnet mit $\alpha, \beta, \gamma, \ldots$). Die Nicht-Terminale sind das, was in der Linguistik als *syntaktische Kategorien* bezeichnet wird – also etwa die Klasse der Nomen, der Adjektive, der Verben oder der Nominalphrasen. Bei den Programmiersprachen beinhalten die Nicht-Terminale Konstrukte wie Klassen, Ausdrücke, Funktionen oder If-Else-Klauseln. Produktionsregeln haben die Form $\alpha \rightarrow \beta$ und werden benutzt, um mit Hilfe einer *Grammatik* aus einem *Startsymbol* $\mathcal{S} \in \mathcal{N}$ alle syntaktisch korrekten Symbolfolgen der Sprache $\mathcal{L}$ zu erzeugen.

Eine Grammatik $\mathcal{G}$ ist also ein 4-Tupel $(\mathcal{T}, \mathcal{N}, \mathcal{S}, \mathcal{P})$ und besteht aus der Menge $\mathcal{T}$ der Terminalsymbole, der Menge $\mathcal{N}$ der Nicht-Terminalsymbole, dem Startsymbol $\mathcal{S} \in \mathcal{N}$ und der Menge $\mathcal{P}$ der Produktionsregeln $\alpha \rightarrow \beta$ mit $\alpha, \beta \in (\mathcal{T} \cup \mathcal{N})^*$. Die Grammatik $\mathcal{G}$ definiert nun eine *Regelsprache*

$$\mathcal{L}(\mathcal{G}) \quad = \quad \{\sigma \in \mathcal{T}^* \mid \mathcal{S} \stackrel{*}{\Rightarrow} \sigma\}$$

Diese besteht somit aus allen Symbolfolgen, die nur Terminale enthalten und direkt oder indirekt aus $\mathcal{S}$ *ableitbar*[4] sind.

Beinhalten die Produktionsregeln Regeln der Form $\lambda\alpha\mu \rightarrow \lambda\beta\mu$, spricht man von einer *kontextsensitiven* Grammatik[5]. Eine solche Regel bedeutet, daß α nur dann durch β ersetzt werden darf, wenn es zwischen λ und μ steht. Von *kontextfreien* Grammatiken redet man, wenn nur Produktionsregeln der Form $\mathcal{A} \rightarrow \alpha$ existieren. Diese sind erheblich leichter zu verarbeiten. Von *kontextfreien Sprachen* redet man, wenn sie von kontextfreien Grammatiken erzeugt werden, kontextsensitive Grammatiken wiederum erzeugen *kontextsensitive Sprachen*. [30]

2.4 Sprachentypologie nach Chomsky

Noam Chomsky unterscheidet grundsätzlich vier Sprachtypen nach den Einschränkungen der Regeln ihrer Grammatiken:

1. **Typ 0-Grammatiken**

   Für die Produktionsregeln gelten keine besonderen Beschränkungen, diese sind lediglich rekursiv aufrufbar. So darf auf der rechten Regelseite auch eine kürzere nicht-leere Kette stehen als auf der linken Seite. Für diese allgemeinste Grammatik und deren erzeugte Sprachen ist das Elementproblem nicht generell lösbar. Es gibt somit Regelsyntaxen $\mathcal{G} = (\mathcal{T}, \mathcal{N}, \mathcal{S}, \mathcal{P})$, so daß sich kein Algorithmus angeben läßt, mit dem für

[4] Der Begriff der Ableitbarkeit ist in Kapitel 3.2 mit Beispielen erläutert.
[5] siehe Kapitel 3.2.2

jede Kette σ über den Symbolen aus $\mathcal{T}$ entschieden werden kann, ob $\sigma \in \mathcal{L}(\mathcal{G})$ oder $\sigma \notin \mathcal{L}(\mathcal{G})$.

2. **Typ 1-Grammatiken**

   Bei den *kontextsensitiven* Grammatiken ist die rechte Seite der Produktionsregeln immer länger als die linke Seite. Das Elementproblem ist für alle kontextsensitiven Grammatiken lösbar. Man sagt auch, daß alle von einer kontextsensitiven Grammatik erzeugten Sprachen *entscheidbar* sind, man kann also entscheiden, ob ein Text einer solchen Sprache korrekt ist oder nicht. Somit können Parser konstruiert werden. Beispiel: $\mathcal{L} = \{x^n y^n z^n \mid n \geq 1\}$.

3. **Typ 2-Grammatiken**

   Die *kontextfreien* Grammatiken haben auf der linken Seite immer nur ein einzelnes Symbol. Auch für diese ist das Elementproblem lösbar, denn es ist bewiesen, daß die von kontextfreien Grammatiken erzeugten Sprachen eine echte Teilmenge der von kontextsensitiven Grammatiken erzeugten Sprachen sind. Beispiel: formale Sprachen wie die der arithmetischen Ausdrücke, $\mathcal{L} = \{x^n y^n \mid n \geq 1\}$.

4. **Typ 3-Grammatiken**

   Reguläre Sprachen schließlich dürfen auf der linken Seite ebenfalls nur ein einzelnes Symbol enthalten und auf der rechten Seite ein Terminalsymbol und ein Nicht-Terminalsymbol. Beispiel: Alle mit einem deterministischen endlichen Automaten oder kurz DFA (engl. deterministic finite automaton) erkennbaren Sprachen wie z.B. die regulären Ausdrücke.

Eine Sprache $\mathcal{L}$ ist vom Typ 0 (Typ 1, Typ 2, Typ 3), falls eine Grammatik $\mathcal{G}$ von Typ 0 (Typ 1, Typ 2, Typ 3) existiert, so daß $\mathcal{L}(\mathcal{G}) = \mathcal{L}$. Es gilt immer: $Typ\ i + 1 \subset Typ\ i$ und $Typ\ i + 1 \neq Typ\ i$. [12] [16] [7] [30]

3 Beschreibung natürlicher Sprachen

Die *Semiotik*, die Lehre von den Zeichen und den Symbolen, ist der Oberbegriff für die *Syntax*, die *Semantik* und die *Pragmatik*, die benötigt werden, wenn man Sprachen beschreiben und analysieren will.

Die Syntax bezeichnet die Lehre von der Beziehung der Zeichen zueinander, den Hierarchiebeziehungen, Vorkommensbeschränkungen und Abfolgebeziehungen von sprachlichen Einheiten wie Worten, Satzteilen und ganzen Sätzen. Die Syntax behandelt also die *Zeichen-Zeichen Relationen*.

Die Semantik ist die Lehre von der Beziehung der Zeichen zum Bezeichneten, es geht um die Bedeutung solcher sprachlichen Einheiten. Die Semantik behandelt *Zeichen-Bedeutung Relationen*.

Als Pragmatik schließlich wird die Lehre von der Beziehung der Zeichen zum Benutzer bezeichnet, der Benutzung von sprachlichen Einheiten in einer bestimmten Situation. Es handelt sich um *Zeichen-Benutzer Relationen*.

Mit der Syntax einer Sprache $\mathcal{L}$ sollen Sätze von $\mathcal{L}$ produziert und ihrer syntaktische Struktur erkannt werden. Es sollen möglichst *alle* Sätze von $\mathcal{L}$ und *nur* Sätze von $\mathcal{L}$ produziert und erkannt werden können. Die Syntax oder auch Grammatik von $\mathcal{L}$ muß also *vollständig* und *korrekt* sein.

Die Forderung nach Vollständigkeit ist aufgrund der Inhomogenität einer natürlichen Sprache nur näherungsweise erfüllbar. Möchte man eine Syntax für eine solche Sprache schreiben, beschränkt man sich auf einen bestimmten, aber umfassenden Ausschnitt dieser Sprache und behält seine Ziele und Resourcen im Auge.

Die Überprüfung der Korrektheit einer Syntax kann mit Hilfe geeigneter Computerprogramme – der Parser – vorgenommen werden, sofern die Syntax in einem streng festgelegten Formalismus formuliert ist[1].

Da ich in diesem Kapitel nicht alle Grammatiken und Theorien aufführen kann, verweise ich die interessierten Leser zu Themen wie der *Transformationsgrammatiken* und der *Theorie*

[1] Zum Thema Entscheidbarkeit siehe Kapitel 2.4 und zum Thema Parser Kapitel 4.

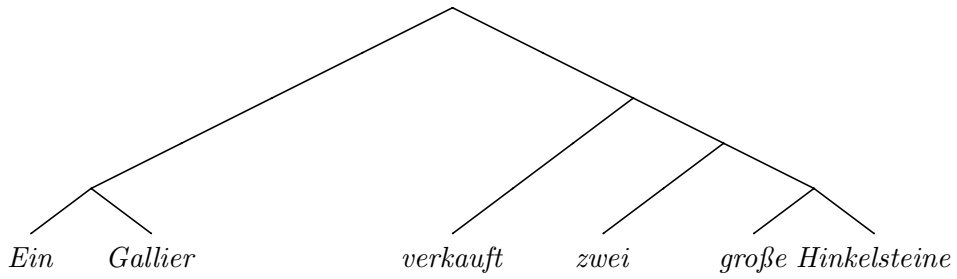

Abbildung 3.1: Die Konstituentenstruktur von Ein Gallier verkauft zwei große Hinkelsteine – *dargestellt als Baum.*

der Prinzipien und Parameter[2], der *GPSG*[3] und der *HPSG*[4], sowie weiteren Theorien und zur Vertiefung der hier behandelten Themen auf [16]. [16] [18] [27]

3.1 Konstituentenstrukturen

Eine Konstituente eines Ausdrucks $\mathcal{A}$ ist ein Ausdruck $\mathcal{B}$ als Teil von Ausdruck $\mathcal{A}$, wobei $\mathcal{B}$ wiederum aus mehreren Konstituenten bestehen kann. In dem Satz *Ein Gallier verkauft zwei Hinkelsteine* sind *Ein Gallier* und *verkauft zwei Hinkelsteine* Konstituenten. Weiterhin sind *verkauft* und *zwei Hinkelsteine* Konstituenten von *verkauft zwei Hinkelsteine*.

Eine Konstituentenstruktur erhält man, indem ein Ausdruck immer in seine längsten Konstituenten zerlegt wird, diese dann wieder in ihre Längsten und so weiter, bis man zu den kürzesten Konstituenten des Ausdrucks gelangt. Da als Konstituenten nur bestimmte Teile von Sätzen gelten, stellt sich die Frage, wie diese ermittelt werden können. *Verkauft zwei* würde beispielsweise nicht als Konstituente gelten. Offenbar stehen die unmittelbaren Konstituenten, die aus mehr als einem Wort bestehen, in einem engen syntaktischen Zusammenhang. Durch das Verschieben von Wortfolgen, durch Ersetzungsproben und an der Tatsache, daß Konstituenten oft Antwort auf eine Frage sein können, können diese durch Probieren ermittelt werden. Diese Tests zum Ermitteln von Konstituentenstrukturen führen nicht immer zu einer eindeutigen Lösung, oft sind alternative Strukturen möglich. Die Konstituentenstruktur von *Ein Gallier verkauft zwei große Hinkelsteine* kann als Baum dargestellt werden (Abbildung 3.1).

Die Konstituenten können in Klassen eingeteilt werden, die als *syntaktische Kategorien*

[2] *Die Generalisierte Phrasenstruktugrammatik:* siehe [6] und [11]
[3] siehe [8], [3] und [20]
[4] *Die Kopfgesteuerte Phrasenstrukturgrammatik:* siehe [22] und [19]

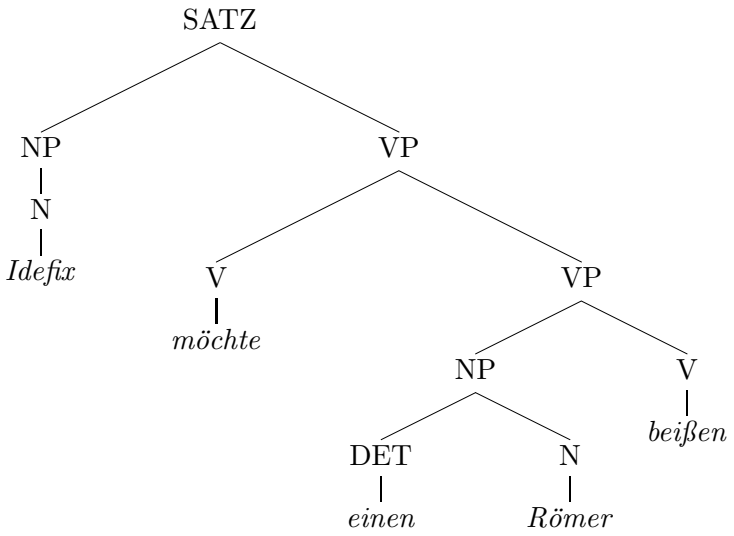

Abbildung 3.2: Die Konstituentenstruktur von Idefix möchte einen Römer beißen *mit an den Knoten notierten syntaktischen Kategorien.*

bezeichnet werden. In einer syntaktischen Kategorie werden Konstituenten zusammengefasst, die innerhalb eines Satzes immer gleiche Funktionen erfüllen. Beispiele für syntaktische Kategorien sind Nomen (N), Adjektive (ADJ), Verben (V) sowie alle anderen Wortarten, Nominalphrasen (NP) und Verbalphrasen (VP), die als zentralen Bestandteil ein Nomen bzw. ein Verb haben und Determinierer (DET), zu denen Artikel, Quantifizierer sowie determinierende Pronomen gehören. NOM bezeichnet einen Nominalkomplex, der das Nomen mit seinem Modifizierern enthält. Die Bäume der Konstituentenstrukturen werden an ihren Knoten mit den entsprechenden syntaktischen Kategorien garniert. *Idefix möchte einen Römer beißen* (Abbildung 3.2) und *Der Barde spielt gern Harfe* (Abbildung 3.3) zeigen deren Konstituentenstrukturen mit ihren Garnierungen.

Ein Ausdruck einer syntaktischen Kategorie kann auf verschiedene Art verwendet werden. Die Nominalphrase *die Gallierin* kann als Subjekt verwendet werden, jedoch auch als direktes Objekt. Die *syntaktische Funktion* der Nominalphrase ist in diesen Fällen entweder Subjekt oder direktes Objekt. Einige Beispiele sollen zeigen, wie die *semantischen Rollen* die Syntax eines Satzes mit der Semantik verknüpfen.

In *Gutemine schläft* ist *Gutemine* das Argument der allgemeinen semantischen Struktur und hat die Bedeutung „Gutemine“ als Person, deren semantische Rolle „Schlafende“ ist. *Schläft* ist die Relation der semantischen Struktur und hat die Tätigkeit „schlafen“ als Bedeutung. Bei *Obelix gibt Idefix einen Knochen* verteilen sich die semantischen Rollen wie in Tabelle 3.1.

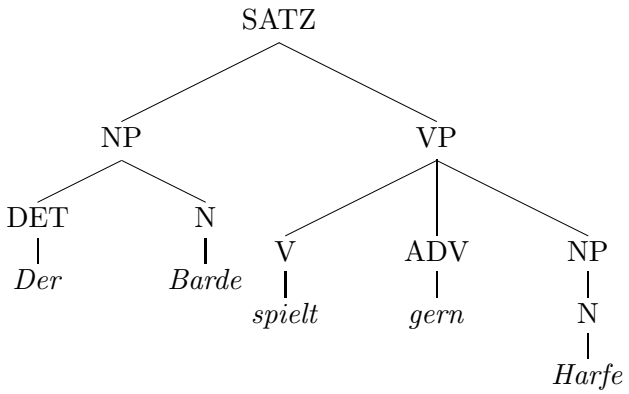

Abbildung 3.3: Die Konstituentenstruktur von Der Barde spielt gern Harfe.

Ausdruck	*Obelix*	*gibt*	*Idefix*	*einen Knochen*
allg. semantische Struktur	Argument	Relation	Argument	Argument
spezifische Bedeutungen	„Obelix“	„geben“	„Idefix“	„Knochen“, Anz.: 1
semantische Rollen	Gebender		Gebendem	Gegebenes

Tabelle 3.1: Die semantischen Rollen in Obelix gibt Idefix einen Knochen.

Die semantischen Rollen werden oft aus den syntaktischen Regeln einer Sprache ersichtlich. Ein Verb erwartet sein zugehöriges Subjekt und seine direkten und indirekten Objekte zum Beispiel in einer bestimmten Reihenfolge und in einem speziellen Kasus. Diese Merkmale werden in den Regeln der Grammatik festgelegt, so daß sie bei der Analyse oder der Generierung eines Satzes berücksichtigt werden können. So können semantische Merkmale oft schon bei der Untersuchung eines Satzes erkannt werden. Allerdings gibt es Sätze, bei denen die semantischen Rollen nicht ganz klar sind, bzw. bei denen die Bedeutung des Satzes nicht eindeutig ist. *Der Krieger bewirft den Druiden mit dem Hinkelstein* ist ein Beispiel für einen solchen Satz[5].

Nimmt man nun eine Konstituentenstruktur und läßt ihre untersten Knoten, die Wörter, weg, bekommt man eine *verallgemeinerte Konstituentenstruktur*. Mit dieser kann nun eine ganze Klasse von Sätzen beschrieben werden. Die Sätze *Idefix möchte einen Römer beißen* und *Tullius Destructivus will die Gallier reinlegen* haben z.B. die gemeinsame verallgemeinerte Konstituentenstruktur wie in Abbildung 3.4. [16]

[5] vergleiche Kapitel 3.2.1, Abbildung 3.9 und 3.10

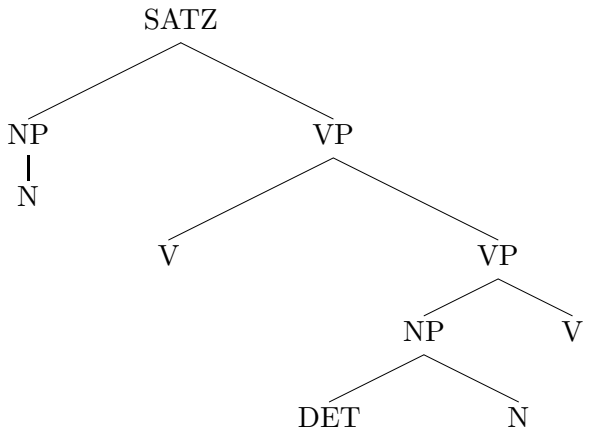

Abbildung 3.4: Eine verallgemeinerte Konstituentenstruktur.

3.2 Phrasenstruktursyntaxen

Als Phrasensturktursyntaxen werden Grammatiken bezeichnet, die die Struktur von Sätzen anhand von Phrasen beschreiben.[6] Um alle möglichen Sätze einer Sprache beschreiben zu können, würde es also reichen, alle möglichen Konstituentenstrukturen aufzuzeigen. Diese Beschreibung muß auf jeden Fall endlich sein, da sie mit einer Maschine verarbeitet werden soll, die mit endlichen Resourcen auskommen muß. Die Menge dieser Strukturen ist jedoch sehr groß und nach Belieben erweiterbar, demnach also nicht endlich (man bedenke, daß z.B. ein Substantiv durch beliebig viele Adjektive erweiterbar ist). Eine bloße Auflistung aller möglichen Konstituentenstrukturen ist somit keine Lösung. Anstatt einer endlosen Liste von Konstituentenstrukturen kann ein Regelsystem aufgestellt werden, mit dem man nun Sätze *ableiten* bzw. *generieren* kann. Ein solches Regelsystem habe ich bereits in Kapitel 2.3 vorgestellt: die formalen Sprachen. [16]

3.2.1 Kontextfreie Phrasenstruktursyntaxen

Eine *kontextfreie Phrasenstruktursyntax* besteht nun aus den endlichen Mengen der *Terminalsymbole*, der *Nicht-Terminalsymbole*, der *kontextfreien Produktionsregeln* sowie einem ausgezeichneten *Startsymbol* aus der Menge der Nicht-Terminale; es handelt sich also um eine kontextfreie Grammatik. Die Menge der Terminalsymbole beinhaltet dann die Wörter der Sprache, die der Nicht-Terminalsymbole beinhaltet die syntaktischen Kategorien. Das Startsymbol schließlich ist der abzuleitende Satz. Die Produktionsregeln erstellt man

[6] Eine Phrase (engl. phrase) ist eine andere Bezeichnung für eine Konstituente.

mit Hilfe der Konstituentenstrukturen so, daß möglichst viele Sätze der zu beschreibenden Sprache ableitbar bzw. generierbar sind[7].

Einige erste Produktionsregeln erhält man, indem man sich die *lokalen Bäume* einer ersten Konstituentenstruktur ansieht. Als einen lokalen Baum bezeichnet man eine Konstituente (den *Mutter-* oder *Vater-Knoten*) samt aller ihrer direkten Nachfolger (die *Tochter-* oder *Sohn-Knoten*), im Folgenden mit *Eltern-* und *Kind-Knoten* benannt. Ein Eltern-Knoten trägt den Namen einer syntaktischen Kategorie und ein Kind-Knoten kann auch mit Wörtern garniert sein, siehe Abildung 3.5.

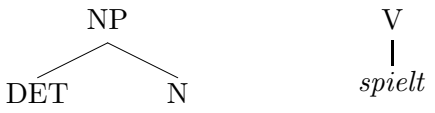

Abbildung 3.5: Zwei Teilbäume aus der Konstituentenstruktur in Abbildung 3.3.

Die lokalen Bäume können nun mit Regeln der Form $A \rightarrow B_1 \dots B_n$ beschrieben werden, wobei $A \in \mathcal{N}$ und $B_{1\dots n} \in (\mathcal{T} \cup \mathcal{N})$ sind. Da es sich um kontextfreie Regeln handelt, darf auf der linken Seite immer nur *ein* Symbol stehen. Dies ist zu lesen als „A besteht aus B_1 und $\dots$ und B_n“. Die zwei Regeln in Abbildung 3.5 lauten dementsprechend:

$$\begin{aligned} \text{NP} &\rightarrow \text{DET N} \\ \text{V} &\rightarrow \textit{spielt} \end{aligned}$$

Dazu kommen die aus Abbildung 3.3 ableitbaren Regeln:

$$\begin{aligned} \text{SATZ} &\rightarrow \text{NP VP} \\ \text{VP} &\rightarrow \text{V ADV NP} \\ \text{NP} &\rightarrow \text{N} \end{aligned}$$

Zuzüglich der Regeln der Struktur aus Abbildung 3.2 ergibt sich das folgende Regelsystem. Die verschiedenen NP und VP auf der linken Seite der Gleichungen in unserem Regelsystem sind als Alternativen zu sehen. Weiterhin kann im Deutschen ein Verb allein als VP gelten, wie auch eine NP als N gilt:

$$\begin{aligned} \text{SATZ} &\rightarrow \text{NP VP} \\ \text{NP} &\rightarrow \text{DET N} \end{aligned}$$

[7] Im Optimalfall wäre es natürlich wünschenswert, *alle* Sätze einer solchen Sprache ableiten zu können. Dies wird aber sicherlich aufgrund der Komplexität der natürlichen Sprachen nur schwer möglich sein. Ich werde mich hier also auf einige Beispiele beschränken und später die Bedürfnisse von *ScienceAtlas* berücksichtigen. Dennoch sind Regelsysteme dieser Art sehr leistungsfähig.

$$
\begin{aligned}
\text{NP} &\rightarrow \text{N} \\
\text{VP} &\rightarrow \text{V ADV NP} \\
\text{VP} &\rightarrow \text{V VP} \\
\text{VP} &\rightarrow \text{NP V} \\
\text{VP} &\rightarrow \text{V}
\end{aligned}
$$

Mit diesen Regeln kann nun schon eine ganze Anzahl deutscher Sätze beschrieben werden. Die Menge der generierbaren Strukturen wächst mit Hinzunahme neuer Regeln schnell an. Erweitert man das System um einige rekursive Regeln und führt den Nominalkomplex (NOM) ein, ändert sich das endliche Regelsystem wie folgt und kann nun eine unendliche Menge von Strukturen beschreiben:

$$
\begin{aligned}
\text{NP} &\rightarrow \text{DET NOM} \\
\text{NOM} &\rightarrow \text{NOM PP} \\
\text{NOM} &\rightarrow \text{N} \\
\text{NOM} &\rightarrow \text{ADJ NOM} \\
\text{PP} &\rightarrow \text{P NP}
\end{aligned}
$$

Wie in Abbildung 3.6 auf Seite 17 zu sehen ist, können nach Belieben weitere Präpositionalphrasen (PP) sowie Nominalkomplexe mit Adjektiven (ADJ NOM) eingefügt werden. Dabei nennt man Regeln der Form $\text{A} \rightarrow \text{A}\sigma$ *linksrekursiv* und Regeln der Form $\text{A} \rightarrow \sigma\text{A}$ *rechtsrekursiv*. Wenn solche Regeln existieren, muß auf jeden Fall gewährleistet sein, daß die Rekursion irgendwann terminieren kann. Mindestens eine Regel der Form $\text{A} \rightarrow \tau$ muß also existieren.

Ein kontextfreies Regelssystem wird nun zum *Ableiten* der Symbole benutzt, was nichts anderes heißt, als daß gewisse Symbole durch andere ersetzt bzw. ausgetauscht werden dürfen, um die gegebene Struktur zu verändern. Einmaliges Ableiten, also das einmalige Ersetzen eines Symbols, wird als *Ableitungsschritt* bezeichnet. Eine Reihe aufeinander folgender Ableitungsschritte wird eine *Ableitung* genannt. Kann man von einer Symbolfolge α durch nur einen Ableitungsschritt zu einer Symbolfolge β kommen, schreibt man $\alpha \Rightarrow \beta$ und sagt, β ist *direkt ableitbar* von α. Ist es möglich, durch mehrere Ableitungsschritte von α nach β zu gelangen, redet man von *indirektem* Ableiten und schreibt $\alpha \overset{*}{\Rightarrow} \beta$.

Als Beispiel hier zwei Ableitungen mit der kontextfreien Syntax $\mathcal{G}_1$ in Abbildung 3.7:

1. DET NOM V $\Rightarrow$ DET NOM *trinkt* $\Rightarrow$ *der* NOM *trinkt* $\Rightarrow$ *der* N *trinkt* $\Rightarrow$ *der Normanne trinkt*

   Daraus folgt: DET NOM V $\overset{*}{\Rightarrow}$ *der Normanne trinkt*

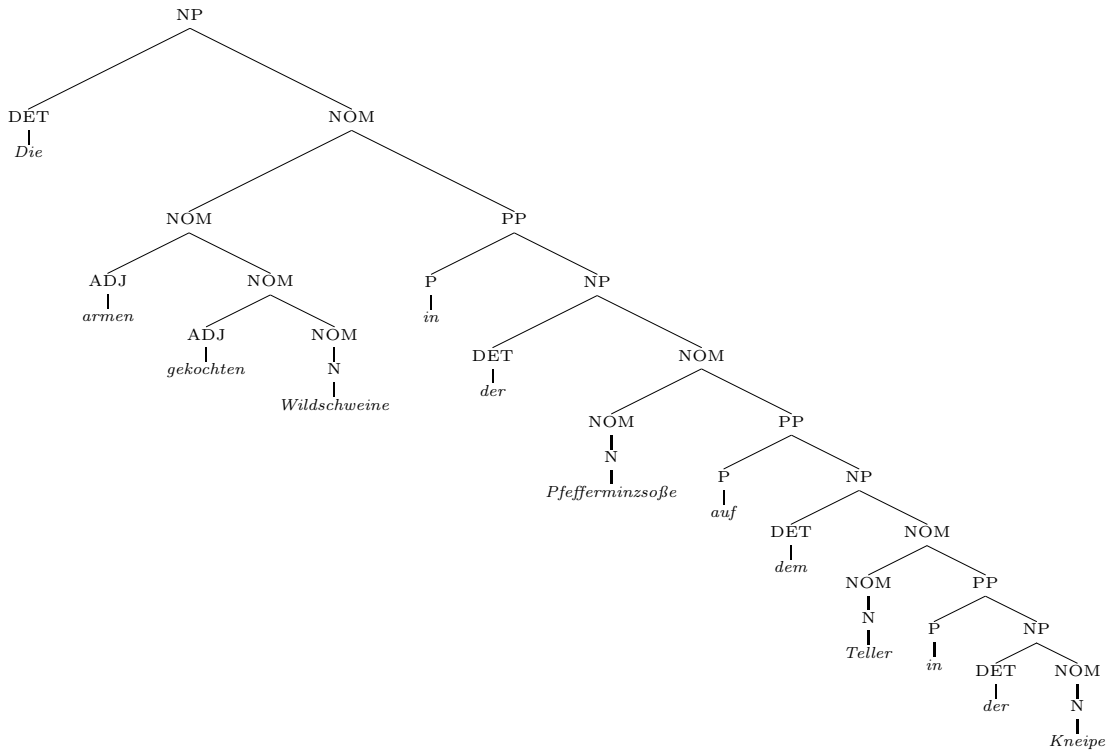

Abbildung 3.6: Eine beliebig erweiterbare Konstituentenstruktur einer Nominalphrase: Die armen gekochten Wildschweine in der Pfefferminzsoße auf dem Teller in der Kneipe.

$$
\begin{aligned}
\mathcal{G}_1 = \Big(\mathcal{T} &= \{\textit{Maulaf, ängstliche, Normanne, trinkt, der}\},\\
\mathcal{N} &= \{\text{SATZ, NP, VP, NOM, DET, ADJ, V, N}\},\\
\mathcal{S} &= \text{SATZ},\\
\mathcal{P} &= \{\text{SATZ} \rightarrow \text{NP VP},\\
&\quad\ \text{NP} \rightarrow \text{DET NOM},\\
&\quad\ \text{NOM} \rightarrow \text{N},\\
&\quad\ \text{NOM} \rightarrow \text{ADJ NOM},\\
&\quad\ \text{VP} \rightarrow \text{V},\\
&\quad\ \text{DET} \rightarrow \textit{der},\\
&\quad\ \text{N} \rightarrow \textit{Normanne},\\
&\quad\ \text{ADJ} \rightarrow \textit{ängstliche},\\
&\quad\ \text{V} \rightarrow \textit{trinkt}\} \Big)
\end{aligned}
$$

Abbildung 3.7: Die kontextfreie Syntax $\mathcal{G}_1$.

2. SATZ $\Rightarrow$ NP VP $\Rightarrow$ DET NOM VP $\Rightarrow$ DET ADJ NOM VP $\Rightarrow$ DET ADJ NOM V $\Rightarrow$ DET ADJ N V $\Rightarrow$ *der* ADJ N V $\Rightarrow$ *der ängstliche* N V $\Rightarrow$ *der ängstliche Normanne* V $\Rightarrow$ *der ängstliche Normanne trinkt*

   Daraus folgt: SATZ $\overset{*}{\Rightarrow}$ *der ängstliche Normanne trinkt*

Es kann nun sein, daß für eine kontextfreie Syntax $\mathcal{G}$ und einen sprachlichen Ausdruck σ aus $\mathcal{L}(\mathcal{G})$ verschiedene Ableitungen existieren, da es für die Ergebniskette verschiedene Ableitungswege gibt. Gilt dies für mindestens ein Element von σ, so ist die Syntax *mehrdeutig*. Am Beispiel der Syntax $\mathcal{G}_2$ in Abbildung 3.8 zeigt sich, daß zwei Wege zu der Ergebniskette

DET N V DET N P DET N

führen, wie in Abbildung 3.9 und 3.10 zu sehen ist. Der Satz *Der Krieger bewirft den Druiden mit dem Hinkelstein* aus Kapitel 3.1 ist in diesem Sinne mehrdeutig. [16] [30]

3.2.2 Kontextsensitive Phrasenstruktursyntaxen

Kontextsensitive Syntaxen können zur Beschreibung von natürlichen Sprachen genutzt werden, zum Beispiel zur Subkategorisierung[8] von Verben oder zur Beschreibung von Kongru-

[8] siehe Kapitel 3.2.5

$$
\begin{aligned}
\mathcal{G}_2 = \Big(\mathcal{T} &= \{\text{DET, N, V, P}\},\\
\mathcal{N} &= \{\text{SATZ, NP, PP, VP}\},\\
\mathcal{S} &= \text{SATZ},\\
\mathcal{P} &= \{\text{SATZ} \rightarrow \text{NP VP},\\
&\quad\ \text{NP} \rightarrow \text{Det NOM},\\
&\quad\ \text{NOM} \rightarrow \text{N},\\
&\quad\ \text{NOM} \rightarrow \text{NOM PP},\\
&\quad\ \text{PP} \rightarrow \text{P NP},\\
&\quad\ \text{VP} \rightarrow \text{V NP},\\
&\quad\ \text{VP} \rightarrow \text{V NP PP}\} \Big)
\end{aligned}
$$

Abbildung 3.8: Die kontextfreie Syntax $\mathcal{G}_2$.

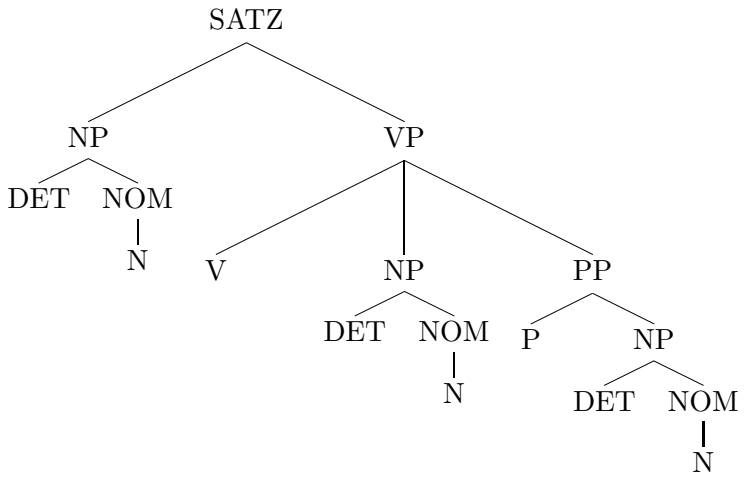

Abbildung 3.9: Ein möglicher Ableitungsbaum für die Grammatik $\mathcal{G}_2$.

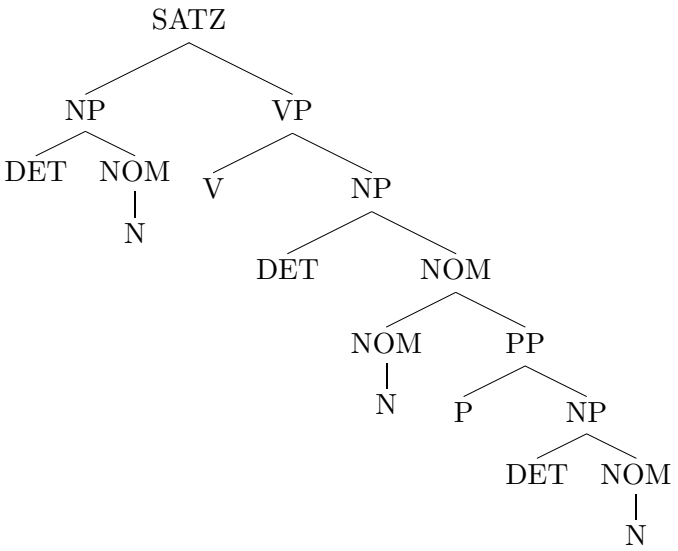

Abbildung 3.10: Ein alternativer Ableitungsbaum für die Grammatik $\mathcal{G}_2$.

enz. Wenn auf ein transitives[9] Verb direkt eine Nominalphrase folgt, kann dies durch die folgende Regel beschrieben werden und bedeutet: „V wird zu $\mathrm{V}_{transitiv}$, wenn NP direkt auf V folgt“. Die wiederzuschreibende Konstituente, hier das V, steht dann an der Stelle „___“:

$$\mathrm{V} \quad \rightarrow \quad \mathrm{V}_{transitiv}/\_\_\_\ \mathrm{NP}$$

Diese Regel kann auch in der Form

$$\mathrm{V}\ \ \mathrm{NP} \quad \rightarrow \quad \mathrm{V}_{transitiv}\ \mathrm{NP}$$

geschrieben werden und wird *kontextsensitive Regel* genannt. Auf der linken Seite steht nun eine Folge von Konstituenten, die durch eine andere Folge (die auf der rechten Seite) zu ersetzen ist. Es wird hier nicht die komplette Folge, sondern nur eine einzelne Konstituente ersetzt.

Kontextsensitive Regeln erschweren die automatische Analyse einer Sprache. Viele Strukturen natürlicher Sprache, die mit kontextsensitiven Regeln beschrieben werden können, sind auch mit kontextfreien Regeln beschreibbar. [16]

[9] Ein *transitives* Verb besitzt ein oder mehrere Objekte, wie zum Beispiel *Methusalix gibt Idefix einen Knochen*. Methusalix gibt jemandem etwas. Ein *intransitives* Verb beschreibt eine Handlung, die kein Objekt erlaubt, z.B. *Der Kessel fällt*. Der Kessel kann selber fallen, doch nicht jemandem etwas fallen.

3.2.3 Probleme bei der Beschreibung natürlicher Sprachen mit kontextfreien Syntaxen

Syntaktische Funktionen

Mit kontextfreien Syntaxen werden die Konstituentenstrukturen natürlichsprachlicher Ausdrücke dargestellt. Hierbei werden die Baumknoten mit den syntaktischen Kategorien garniert. Die syntaktischen Funktionen wie *ist-Subjekt-von*, *ist-Prädikat von* und *ist-Objekt-von* werden nicht explizit angegeben, können jedoch in vielen Fällen der Darstellung des Baumes entnommen werden. Das Subjekt ist als unmittelbare Konstituente des gesamten Satzes gegeben (NP), das Prädikat des Satzes ist gekennzeichnet durch das Verb (V) und die Objekte des Verbs sind Konstituenten der Verbalphrase (VP). Für eine korrekte semantische Analyse müssen jedoch auch alle weiteren syntaktischen Funktionen der einzelnen Konstituentenstrukturen klar definiert sein. Aus der Struktur des Baumes sind diese nicht immer zu entnehmen.

Kongruenz

Kongruenz bedeutet formal gesehen Übereinstimmung von Konstituenten hinsichtlich ihrer Merkmale. Diese tritt in vielen Sprachen auf. In romanischen Sprachen muß im Allgemeinen ein Determinierer mit seinem determinierten Substantiv in Genus und Numerus übereinstimmen, im Deutschen muß außerdem noch der Kasus beachtet werden. Zum Beispiel heißt es *Der Misteln schneidende Druide mit der Sichel des berühmten Sichelhändlers Talentix*, aber *Die Misteln schneidenden Druiden mit den Sicheln der berühmten Sichelhändler Talentix und Gibtesnix*. Mit der kontextfreien Regel

$$\mathrm{NP} \rightarrow \mathrm{DET}\ \mathrm{N}$$

sowie allen Determinierern und Substantiven dieser Sprache wären *alle* Kombinationen erzeugbar, somit auch alle nicht korrekten. Die Regeln könnten nun so verfeinert werden, daß alle korrekten Fälle erzeugbar sind, da die Anzahl der Möglichkeiten begrenzt ist. Die folgenden Regeln beschreiben die Kongruenz zwischen Determinierer und Substantiv für das Deutsche[10]:

$$\begin{aligned}
\mathrm{NP} &\rightarrow \mathrm{DET}_{MaSg}\ \mathrm{N}_{MaSg} \\
\mathrm{NP} &\rightarrow \mathrm{DET}_{MaPl}\ \mathrm{N}_{MaPl} \\
\mathrm{NP} &\rightarrow \mathrm{DET}_{FeSg}\ \mathrm{N}_{FeSg}
\end{aligned}$$

[10] Hier sind die drei Genera der deutschen Sprache bezeichnet mit *Ma* für Maskulin, *Fe* für Feminin und *Ne* für Neutrum, sowie die zwei Numeri mit *Sg* für Singular und *Pl* für Plural. Den Kasus mit seinen vier Varianten Nominativ, Akkusativ, Genitiv und Dativ habe ich der Übersichtlichkeit halber erst später hinzugefügt. Aus jeder dieser sechs Regeln würden dann vier werden, womit man auf 24 Regeln nur für die Beschreibung von Determinierer und Substantiv führen würde!

$$
\begin{aligned}
\text{NP} &\rightarrow \text{DET}_{FePl}\ \text{N}_{FePl} \\
\text{NP} &\rightarrow \text{DET}_{NeSg}\ \text{N}_{NeSg} \\
\text{NP} &\rightarrow \text{DET}_{NePl}\ \text{N}_{NePl}
\end{aligned}
$$

Diese Art der Beschreibung und Verfeinerung müßte nun auf alle Fälle von Kongruenz in der beschriebenen Sprache angewendet werden. Eine Flut von neuen Regeln wäre die Folge. Eine Erweiterung der kontextfreien Syntaxen ist hier sinnvoller, wie in Kapitel 3.2.5 vorgeschlagen wird.

Permutationen

Aus der variablen Position der Konstituenten innerhalb eines Satzes ergeben sich weitere Probleme, was die Anzahl der Permutationen dieser Konstituenten – also die Menge der Anordnungsmöglichkeiten – angeht. Der Satz *Cäsar verschenkt ein Dorf* läßt sich bequem in seine Konstituenten zerlegen. In den Sätzen *Verschenkt Cäsar ein Dorf?*, *Letzte Woche hat Cäsar ein Dorf verschenkt* und *Hat Cäsar letzte Woche ein Dorf verschenkt?* sind die Konstituenten der Verbalphrase durch das Subjekt voneinander getrennt. In einem Baum würden sich hier die Kanten überschneiden. Durch Hinzunahme weiterer Regeln, die eine Aufteilung von SATZ in Deklarativ- und Fragesatz erlauben, könnte dieses Problem gelöst werden:

$$
\begin{aligned}
\text{SATZ} &\rightarrow \text{SATZ}_D \\
\text{SATZ} &\rightarrow \text{SATZ}_F \\
\text{SATZ}_D &\rightarrow \text{NP}\ \ \text{VP} \\
\text{SATZ}_F &\rightarrow \text{V}\ \ \text{NP}\ \ \text{NP}
\end{aligned}
$$

Um nun alle Permutationen abzudecken, müßte wiederum eine weitere Flut von Regeln erdacht werden. Als Lösung hierfür bieten erweiterte Phrasenstruktursyntaxen wie die Generalisierte Phrasenstrukturgrammatik (GPSG) oder die Kopfgesteuerte Phrasenstrukturgrammatik (HPSG) diverse Möglichkeiten. [1] [16]

3.2.4 Merkmalsmengen

Durch einige Modifikationen kann die Regelflut aus den vorgehenden Abschnitten ein wenig eingedämmt werden. Das Lexikon, das bis jetzt nur die Terminale und ihre syntaktischen Kategorien der zu behandelnden Sprachen enthält, wird nun in der Form erweitert, daß noch weitere Informationen zu den einzelnen Wörtern ergänzt werden. Die Wörter werden

$$
\textit{der}\ \text{DET}\ \begin{bmatrix} \text{NUM} & \text{SG} \\ \text{GEN} & \text{MAS} \\ \text{KAS} & \text{NOM} \end{bmatrix}, \quad
\textit{die}\ \text{DET}\ \begin{bmatrix} \text{NUM} & \text{SG} \\ \text{GEN} & \text{FEM} \\ \text{KAS} & \text{NOM} \end{bmatrix},
$$

$$
\textit{das}\ \text{DET}\ \begin{bmatrix} \text{NUM} & \text{SG} \\ \text{GEN} & \text{NEU} \\ \text{KAS} & \text{NOM} \end{bmatrix}, \quad
\textit{Gallier}\ \text{N}\ \begin{bmatrix} \text{NUM} & \text{SG} \\ \text{GEN} & \text{MAS} \\ \text{KAS} & \text{NOM} \end{bmatrix},
$$

$$
\textit{Römerin}\ \text{DET}\ \begin{bmatrix} \text{NUM} & \text{SG} \\ \text{GEN} & \text{FEM} \\ \text{KAS} & \text{NOM} \end{bmatrix}, \quad
\textit{Schild}\ \text{DET}\ \begin{bmatrix} \text{NUM} & \text{SG} \\ \text{GEN} & \text{NEU} \\ \text{KAS} & \text{NOM} \end{bmatrix}
$$

Abbildung 3.11: Einige Wörter eines Lexikons der deutschen Sprache.

nun hinsichtlich ihrer lautlichen, morphologischen, syntaktischen und semantischen Eigenschaften beschrieben.

Die Kongruenzmerkmale Genus, Numerus und Person sowie Subkategorisierungseigenschaften wie Transitivität und Intransitivität gehören als syntaktische Eigenschaften zu diesen Erweiterungen. Diese werden in Form von *Merkmalsspezifikationen* dargestellt. Eine Merkmalsspezifikation ist ein Paar $[M, W]$, wobei M ein Merkmal und W der dazugehörige Wert ist. Hat W den Wert „+“ oder „-“, besitzt ein Wort dieses Merkmal oder nicht. Eine Menge von Merkmalsspezifikationen für ein sprachliches Element heißt *Merkmalsmenge* oder *Merkmalsstruktur*. Eine Merkmalsmenge mit den Merkmalen M_i und den dazugehörigen Werten W_i mit $1 \leq i \leq n$:

$$
\begin{bmatrix} M_1 & W_1 \\ \vdots & \vdots \\ M_n & W_n \end{bmatrix}
$$

Ein Wert W muß genau einem Merkmal M zugeordnet sein. Bei Substantiven, Adjektiven und Determinierern werden Genus (GEN), Numerus (NUM) und im Deutschen der Kasus (KAS) als Kongruenzmerkmale angegeben, bei Substantiven und Personalpronomen auch die Person (PER). Diese Merkmalsstrukturen werden nun in die Regeln der kontextfreien Grammatik integriert.

Anstatt wie in Kapitel 3.2.3 viele neue Regeln zu erstellen, werden sowohl die Terminale als auch die Nicht-Terminale mit den Merkmalsstrukturen verknüpft. Da man nun wieder sehr viele Regeln erhalten würde, stellt man ein *Regelschema* auf, welches sicherstellt, daß die Konstituenten nun zueinander passende Merkmalsstrukturen aufweisen. Die einzelnen

$$\text{NP} \rightarrow \underset{\begin{bmatrix}\text{NUM} & \alpha\\ \text{GEN} & \beta\\ \text{KAS} & \gamma\end{bmatrix}}{\text{DET}} \quad \underset{\begin{bmatrix}\text{NUM} & \alpha\\ \text{GEN} & \beta\\ \text{KAS} & \gamma\end{bmatrix}}{\text{NOM}},$$

$$\underset{\begin{bmatrix}\text{NUM} & \alpha\\ \text{GEN} & \beta\\ \text{KAS} & \gamma\end{bmatrix}}{\text{NOM}} \rightarrow \underset{\begin{bmatrix}\text{NUM} & \alpha\\ \text{GEN} & \beta\\ \text{KAS} & \gamma\end{bmatrix}}{\text{N}},$$

$$\underset{\begin{bmatrix}\text{NUM} & \alpha\\ \text{GEN} & \beta\\ \text{KAS} & \gamma\end{bmatrix}}{\text{NOM}} \rightarrow \underset{\begin{bmatrix}\text{NUM} & \alpha\\ \text{GEN} & \beta\\ \text{KAS} & \gamma\end{bmatrix}}{\text{N}} \quad \underset{\begin{bmatrix}\text{NUM} & \alpha\\ \text{GEN} & \beta\\ \text{KAS} & \gamma\end{bmatrix}}{\text{ADJ}}$$

Abbildung 3.12: Einige Produktionsregeln mit um Merkmalsmengen erweiterten Kategorien.

möglichen Werte werden nun nicht mehr alle einzeln aufgeführt (siehe Abbildung 3.12), sondern mit Variablen versehen, wobei gleiche Variablennamen gleiche Werte repräsentieren. Hier gilt $\alpha \in \{MAS, FEM, NEU\}$, $\beta \in \{SG, PL\}$ und $\gamma \in \{NOM, AKK, DAT, GEN\}$. Jedes der Schemata steht nun allerdings für jeweils 24 Regeln. Bevor ein solches Schema angewendet werden kann, müssen die Variablen α, β und γ mit Werten belegt werden. [1] [16]

3.2.5 Subkategorisierung

Durch diese Merkmalsspezifikation läßt sich auch die Subkategorisierung der Verben in Bezug auf deren Objekte beschreiben. Da es sehr viele Strukturen dieser Art gibt[11], reicht es nicht aus, mit Begriffen wie *transitiv* und *intransitiv* zu arbeiten. Eine Durchnummerierung der einzelnen Fälle ist an dieser Stelle sinnvoll, wie z.B. in der *Generalized Phrase Structure Grammar (GPSG)* [8] vorgeschlagen wird. Jedes Verb erhält im Lexikon ein Merkmal SUBCAT, das mit einer Zahl versehen ist. So definieren die Regeln für jedes Merkmal, wie die Verben verwendet werden können bzw. dürfen und welche Objekte oder andere Komplemente sie benötigen. Auf folgende Art und Weise werden alle Subkatgorien von Verben beschrieben:

[11] Strukturen mit präpositionalen Objekten, Nebensatzobjekten, Infinitivkonstruktionen, zwei Objekten oder obligatorischen und fakultativen Konstituenten

Im Lexikon	*schläft*	V[SUBCAT 1]
	nimmt	V[SUBCAT 2]
	gibt	V[SUBCAT 3]
	verspricht 1.	V[SUBCAT 3]
	verspricht 2.	V[SUBCAT 4]

In den Regeln	VP →	V[SUBCAT 1]			
	VP →	V[SUBCAT 2]	NP		
	VP →	V[SUBCAT 3]	NP	NP	
	VP →	V[SUBCAT 4]	NP	(PP[PFORM *a*])	VP[VFORM INF]

Erläuterung	[SUBCAT 1]:	*schläft*	bildet alleine eine VP: *Idefix schläft.*
	[SUBCAT 2]:	*nimmt*	erfordert min. eine NP als direktes Objekt: *Idefix nimmt den Knochen.*
	[SUBCAT 3]:	*gibt*	erfordert min. eine NP als direktes und eine NP als indirektes Objekt: *Obelix gibt Idefix einen Knochen.*
	[SUBCAT 3]:	*verspricht*	erfordert min. eine NP als direktes und eine NP als indirektes Objekt: *Obelix verspricht Idefix einen Knochen.*
	[SUBCAT 4]:	*verspricht*	fordert eine NP als direktes Objekt, eine optionale PP mit einer Präposition *a* sowie eine VP mit infinitiver Verbform: *Obelix verspricht Idefix(, in den Wald) zu gehen.*

Die Frage, ob die in Kapitel 3.2.4 und Kapitel 3.2.5 besprochenen Regeln noch kontextfrei sind, kann man getrost mit ja beantworten, denn ein Symbol einer syntaktischen Kategorie in Verbindung mit seiner Merkmalsstruktur kann immer noch als ein einziges Symbol gelten, auch wenn dieses sehr komplex ist. [16]

3.3 Die Lexikalisch-funktionale Grammatik (LFG)

Die Lexikalisch-funktionale Grammatik ist ein unifikationsbasiertes Grammatikmodell. Sie umfasst ein Lexikon, eine syntaktische und eine semantische Komponente und zusätzlich eine funktionale Ebene zur Beschreibung von syntaktischen Funktionen wie *Subjekt-von* oder *Objekt-von*.

Die syntaktische Komponente besteht aus einer Phrasenstrukturgrammatik, wie bereits in

Kapitel 3.2 behandelt. Die in den Regeln benutzten Symbole für die syntaktischen Kategorien sind durch funktionale Schemata annotiert, die es mit Hilfe der Lexikoneinträge erlauben, einer von der syntaktische Komponente erzeugten Konstituentenstruktur eines sprachlichen Zeichens eine *funktionale Struktur* zuzuordnen. Mit dieser funktionalen Struktur kann nun das Zeichen im Sinne einer logischen Kalkülsprache semantisch interpretiert werden.

3.3.1 Das Lexikon

Das Lexikon beschreibt die Wörter unter Angabe ihrer Wortart und ihrer semantischen Merkmale. Die dazugehörigen Argumente, falls vorhanden, werden ebenfalls beschrieben. Die Wortart wird durch ein einfaches Symbol notiert, weitere Merkmale des Wortes werden als Attribute in die Beschreibung aufgenommen. Für *leiht* in *Verleihnix leiht Automatix einen Fisch* mit den drei Argumenten *Verleihnix*, *Automatix* und *einen Fisch* sieht ein Lexikoneintrag aus wie folgt:

leiht:	V,	(↑PRÄD)	=	„leihen <(↑ $\overset{\text{Agens}}{\text{SUBJ}}$) (↑ $\overset{\text{Empfänger}}{\text{IOBJ}}$) (↑ $\overset{\text{Thema}}{\text{DOBJ}}$)>“	①
		(↑TEMPUS)	=	PRÄS	②
		(↑MODUS)	=	IND	③
		(↑SUBJ NUM)	=	SG	④
		(↑SUBJ PER)	=	3	⑤

Eine Gleichung der Form $(\uparrow F) = W$ oder $(\downarrow F) = W$ mit einem Attribut F und einem Wert W heißt *funktionales Schema*. Gleichung ② und ③ bezeichnen Tempus und Modus von *leiht*, Gleichung ④ und ⑤ bezeichnen Numerus und Person des Subjektes. Die erste Gleichung stellt die Relation „leihen“ mit ihren Argumenten und deren syntaktischen Funktionen Subjekt, indirektes Objekt und direktes Objekt dar. Dessen semantische Rollen sind Thema (einen Fisch), Empfänger (Automatix) und Agens (der Handelnde, in diesem Fall also Verleihnix). Ein solches Prädikat mit seinen syntaktischen Funktionen heißt *lexikalische Form*, wobei der Pfeil „↑“ bedeutet, daß die jeweilige Information an die syntaktische Struktur, in der das Wort verwendet wird – also in der Hierarchie der Konstituentenstruktur – „nach oben“ gereicht wird.

Lexikalische Regelmäßigkeiten können durch *lexikalische Regeln* beschrieben werden. Diese können aus einer lexikalischen Form eine andere lexikalische Form generieren. Wenn im Lexikon z.B. nur die lexikalischen Formen für das Aktiv vorhanden sind, kann eine solche Regel für das Passiv angegeben werden. So können auch Sätze im Passiv erkannt und generiert werden, ohne die jeweiligen Formen zusätzlich in das Lexikon aufzunehmen.

$$
f\colon \begin{bmatrix}
\text{PRÄD} & \text{„verleihen}\left\langle \overset{Agens}{\left(f\ \text{SUBJ}\right)}\ \overset{Thema}{\left(f\text{DOBJ}\right)}\ \overset{Empfänger}{\left(f\ \text{IOBJ}\right)} \right\rangle\text{“} \\
\text{SUBJ} & g\colon \begin{bmatrix} \text{PRÄD} & \text{„Fischhändler“} \\ \text{SPEZ} & \text{DEF} \\ \text{NUM} & \text{SG} \\ \text{GEN} & \text{MAS} \\ \text{PER} & 3 \\ \text{KAS} & \text{NOM} \end{bmatrix} \\
\text{DOBJ} & h\colon \begin{bmatrix} \text{PRÄD} & \text{„Fisch“} \\ \text{SPEZ} & \text{INDEF} \\ \text{NUM} & \text{SG} \\ \text{GEN} & \text{MAS} \\ \text{PER} & 3 \\ \text{KAS} & \text{AKK} \end{bmatrix} \\
\text{TEMPUS} & \text{PRÄS} \\
\text{MODUS} & \text{IND}
\end{bmatrix}
$$

Abbildung 3.13: Die F-Struktur des Satzes Der Fischhändler verleiht einen Fisch.

3.3.2 Die funktionale Struktur

Eine *funktionale Struktur* f oder auch *F-Struktur*, wird in Form einer *Attribut-Wert-Matrix* dargestellt, ähnlich der Merkmalsmengen aus Kapitel 3.2.4. Allerdings mit dem Unterschied, daß hier die Werte als Attribute F des Zeichens bezeichnet sind und W dessen Werte repräsentieren, die selbst wieder funktionale Strukturen sein dürfen. Die Attribute bezeichnen entweder syntaktische Funktionen oder grammatische Merkmale wie Numerus und Tempus. Dazu kommt PRÄD, das eine Liste der syntaktischen Funktionen der Komplemente und deren semantischen Rollen bereitstellt. Eine F-Struktur des Satzes *Der Fischhändler verleiht einen Fisch* ist in Abbildung 3.13 dargestellt.

Die Angaben zu dieser Attribut-Wert-Matrix stammen aus einer Lexikon-Notation ähnlich der von *leiht* auf Seite 26 mit den geforderten Komplementen und deren Eigenschaften, wobei das indirekte Objekt optional[12] ist, also in der F-Struktur nicht auftauchen muß. Die übrigen Angaben werden von den Lexikoneinträgen der weiteren Wörter geliefert. Hier einige Beispiele:

[12] Dies wird in der Grammatik festgelegt.

Fischhändler:	N,	(↑PRÄD)	=	„Fischhändler“
		(↑NUM)	=	SG
		(↑GEN)	=	MAS
		(↑PER)	=	3
		(↑KAS)	=	NOM
einen:	N,	(↑SPEZ)	=	INDEF
		(↑NUM)	=	SG
		(↑GEN)	=	MAS
		(↑KAS)	=	AKK
Fisch:	N,	(↑PRÄD)	=	„Fisch“
		(↑NUM)	=	SG
		(↑GEN)	=	MAS
		(↑PER)	=	3
		(↑KAS)	=	AKK

Die F-Struktur in Abbildung 3.13 wird mit Hilfe des Lexikons aus einer Konstituentenstruktur erzeugt, die durch die kontextfreien Regeln in Abbildung 3.14 beschrieben wird. Einige der Kategoriensymbole sind durch funktionale Schemata annotiert. Diese Schemata geben die syntaktische Funktion der Kategorie an dieser Stelle an. Die Gleichung (↑SUBJ)=↓ unter dem Symbol NP bedeutet, daß der Wert von SUBJ aus den Konstituenten von NP zu ermitteln ist. (↓KAS)=NOM zeigt an, daß der Kasus der Konstituente ein Merkmal der NP selbst ist. Die Gleichung ↑=↓ drückt aus, daß einer Konstituenten keine Funktion zugeordnet ist, diese aber die funktionale Information ihrer Teilkonstituenten übernimmt.

Mit dem Lexikoneintrag von *verleiht*, der schon in der F-Struktur in Abbildung 3.3.1 benutzt wurde, den übrigen Lexikoneinträgen und den kontextfreien Regeln in Abbildung 3.14 ergibt sich eine Konstituentenstruktur für *Der Fischhändler verleiht einen Fisch* wie in Abbildung 3.15, an deren Knoten die funktionalen Schemata aus dem Lexikon und aus den Produktionsregeln anotiert sind. Aus dieser Konstituentenstruktur oder kurz *K-Struktur* resultiert nun die F-Struktur, indem für das Subjekt und das direkte Objekt (SUB und DOBJ) alle Attribute des jeweiligen Teilbaumes zusammengeführt werden. Die funktionalen Informationen an den dominierten Knoten werden *unifizert*, wobei jedes auftretende Attribut eindeutig spezifiziert sein muß. Erhält man für ein Attribut mehr als einen Wert, ist die Stuktur nicht korrekt, die Unifikation schlägt fehl.

Die Unifikation wird von den Blättern bis zum Wurzelknoten durchgeführt. Dieses Verfahren für die Generierung der F-Struktur wird durch die *funktionalen Beschreibungen* for-

SATZ	→	NP (↑SUBJ) =↓ (↓KAS) = NOM	VP ↑=↓	
VP	→	V ↑=↓	(NP) (↑IOBJ) =↓ (↑KAS) = DAT	(NP) (↑DOBJ) =↓ (↑KAS) = AKK
NP	→	(DET) ↑=↓	N ↑=↓	

Abbildung 3.14: Die kontextfreien Produktionsregeln mit anotierten funktionalen Schemata.

malisiert.[13]

Eine F-Struktur muß drei Bedingungen erfüllen, um wohlgeformt zu sein. Mit diesen Bedingungen ermöglicht die Unifikation, Kongruenzfehler sowie funktionale Unvollständigkeit bzw. das Fehlen einer lexikalischen Form festzustellen:

1. **Vollständigkeit:** Eine F-Struktur muß allen syntaktischen Funktionen, die in den lexikalischen Formen aufgeführt sind, Werte zuweisen.

   Wenn „suchen <(↑SUBJ)(↑DOBJ)>“ in einer intransitiven VP auftritt, ist dies nach den Produktionsregeln zwar möglich, das Ergebnis ist jedoch funktional unvollständig.

2. **Kohärenz:** Umgekehrt darf eine F-Struktur nicht mehr syntaktische Funktionen spezifizieren, als durch die lexikalische Form gefordert sind.

   So darf „fallen <(↑SUBJ)>“ beispielsweise nicht in einer F-Struktur vorkommen, die außer dem Subjekt noch ein direktes Objekt benötigt.

3. **Konsistenz:** Jedes Attribut in einer Matrix muß genau einen Wert haben. [16]

[13] Dieses Verfahren ist detailliert beschrieben in [16].

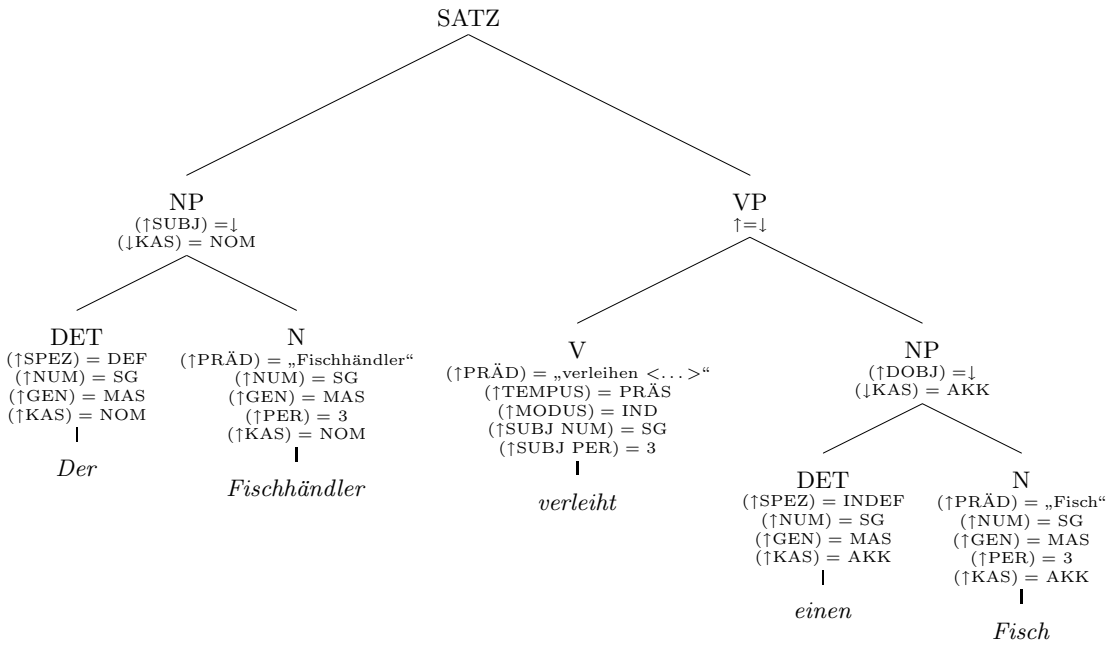

Abbildung 3.15: Die K-Struktur des Satzes Der Fischhändler verleiht einen Fisch.

4 Parsing natürlicher Sprache

Ein Parser ist ein System, das einem Satz eine Menge von Strukturbeschreibungen zuordnet – von denen bereits einige besprochen wurden – und dessen syntaktische Korrektheit überprüft. Eine Eingabe $\sigma \in (\mathcal{T} \cup \mathcal{N})^*$ wird als korrekt erkannt bzw. akzeptiert, wenn diese mit der verwendeten Grammatik $\mathcal{G}$ erzeugt werden kann. Dann gilt $\sigma \in \mathcal{L}(\mathcal{G})$, das heißt, σ ist Bestandteil der von $\mathcal{G}$ erzeugten Sprache $\mathcal{L}$.

Das Verhalten des Parsers wird durch den verwendeten Parsingalgorithmus und durch die zur Verfügung stehenden sprachlichen Informationen wie Syntax und Lexikon festgelegt. Der Begriff Parsing wird nicht nur im Zusammenhang mit der syntaktischen Analyse von Sprache verwendet, es gibt unter anderem auch morphologische Parser (für gesprochene Sprache), semantische Parser und Text-Parser. Als Parser beschreibe ich im Folgenden einige Verfahren zur Analyse einzelner Sätze einer Texteingabe.

4.1 Elementare Parsing-Algorithmen

Top-down-Parsing

Top-down-Parser arbeiten *zielgesteuert*. Sie beginnen die Analyse eines Satzes α mit dem Startsymbol der Syntax und versuchen durch *Expansion* den zu analysierenden Satz α zu erreichen. Ein Top-down-Algorithmus sieht wie folgt aus:

1. Beginne mit dem Startsymbol $\mathcal{S}$.
2. Wende eine Regel für das erste Nicht-Terminal an.
3. Lösche führende Terminale.
4. Wenn noch Terminale existieren, dann gehe zu 2.

Bottom-up-Parsing

Bottom-up-Algorithmen arbeiten *datengesteuert*. Sie gehen von α aus und versuchen durch *Reduktion* von α das Startsymbol zu erreichen. Ein solcher arbeitet wie folgt:

1. Untersuche das nächste Wort.

2. Wende eine Regel an, deren rechte Seite mit den letzten Symbolen des Satzes übereinstimmen und gehe zu 2.

3. Gehe zu 1.

Left-corner-Parsing

Diese Parser verknüpfen die beiden oben genannten Algorithmen. Die erste Teilkonstituente (*left corner*) einer jeden Konstituente wird bottom-up erkannt, alle übrigen top-down. Die Vorgehensweise ist folgende:

1. Die aktuelle Top-down-Erwartung ist der Satz α.

2. Unteruche das nächste Wort.

3. Suche eine Regel, deren *linke Ecke* mit der gefundenen Kategorie übereinstimmt. Die Kategorien der rechten Regelseite (außer der *linken Ecke*) werden verwendet, um die Top-down-Erwartung zu aktualisieren.

4. Gehe zu 2.

Beispiele

Sei $\mathcal{G}_3$ eine Syntax mit den Produktionsregeln

$$\begin{aligned}\mathcal{P} = \{\text{SATZ (r1)} &\rightarrow \text{NP VP},\\ \text{NP (r2)} &\rightarrow \text{N},\\ \text{VP (r3)} &\rightarrow \text{V NP PP},\\ \text{PP (r4)} &\rightarrow \text{P NP}\}\end{aligned}$$

und dem Lexikon

$$\mathcal{X} = \{\textit{Obelix}\ [\text{N}],\ \textit{Troubadix}\ [\text{N}],\ \textit{Wald}\ [\text{N}],\ \textit{sucht}\ [\text{V}],\ \textit{im}\ [\text{P}]\}.$$

Für *Obelix sucht Troubadix im Wald* erhält man die Konstituentenstruktur in Abbildung 4.1. Einen direkten Vergleich der einzelnen Arbeitsschritte der drei Algorithmen ist in Tabelle 4.1 zusammengestellt.

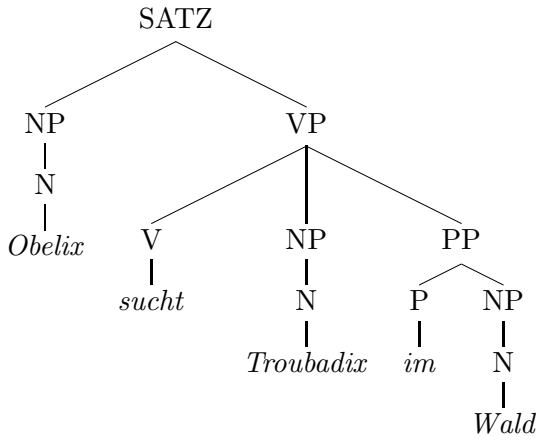

Abbildung 4.1: Konstituentenstruktur von Obelix sucht Troubadix im Wald.

Top-down-Analyse		Bottom-up-Analyse		Left-corner-Analyse	
Ableitung	Regel	Ableitung	Regel	Ableitung	Regel
SATZ	r1	N	r2	N	r2
NP VP	r2	NP		NP	r1
N VP		NP V		SATZ/VP	
VP	r3	NP V N	r2	V	r3
V NP PP		NP V NP		VP/NP PP	
NP PP	r2	NP V NP P		N	r2
N PP		NP V NP P N	r2	NP	
PP	r4	NP V NP P NP	r4	P	r4
P NP		NP V NP PP	r3	PP/NP	
NP	r2	NP VP	r1	N	r2
N		SATZ		NP	

Tabelle 4.1: Ein Vergleich der drei Algorithmen für die Top-down-, Bottom-up- und Left-corner-Analyse des Satzes Obelix sucht Troubadix im Wald. *[10]*

Probleme

Die oben genannten Algorithmen arbeiten nur dann korrekt, wenn Grammatik und Lexikon eine *deterministische* Analyse eines Satzes α erlauben. Es darf in jedem Schritt nur eine Regel anwendbar sein. Bei umfangreicheren Grammatiken ist dies selten der Fall. Erweitert man die Produktionsregeln der Syntax $\mathcal{G}_3$ um die folgenden, dann kann man für keinen der drei Algorithmen sicherstellen, daß alle Sätze korrekt analysiert werden:

$$\begin{aligned} \text{NP (r5)} &\rightarrow \text{DET N} \\ \text{VP (r6)} &\rightarrow \text{V NP} \end{aligned}$$

Dies ist auch schon in Grammatik $\mathcal{G}_2$ zu sehen. Die Wahl der falschen Regel führt zu einem vorzeitigen Scheitern der Analyse oder in eine Endlosschleife, da diese Entscheidung nicht rückgängig gemacht werden kann.

Als Lösungen hierfür sind zum einen die Depth-first-Search mit Backtracking (Tiefensuche mit Rücksprung) und zum anderen die Breadth-first-Search (Breitensuche) zu nennen. Bei einer Tiefensuche wird ein Pfad verfolgt, bis dieser nicht mehr weiter expandiert werden kann. Dann wird vom Startknoten aus der nächste Pfad verfolgt. Eine Breitensuche sortiert alle bereits untersuchten Pfade ihrer Länge nach und erweitert den kürzesten.

Die zwei genannten Algorithmen weisen zwei Schwächen auf. Zum einen ist die Klasse der Syntaxen, die sie verarbeiten können, recht eingeschränkt. Top-down-Parsing erlaubt keine links- bzw. rechtsrekursiven Regeln[1], Bottom-up- und Left-corner-Parsing erlauben keine Tilgungsregeln[2]. Zum anderen sind sie *ineffizient*, da bei allen drei Algorithmen nicht verhindert werden kann, daß in vielen Fällen eine Konstituente mehrfach auf dieselbe Art und Weise untersucht wird. [10] [21] [30]

4.2 Chart-Parser

Chart-Parser vermeiden Mehrfachanalysen, indem sie während der Analyse gewonnene Ergebnisse speichern, um darauf gegebenenfalls zurückgreifen zu können. Zur Speicherung wird oft eine sogenannte *Chart* verwendet. Eine Chart ist ein abstrakter Datentyp, der über seine Schnittstelle und somit über seine erzeugten Objekte, und über die Operationen auf diesen definiert ist.

Die in einer Chart gespeicherten Objekte werden als *Kanten* bezeichnet. Es gibt zwei Arten von Kanten. Die *aktiven* Kanten repräsentieren teilweise erkannte Konstituenten des

[1] Angenommen, die Eingabe wird von links nach rechts bearbeitet und es wird in einem Schritt dasjenige Nicht-Terminal ersetzt, das am weitesten rechts in der Eingabe steht. Eine rechtsrekursive Regel kann so das Programm in eine Endlosschleife führen.

[2] Regeln der Form $A \rightarrow \epsilon$ werden *Tilgungsregeln* genannt. Sie bewirken Tilgungen einer Struktur der Kategorie A und sind für linguistische Beschreibungen manchmal nützlich.

Satzes und die *passiven* Kanten die vollständig erkannten Konstituenten. Eine Kante enthält mindestens folgende Informationen:

1. die syntaktische Kategorie der repräsentierten Konstituente,
2. die Angabe des Satzteils, über den sich die Konstituente erstreckt und
3. bei aktiven Kanten die Spezifikation des bereits erkannten sowie des noch zu untersuchenden Abschnitts der Konstituente.

Die Positionen zwischen den Wörtern des Satzes werden zur Identifizierung durchnummeriert, die einzelnen Wörter bekommen Indizes. Ein Satz $\omega = \omega_1 \dots \omega_n$ wird repräsentiert durch einen Ausdruck wie:

$$0\omega_1 1\omega_2 2 \dots (n-1)\omega_n n.$$

Die Operationen, die auf einer Chart definiert sind, fügen neue Kanten in die Chart ein. Hier gibt es zwei Typen von Operationen: Die einen tragen passive Kanten auf Grundlage des verfügbaren lexikalischen Wissens in die Chart ein. Die anderen erzeugen aktive oder passive Kanten abhängig von schon in der Chart vorhandenen Kanten und dem verfügbaren syntaktischen Wissen.

Beispiel: Sei $k \in \mathcal{C}$ eine beliebige Kante einer Chart $\mathcal{C}$. k beinhaltet dann die folgenden Informationen und wird notiert als $[start(k), end(k), head(k), passivePart(k), activePart(k)]$:

1. $start(k) \in \mathbb{N}_0$ die Abfangsposition der Kante,
2. $end(k) \in \mathbb{N}_0$ die Endposition der Kante,
3. $head(k) \in \mathcal{N}$ den Typ der durch k repräsentierten Konstituente,
4. $passivePart(k) \in (\mathcal{T} \cup \mathcal{N})^*$ den erkannten Teil und
5. $activePart(k) \in (\mathcal{T} \cup \mathcal{N})^*$ den noch nicht erkannten Teil der durch k repräsentierten Konstituente.

4.2.1 Earley-Parser

Einer der wichtigsten Chart-Parsingalgorithmen ist der *Earley-Parser*, der von J. Earley entwickelt wurde und heute in vielen natürlichsprachlichen Systemen angewandt wird. Dieser Parser erlaubt es, beliebige kontextfreie Syntaxen zu verarbeiten und unterschiedlichste Suchstrategien zu realisieren.

Zur Repräsentation der Kanten werden *geteilte Produktionsregeln* verwendet, auch *dotted rules* genannt. Diese geben an, welche Ausdrücke vom Parser bereits erkannt wurden und

welche nicht. Produktionsregeln der Form $\alpha \rightarrow \beta.\gamma$ mit $\gamma = \epsilon$ repräsentieren passive Kanten, Regeln mit $\gamma \neq \epsilon$ bezeichnen aktive Kanten, wobei β dem geschlossenen Teil und γ dem offenen Teil der Kante entspricht. Beispiele für solche geteilten Regeln sind:

$$\begin{aligned} <2,5>\text{VP} &\rightarrow \text{V NP . NP} \\ {}_2[\text{VP} &\rightarrow \text{V NP}_5 \text{ NP}] \end{aligned}$$

Beide Regeln bedeuten, daß die bereits erkannten Konstituenten der Regel

$$\text{VP} \rightarrow \text{V NP . NP}$$

zwischen Position zwei und Position fünf liegen.

Ein Earley-Parser verwendet zur Berechnung der Chart drei Prozeduren, bezeichnet mit *Expand*, *Complete* und *Scan*. Die Prozedur Scan prüft, ob eine Eingabe den erwarteten terminalen Kategorien entspricht. Mit der Prozedur Expand werden alle nicht-terminalen Kategorien so lange erweitert, bis die terminalen erreicht sind. Die Prozedur Complete schließlich sorgt dafür, daß die erkannten Teilkonstituenten in den übergeordneten Strukturen Verwendung finden.

Die Prozedur Expand

Die Prozedur Expand generiert Kanten, die aufzeigen, wie eine begonnene Ableitung top-down fortgesetzt werden kann. Sie trägt zyklische Kanten in die Chart ein. Ausgehend von einer bereits in der Chart enthaltenen aktiven Kante wird für jede Regel der Syntax, deren linke Seite mit dem ersten Symbol des offenen Abschnitts der Kante identisch ist, eine neue Kante in die Chart eingetragen. Tilgungsregeln führen zum Eintrag von passiven Kanten, alle übrigen Regeln zu aktiven Kanten.

Beispiel: Gegeben sei die Syntax $\mathcal{G}_3$ mit den erweiterten Regeln (r5) und (r6) und eine aktive Kante [0, 0, SATZ, ϵ, NP VP]. Expand erzeugt dann die Kante [0, 0, NP, ϵ, N] aus Regel (r2) und die Kante [0, 0, NP, ϵ, DET N] aus Regel (r6).

Die Prozedur Scan

Ist das erste Symbol eines offenen Abschnitts einer aktiven Kante eine syntaktische Kategorie, der das nächste Wort des Satzes angehört, wird eine Kante in die Chart eingetragen, die sich von der ausgehenden Kante nur darin unterscheidet, daß die syntaktische Kategorie das letzte Symbol des geschlossenen Abschnitts der neuen Kante bildet.

Beispiel: Gegeben sei der Satz *Obelix sucht Troubadix*, der Lexikoneintrag *Obelix* [N] und die Chart aus dem Beispiel der Prozedur Expand mit der Kante [0, 0, NP, ϵ, DET N]. Die Prozedur Scan erzeugt eine neue Kante [0, 1, NP, N, ϵ].

Die Prozedur Complete

Durch Kombination einer aktiven Kante mit einer geeigneten passiven Kante erzeugt die Prozedur Complete eine neue aktive oder passive Kante, die dann in die Chart eingetragen wird. Die aktive Kante wird bottom-up durch eine passive ergänzt. Eine aktive Kante k_a kann mit einer passiven Kante k_p zusammengeführt werden, wenn das erste Symbol des offenen Abschnitts von k_a mit dem Kopf von k_p übereinstimmt und $end(k_a) = start(k_p)$.

Beispiel: Gegeben sei die Chart aus der Prozedur Scan. Die Prozedur Complete generiert aus [0, 0, NP, ϵ, N] und [0, 1, NP, N, ϵ] die neue Kante [0, 1, NP, N, ϵ] und in einem weiteren Schritt aus dieser Kante und der Kante [0, 0, SATZ, ϵ, NP VP] die Kante [0, 1, SATZ, NP, VP].

Die Berechnung der Chart

Mit diesen drei Prozeduren kann nun eine Chart berechnet werden, die Informationen über sämtliche syntaktischen Lesarten eines Satzes enthält. Mit ihr können dann alle möglichen Konstituentenstrukturen eines Satzes erzeugt werden, die mit der zugrundeliegenden Grammatik beschreibbar sind.

Eine Möglichkeit, die gewünschte Chart zu erzeugen, besteht darin, jede von Expand oder Complete erzeugte Kante direkt weiterzuverarbeiten. Ist diese Kante aktiv, wird Expand aufgerufen, ist sie passiv, ruft man Complete auf. Das Vorgehen kann durch eine Prozedur *Closure* gesteuert werden, die jede neue Kante in die Chart einträgt und diese neue Kante weiter untersucht und entsprechend Complete oder Expand aufruft. Die Prozeduren Expand, Complete und Scan müssen dann so geändert werden, daß sie mit jeder neu berechneten Kante Closure aufrufen. Der Algorithmus kann wie folgt beschrieben werden:

Vorraussetzung: Die mit den Regeln (r5) und (r6) erweiterte Grammatik $\mathcal{G}_3$ (ohne Tilgungs- und Kettenregeln) mit Lexikon $\mathcal{X}$ aus Kapitel 4.1.

Eingabe: Ein Satz $\omega = \omega_1 \dots \omega_n$ mit $n \geq 1$.

Ausgabe: *TRUE* oder *FALSE*[3].

Datenstruktur: Eine leere Chart $\mathcal{C}$.

Methode: *Initialisierung:* Für jede Regel SATZ $\rightarrow \alpha \in \mathcal{P}$ rufe die Prozedur Closure mit der Kante [0, 0, SATZ, ϵ, α] auf.

[3]Dieser Erkennungsalgorithmus prüft nur, ob der Satz ω der Grammatik entspricht oder nicht. Ein Parser nimmt ω als Eingabe und erzeugt, wenn die Eingabe der Grammatik entspricht, alle Konstituentenstrukturen, die laut Grammatik zulässig sind.

Nr.	neue Kante	Prozedur	Kanten/Regeln
k1	[0, 0, SATZ, ϵ, NP VP]	(Initialisierung)	r1
k2	[0, 0, NP, ϵ, N]	Expand	r2
k3	[0, 0, NP, ϵ, DET N]	Expand	r5
k4	[0, 1, NP, N, ϵ]	Scan	k2
k5	[0, 1, SATZ, NP, VP]	Complete	k1, k4
k6	[1, 1, VP, ϵ, V NP VP]	Expand	r3
k7	[1, 1, VP, ϵ, V NP]	Expand	r6
k8	[1, 2, VP, V, NP PP]	Scan	k6
k9	[2, 2, NP, ϵ, N]	Expand	r2
k10	[2, 2, NP, ϵ, DET N]	Expand	r5
k11	[1, 2, VP, V, NP]	Scan	k7
k12	[2, 3, NP, N, ϵ]	Scan	k9
k13	[1, 3, VP, V NP, ϵ]	Complete	k11, k12
k14	[0, 3, SATZ, NP VP, ϵ]	Complete	k13, k5
k15	[1, 3, VP, V NP, PP]	Complete	k8, k12
k16	[3, 3, PP, ϵ, P NP]	Expand	r4

Tabelle 4.2: Die mit einem Earley-Parser erstellte Chart für den Satz Obelix sucht Troubadix. *[10]*

Berechnung der übrigen Kanten: Für alle $i \in \{1, \ldots, n\}$ rufe die Prozedur Scan mit ω_i auf. Wenn die Kante [0, N, SATZ, α, ϵ] Element der Chart $\mathcal{C}$ ist, gib *TRUE* zurück, ansonsten *FALSE*.

Tabelle 4.2 zeigt die Ableitung für den Satz *Obelix sucht Troubadix* mit der Grammatik $\mathcal{G}_3$ und den erweiterten Regeln (r5) und (r6). [10] [21]

4.3 Deterministische Parser

Deterministische Parser sind ursprünglich für den Compilerbau entwickelt worden, da sie zu jedem Zeitpunkt der Analyse eindeutig feststellen können, welche Aktion im nächsten Schritt durchzuführen ist. Somit entfällt die Speicherung von Teilergebnissen und auch die Untersuchung von alternativen Analysemöglichkeiten. Diese Parser arbeiten aus diesem Grund wesentlich effizienter als die Parser der vorangegangenen Kapitel.

In der Computerlinguistik haben die deterministischen Parser aufgrund ihrer Beschränkung, nur *deterministische kontextfreie Sprachen* beschreiben zu können, lange Zeit nur eine geringe Rolle gespielt. Die Mehrdeutigkeiten natürlicher Sprachen lassen sich mit diesen Syntaxen nicht beschreiben. Es gibt jedoch einen von M. Tomita entwickelten *generalisier-*

ten LR-Parsingalgorithmus, der mittlerweile sehr häufig in natürlichsprachlichen Systemen verwendet wird. [31] [32]

4.3.1 LR-Parser

Ein LR-Parser ist ein Bottom-up-Parser, der die Eingabe von links nach rechts abarbeitet, erkannte Strukturen jedoch von rechts nach links auflöst. Er wird daher als Left-to-right Rightmost Parser bezeichnet. Der LR-Parser basiert auf dem Kellerautomaten, einer Erweiterung eines *deterministischen endlichen Automaten* (engl. deterministic finite automaton). Wesentliche Bestandteile eines Kellerautomaten sind die Operationen *Shift* und *Reduce*, weshalb ein LR-Parser auch als *Shift-Reduce-Parser* bezeichnet wird.

Die Analyse eines Satzes wird also entweder durch eine Reduktion *(Reduce)* fortgesetzt oder es wird per *Shift* das nächste Wort gelesen. Ein LR-Parser besteht aus zwei Teilen. Zum einen aus einer Parsingtabelle, die auf Grundlage der Grammatik berechnet wurde und zum Zweiten aus einer einfachen Steuerprozedur. Die Parsingtabelle besteht aus den Funktionen *Action* und *Goto*.

Die Funktion Action nimmt einen Zustand und ein Terminal als Argumente und liefert einen der Werte *Shift* <*neuer Zustand*>, *Reduce* <*Regel*>, *Accept* (Satz $\in \mathcal{L}(\mathcal{G})$) oder *Fail* (Satz $\notin \mathcal{L}(\mathcal{G})$). Die Funktion Goto nimmt einem Zustand z und ein Nicht-Terminal $\mathcal{N}$ als Argumente und liefert einen Zustand z' als Wert. Sie wird benötigt, um nach einer Reduce-Operation den neuen Zustand des Parsers zu bestimmen.

Die Steuerprozedur eines LR-Parsers geht von einem Startzustand aus und führt dann eine Folge von Operationen aus, die ihn letztendlich in einen Zustand führt, in dem die Funktion Action den Wert *Accept* oder *Fail* liefert. In den Tabellen 4.3 und 4.4 sind die Parsingtabelle und die Ableitung der Syntax $\mathcal{G}_3$ des Satzes *Obelix sucht Troubadix im Wald* gezeigt. [10] [21] [30]

4.3.2 Generalisiertes LR-Parsing

Wird ein LR-Parser mit einer Grammatik angewendet, die nicht zur Klasse der LR-Syntaxen gehört, enthält die Parsingtabelle mindestens ein Feld mit mehreren Einträgen. Die Eindeutigkeit des folgenden Arbeitsschrittes ist somit nicht mehr gegeben. Der *generalisierte LR-Parsingalgorithmus* von M. Tomita arbeitet mit einer besonderen Stackstruktur, dem *graph-strukturierten Stack*, mit der es möglich ist, die Mehrfacheinträge in der Tabelle quasi gleichzeitig, also *breadth-first* zu bearbeiten.

Man könnte jetzt für jeden Eintrag einen eigenen Prozeß starten. Jede alternative Analysemöglichkeit wird als eigener LR-Parsing Prozeß gestartet und solange verfolgt, bis er im

	Action				Goto			
Zustand	N	V	P	End	NP	VP	PP	SATZ
0	s3				1			2
1		s5				4		
2				acc				
3		r2						
4				r1				
5	s7				6			
6			s9				8	
7			r2					
8				r3				
9	s11				10			
10				r4				
11				r2				

Tabelle 4.3: Die Parsingtabelle der Grammatik $\mathcal{G}_3$ für den LR-Parser. Ein leeres Feld symbolosiert den Zustand Fail. *[10]*

Schritt	Stack	Eingabe	
0	0	N V N P N End	s3
1	0 N 3	V N P N End	r2
2	0 NP 1	V N P N End	s5
3	0 NP 1 V 5	N P N End	s7
4	0 NP 1 V 5 N 7	P N End	r2
5	0 NP 1 V 5 NP 6	P N End	s9
6	0 NP 1 V 5 NP 6 P 9	N End	s11
7	0 NP 1 V 5 NP 6 P 9 N 11	End	r2
8	0 NP 1 V 5 NP 6 P 9 NP 10	End	r4
9	0 NP 1 V 5 NP 6 PP 8	End	r3
10	0 NP 1 VP 4	End	r1
11	0 SATZ 2	End	acc

Tabelle 4.4: Die Ableitung des Satzes Obelix sucht Troubadix im Wald *mit einem LR-Parser. [10]*

Zustand Fail endet. Er wird dann aus der Stack-Liste entfernt. Da ein Prozeß jedoch nicht ohne weiteres mit anderen kommunizieren kann und viele Informationen mehrfach vorliegen, steigt die Anzahl der Stacks mit dem Auftreten von Ambiguitäten exponentiell.

Baum-strukturierter Stack

Mit Hilfe eines *baum-strukturierten Stacks* wird das Verfahren effizienter. Solange zwei synchronisierte Prozesse sich im selben Zustand befinden, führen sie solange die gleichen Operationen aus und werden zu einem Prozeß zusammengefasst, bis eine Reduce-Aktion dieses Zustandssymbol vom Stack entfernt und die Prozesse wieder voneinander getrennt werden. Der Stack des neuen Prozesses kann als Baum repräsentiert werden, dessen Wurzel den Zustand der beiden zusammengefassten Prozesse darstellt. Dieses Verfahren ist zwar wesentlich effizienter als die vorher besprochene Lösung, der Stack wird beim Aufspalten eines Prozesses jedoch weiterhin komplett kopiert. Somit steigt die Anzahl der Prozesse immer noch exponentiell.

Graph-strukturierter Stack

Mit dem *graph-strukturierten Stack* wird beim Starten eines neuen Prozesses nicht einfach der gesamte Stack kopiert, der Stack wird vielmehr als ein Baum repräsentiert, dessen Wurzel den Anfangszustand des ursprünglichen Prozesses darstellt. Die Äste bilden den unterschiedlichen Verlauf der neuen Prozesse ab. Der Stack kann nun als gerichteter azyklischer Graph so aufgebaut werden, daß die Stackgruppe nicht mehr eine Liste von Stacks, sondern als ein einziger, unter Umständen sehr komplexer, graph-strukturierter Stack abgebildet wird.

Subtree-Sharing und Local Ambiguity Packing

Da der Ambiguitätsgrad von Sätzen, also die Anzahl der Strukturen, die diesem Satz zugeordnet werden können, exponentiell mit deren Länge steigt, würde auch die vom Parser benötigte Zeit, diese Strukturen zu berechnen und auszugeben, exponentiell wachsen. Neben einem effizienten Verfahren der Analyse ist also ebenfalls eine effiziente Verwaltung der Strukturen nötig. Der Tomita-Algorithmus bedient sich hier zweier Techniken, die er zu diesem Zweck kombiniert.

Das *Subtree-Sharing* nutzt gemeinsame Teilbäume der zu generierenden Strukturen, indem diese übereinstimmenden Teilbäume „aufeinander“ gelegt werden. So entsteht ein Graph, der gemeinsame Teilstrukturen nur einmal repräsentiert. Das Ergebnis wird als *Shared-*

Forest-Repräsentation bezeichnet.

Mehrere Konstituentenstrukturen eines Satzes werden als *lokale Ambiguität* eines Satzes bezeichnet, wenn sie gemeinsame Terminalknoten besitzen und die Etiketten ihrer Wurzelknoten gleich sind. Um zu viele solcher lokaler Ambiguitäten zu vermeiden, können zu ihrer Repräsentation *gepackte Knoten* verwendet werden. Die Wurzelknoten der Teilbäume, die eine lokale Ambiguität aufweisen, werden zu einem solchen gepackten Knoten zusammengefasst, der von übergeordneten Strukturen als ein Knoten behaldelt und als *Subknoten* des gepackten Knotens bezeichnet wird. Dieses Verfahren heißt *Local Ambiguity Packing*. Wenn diese zwei Verfahren kombiniert werden, dann werden verschiedene Strukturbeschreibungen eines Satzes miteinander verknüpft und als *Packed Shared Forests* repräsentiert.

An den beiden Konstituentenstrukturen in Abbildung 3.9 und 3.10 kann man sehen, daß diese zwei Strukturen gemeinsame Teilbäume besitzen, die sich mit Hilfe der hier behandelten Techniken miteinander verknüpfen lassen. [21] [10]

4.4 Parser für Unifikationsgrammatiken

Die unifikationsbasierten Grammatikformalismen wie die bereits besprochene Lexikalisch-funktionale Grammatik haben eine weitaus größere generative Kapazität als die kontextfreien Syntaxen. Die LFG wie auch die HPSG beschreiben auch die *mild kontextsensitiven* Grammatiken[4], somit ist ein Übernehmen der Analysetechniken der kontextfreien Grammatiken nicht ohne weiteres möglich.

Ein Earley-Parser kann durchaus zur Analyse einer Unifikationsgrammatik – genauer einer kontextfreien Grammatik mit komplexen Symbolen – genutzt werden. Zunächst ist die Operation der *Unifikaton* zu implementieren, aus zwei Merkmalsstrukturen M_1 und M_2 soll eine neue Merkmalsstruktur M_3 generiert werden. Hierzu gibt es unterschiedliche Methoden. Eine Lösung ist, eine Kopie von M_1 und M_2 derart zu erweitern, daß sie implizit das Ergebnis der Unifikation repräsentiert.

Die vorhandenen Merkmalsstrukturen müssen nun mit ihren Attributen und Werten ebenfalls in die Chart eingetragen und bei der weiteren Verarbeitung beachtet werden. Die Merkmale und Werte aller Kategorien einer Kante werden in der Menge *varPart* zusammengafasst. Eine Kante k der Chart hat jetzt das Format [*start(k). end(k)*, *head(k)*, *passivePart(k)*, *activePart(k)*, *varPart(k)*]. Die Operationen *Expand*, *Complete*, *Scan* und vor allem *Closure* müssen nun an die neue Kantenstruktur angepasst werden, z.B. müssen

[4] Die *Tree Adjoining Grammars* oder TAGs sind ebenfalls in der Lage, eine bestimmte Klasse mild kontextsensitiver Sprachen zu akzeptieren. Für weitere Informationen siehe [10] und [15].

anstatt der Forderung von Gleichheit zweier Kanten diese nun unifiziert werden.

Durch die nun vorhandene Komplexität der Kanten und Regeln entstehen noch einige weitere Probleme, deren Lösung an dieser Stelle zu weit führen würde. Den interessierten Leser verweise ich auf die Werke [10] und [21].

4.5 Statistisches Parsen

Statistische Methoden spielen traditionell eine wesentliche Rolle, wenn es um die Verarbeitung von großen Datenmengen geht. Aus *Korpora*[5] können so linguistische Informationen z.B. für den Lexikonaufbau, für Grammatikdefinitionen oder auch für den Aufbau von Weltwissen aus diesen extrahiert werden. Viele statistische Formalismen zur Analyse von natürlicher Sprache basieren auf bekannten Techniken wie endlichen Automaten oder kontextfreien Syntaxen. Sie nutzen zusätzlich statistische oder probabilistische Informationen, um so über die Wahrscheinlichkeit eines Zustandsübergangs oder die Wahrscheinlichkeit einer Regel weitere Entscheidungen zu treffen. Die zusätzlichen statistischen Informationen haben vor allem folgende Vorteile:

- Bei mehrdeutigen Eingaben können statistische Daten eine weitere Informationsquelle zur Entscheidungsfindung sein.
- Im laufenden Parsingprozeß können weniger gut bewertete Teilergebnisse verworfen werden. Kriterien hierfür können Schwellwerte sein, die eine Mindestgüte eines Ergebnisses fordern.
- Mit statistischen Systemen sind sehr robuste Analysen möglich. Es ist in vielen Systemen möglich, auch nicht-wohlgeformte Eingaben grundsätzlich zu verarbeiten und sogar ein Maß für den Grad der Abweichung von der verwendeten Grammatik anzugeben.
- Stochastische Grammatiken und Automaten können mit entsprechenden Algorithmen automatisch erzeugt werden. Eine manuelle Grammatikentwicklung, die aufwändig und fehlerträchtig ist, kann so vermieden werden.

Im wesentlichen können drei Gruppen von Ansätzen im Bereich des statistischen Parsens unterschieden werden:

[5] Der Begriff *Korpus* bezeichnet im allgemeinen Textsammlungen, die aus bestimmten Gründen zusammengehören, wie z.B. das Gesamtwerk eines Schriftstellers. Korpora können jedoch auch gezielt zusammengestellte Texte sein wie Zeitungsjahrgänge eines Jahres oder Kriminalromane. Als Beispiel läßt sich hier das *Brown Corpus* nennen, das von W. Nelson Francis und Henry Kucera an der Brown University zusammengestellt und 1964 erstmals zugänglich gemacht wurde. Das Korpus enthält englischsprachige Prosa, die in den Vereinigten Staaten während des Kalenderjahres 1961 gedruckt wurde. [17]

Hidden-Markov-Modelle: Ein *Hidden-Markov-Modell (HMM)* ist ein endlicher Automat, dessen Zustände mit *Emissionswahrscheinlichkeiten* für die Ausgabesymbole bezeichnet sind. Seine Zustandsübergänge werden mittels *Übergangswahrscheinlichkeiten* bewertet und er besitzt einen Vektor für die *Anfangswahrscheinlichkeiten*. Für Hidden-Markov-Modelle gibt es effiziente Trainingsverfahren und Algorithmen zur Bestimmung der wahrscheinlichsten Analyse für ein gegebenes Modell und eine gegebene Eingabe. Hidden-Markov-Modelle werden besonders im Bereich der Spracherkennung eingesetzt.

Stochastische kontextfreie Grammatiken: Bei *stochastischen kontextfreien Grammatiken (SCFG)* wird jeder Produktionsregel ein Wahrscheinlichkeitswert zugeordnet. Mit diesen Grammatiken sowie mit stochastischen Grammatiken mit kontextfreiem Skelett wurden unter anderem der Tomita-Algorithmus [4] und der Earley-Algorithmus [29] beschrieben.

Stochastische Baum-Grammatiken: Einer stochastischen kontextfreien Grammatik liegt die Annahme zugrunde, daß jeder Regel unabhängig von ihrem Kontext genau eine Wahrscheinlichkeit zugeordnet ist. Dies kann unter Umständen zu Problemen bei der Bewertung von vollständigen Strukturen führen. Anstatt kontextfreier Regeln werden oft auch Teilbäume eingesetzt. [10] [21]

5 Expertensysteme

Expertensysteme sollen die Fähigkeiten eines menschlichen Experten auf einem speziellen Gebiet simulieren. Hierfür nutzen sie Techniken der künstlichen Intelligenz wie z.B. *die Prädikatenlogik*, verschiedene *Schlußmethoden* und *semantische Netze*. Ein Expertensystem soll auf Nachfrage des Anwenders seine Schlüsse erklären und auch begründen können. Menschliche Experten sollen so bei Routieneaufgaben unterstützt und entlastet werden. Weiterhin können auch Menschen mit weniger Expertise von dem Wissen eines solchen Expertensystems profitieren, indem sie dieses als Werkzeug benutzen. Ein weiteres Ziel ist es, das Wissen, das menschliche Experten durch Lernen und jahrelange Erfahrung gesammelt haben, automatisiert verarbeitbar gespeichert wird und somit jederzeit reproduzierbar ist.

Da die Hauptaufgabe eines Expertensystems somit die Verarbeitung von Wissen ist, spielen die Wissensbasis und der Inferenzmotor (der Schlußfolgerungsmechanismus) im Aufbau eines Expertensystems eine wesentliche Rolle. Das System muß in der Lage sein, über eine Benutzerschnittstelle mit dem Anwender zu kommunizieren und die Ergebnisse zu präsentieren. Eine weitere wichtige Option ist die Erweiterung der Wissensbasis, sei es über eine einfache Eingabe in eine Datenbank oder automatisiert mit diversen Algorithmen[1]. Im Folgenden werden die fünf Komponenten von Abbildung 5.1 erläutert.

In der Wissensbasis ist das problembezogene Wissen eines Expertensystems gespeichert. Um die Vorteile der expliziten Wissenspeicherung auszunutzen, sollte in keiner anderen Komponente problembezogenes Wissen gespeichert sein. Der Inhalt der Wissensbasis kann grob unterschieden werden in generisches Wissen, das unabhängig vom aktuellen Anwendungsfall gespeichert ist, und dem fallspezifischen Wissen, das zur Lösung des aktuellen Anwendungsfalls notwendig ist. Lernfähige Systeme können fallspezifisches Wissen, nach der Lösung eines Problems, in den generischen Teil der Wissensbasis aufnehmen. Das generische Wissen umfaßt Fakten zum Problembereich, Wissen über deren Zusammenhänge und Wissen über Strategien, wie das vorhandene Wissen eingesetzt werden kann.

Der Inferenzmotor stellt die zentrale Problemlösungskomponente dar. Hier werden die aus der Problemstellung extrahierten Fakten und die Fakten aus der Wissensbasis verknüpft und auf neue Fakten geschlossen. Die Reihenfolge, in der dies geschieht, wird

[1] Einige dieser Lernmethoden werden näher erläutert in [23]

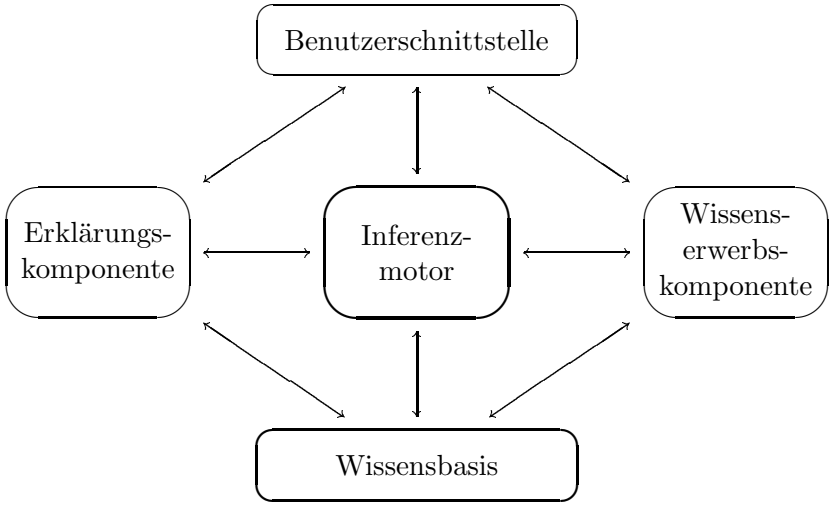

Abbildung 5.1: Aufbau eines Expertensystems. Die Wissensbasis und der Inferenzmotor nehmen eine zentrale Rolle ein.

von der Inferenzmaschine selbst festgelegt. Nach Beendigung des Schlußflogerungsprozesses werden die neugewonnenen Fakten von der Benutzerschnittstelle aufbereitet und als Lösung des Problems dem Anwender präsentiert.

Die Erklärungskomponente eines Expertensystems liefert Informationen über das Zustandekommen der Lösung. Typische Fragestellungen dabei sind *Wie wurde die Lösung eines Problems gefunden?*, *Warum wird eine bestimmte Information vom System nachgefragt?* oder *Warum wurde eine bestimmte Lösung nicht gefunden?*.

Die Wissenserwerbskomponente hat die Aufgabe, den Aufbau und die Erweiterung der Wissensbasis zu unterstützen. Sie stellt Funktionen zur Verfügung, die die Konsistenz und Vollständigkeit des gespeicherten Wissens überprüfen.

Die Benutzerschnittstelle unterscheidet zwei Arten der Interaktionen mit dem Expertensystem: Die Kommunikation mit dem Anwender, der nach der Lösung eines bestimmen Anwendungsproblems sucht, und der Kommunikation mit dem Knowledge Engineers, das ist die Person, die die Wissensbasis erstellt und wartet. Zu diesem Zweck werden in beiden Fällen die Fakten über die Problemlösung annähernd natürlichsprachlich aufbereitet oder Zusammenhänge graphisch dargestellt.

Expertensysteme können auf dem *wissens- und regelbasierten Ansatz* beruhen. Sie setzen in diesem Fall die klassische symbolische Logik einschließlich einer Wissensbasis und einer

Menge von Regeln für ihre Schlußfolgerungen ein. Andere Expertensysteme beruhen auf *neuronalen Netzen* oder *statistischen Elementen* und *Fuzzy Logic*. Moderne Expertensysteme bedienen sich einer Vielzahl von Techniken aus unterschiedlichen Bereichen der Mathematik und der Informatik und koppeln dieses Wissen und diese Techniken mit dem Spezialwissen ihres jeweiligen Anwendungsbereiches.

Einige Beispiele von Expertensystemen wie oben beschrieben sind in [23] näher erläutert. Der Begriff *Expertensystem* kann jedoch auch ein wenig weiter gefaßt werden. So können auch Systeme, die sich nicht direkt eines Inferenzmotors bedienen, als Expertensysteme bezeichnet werden. Das Expertensystem *ScienceAtlas* zum Beispiel ist „Experte“ in mathematischen Berechnungen und im Erklären von mathematischen Begriffen. Ein Schlußfolgerungsalgorithmus im klassischen Sinne ist (noch) nicht implementiert, jedoch können unter anderem Berechnungen mit Funktionen, Matritzen und komplexen Zahlen durchgeführt sowie Graphen dargestellt werden – also: ein Experte der Mathematik. [23] [9] [26]

6 Das Expertensystem ScienceAtlas

„Das Expertensystem *ScienceAtlas*[1] wurde in den Jahren 1998-2003 in Göttingen, Köln und London als Prototyp erdacht. Seitdem findet eine kontinuierliche Weiterentwicklung und Professionalisierung des umfangreichen Systems statt.“ [24]

ScienceAtlas ist ein mathematisches Expertensystem, das es dem Nutzer über einen Internet-Browser ermöglicht, in der Wissensbasis Begriffe nachzuschlagen oder Vorlesungsscripte ganz oder teilweise einzusehen. Weiterhin bietet es die Möglichkeit, mit Variablen, Zahlen, Brüchen, Matritzen und auch Funktionen zu arbeiten, die zwei- oder auch dreidimensional visualisiert werden können. Diese Variablen, Berechnungen und Funktionen können dann in einer systemeigenen Programmierumgebung verwendet werden. Der Kern von ScienceAtlas ist in C++ programmiert. Um über das Medium Internet mit den Nutzern kommunizieren zu können, nutzt das System PHP zur Generierung von HTML sowie Erweiterungen in JavaScript sowie ein relationales Datenbankmodell. Im Einzelnen besteht ScienceAtlas aus

- der **Wissensbasis**, in der unterschiedlichste Themen erläutert werden und gespeichert sind,
- einem **Sprachparser**, der die natürlichen Fragen und Anweisungen der Teilnehmer verarbeitet und zu beantworten versucht,
- einer **Programmierumgebung**, in der die Nutzer Programme erstellen können,
- einer **Grafik-Komponente**, mit der zwei- und dreidimensionale Grafiken ausgegeben werden können
- und einer umfangreichen **Funktionsbibliothek** zu mathematischen und naturwissenschaftlichen Themen.

ScienceAtlas wurde entwickelt, um Grundlagenwissen in einfacher und effektiver Weise verfügbar zu machen. Das Ziel, die Beschränkungen einer Programmiersprache mit ihren oft starren Syntaxregeln zu umgehen, soll mit einem natürlichsprachlichen Zugang erreicht werden, der in vielen Fällen durch Mausklick optimiert werden kann.

[1] http://www.ScienceAtlas.de

Das System wird in Schulen eingesetzt, um Schülern das Erlernen von Mathematik, Physik, Informatik und weiteren Naturwissenschaften zu erleichtern. Im Hochschulbereich begleitet ScienceAtlas mathematische Vorlesungen für Biologen, Wirtschaftswissenschaftler, Physiker, Mathematiker, Informatiker oder Ingenieure. In der Forschung wird es genutzt, um Forschungsergebnisse schnell verfügbar und für Fachkollegen leichter zugänglich zu machen. Zu guter Letzt wird ScienceAtlas auch in der Industrie von Unternehmen eingesetzt, um das Unternehmenswissen den Mitarbeitern effektiv verfügbar zu machen.

Die Wissensbasis bietet viele Möglichkeiten der Ergänzung, somit ist das System im Prinzip beliebig erweiterbar und auch in weiteren Gebieten nutzbar. [24]

6.1 Die Wissensbasis

Der Zugriff auf die Wissensbasis von ScienceAtlas erfolgt in sogenannten *Views*, die unterschiedliche Sichten auf das gespeicherte Wissen erlauben und verschiedene, an die Situation angepasste Hilfsmittel für die Arbeit mit dem System zur Verfügung stellen.

Mit Ausnahme der SeminarView verfügen alle Sichten über ein Eingabefeld als Kommunikations-Komponente, über das der Nutzer Anfragen und Anweisungen an das System senden kann. Diese werden in einer *Benutzer-History* gespeichert, so daß der Nutzer per Mausklick auf bereits eingebene Texte zurückgreifen kann. Ein Antwortfeld des Systems ist ebenfalls Bestandteil der Nutzer-Eingabe. Folgende Sichten sind vom System vorgesehen:

TreeView: Aufgabe dieser Ansicht des Systems ist es, den Baum hierarchisch zu erforschen und – einem Dateisystem gleich – die Daten an den Blättern abzurufen. Der Nutzer kann sehen, welche Blätter verfügbar sind und diese aufrufen, um sich Definitionen und Erklärungen – also das gespeicherte Wissen – anzeigen zu lassen. *Beispiel:* Mathematik $\gg$ Hochschul-Mathematik $\gg$ Funktionalanalysis $\gg$ Norm $\gg$ Matrixnorm.

ComunityView: Die CommunityView blendet zusätzlich noch eine Zeile mit Nutzerkommentaren und Bemerkungen ein.

ProgrammingView: In der ProgrammingView hat der Nutzer Zugriff auf die von ihm definierten Variablen, Funktionen und Programme, also auf seinen *Variablenraum*, der wie die History per Mausklick abgefragt werden kann. Außerdem wird eine Referenz mit einigen Hilfen zu Syntax und Funktion der ScienceAtlas Eingabe angeboten.

SeminarView: Ähnlich der TreeView ist es mit der SeminarView möglich, durch Vorlesungs- und Semiarscripte zu browsen. Diese können ausgehend von einer Seminarliste als *ganzes Seminar*, *Kapitel*, *Abschnitt*, *Teilabschnitt* oder *Element* dargestellt werden. Als einzige Sicht bietet die SeminarView kein Nutzereingabefeld – so kann sich der Nutzer ganz auf den Inhalt der Scripte konzentrieren.

PlainView: Die PlainView zeigt nur die Kommunikations-Komponenten von ScienceAtlas an, zu der abgesehen von Eingabe- und Antwortfeld nur noch die Benutzer-History gehört.

HelpView: Mit Hilfe dieser Sicht auf das System ist es dem Nutzer möglich, das Einführungs-Script, das ebenfalls in der SeminarView zu sehen ist, zu lesen und gleichzeitig das Gelesene an Ort und Stelle auszuprobieren.

6.1.1 Die allgemeine Wissensdatenbank

Die allgemeine Wissensdatenbank von ScienceAtlas ist als *Wissensbaum* organisiert. Dieser bietet eine einfache Möglichkeit, Wissen in systematischer Weise abzulegen und auf einfache und effektive Weise für die Nutzer zugänglich zu machen. An den Knoten des Baumes sind die Begriffe, Themen und Abschnitte notiert, an den Blättern die eigentliche Information. So wird eine hierarchische Struktur aufgebaut, mit der diese Begriffe und Abschnitte gut gegliedert und dem Nutzer zur Verfügung gestellt werden können. Die Blätter sind nach verschiedenen Standardfragen sortiert wie *was*, *wie*, *wo*, *woraus*, *wann*, *wer* und *wozu*.

6.1.2 Die Funktionsbibliothek

Weiterhin enthält ScienceAtlas eine umfassende *Funktionsbibliothek* mit Funktionen wie `sin()`, `cos()`, `tan()` und `mod()`, die fortlaufend erweitert wird.

Eingaben in die Wissensdatenbank und in die Funktionsbibliothek werden nur von autorisierten Personen vorgenommen, so daß nicht jeder Nutzer Wissen in das System einbringen kann. Autorisierte Personen sind zum Beispiel Administratoren und Seminarleiter, die mit ScienceAtlas arbeiten. So ist sichergestellt, daß keine abstrusen Falschaussagen im System vorhanden sind. Schließlich sollen die Antworten eines Expertensystems korrekt sein.

6.1.3 Der Variablenraum des Nutzers

Das Expertensystem legt alle vom Benutzer definierten Variablen, Funktionen, Programme, Matritzen und andere definierbare Dinge wie Scripte (Folgen von Anweisungen), dauerhaft im System ab, so daß diese bei jedem erneuten Anmelden des Nutzers sofort per Mausklick oder Anweisung verfügbar sind. Dies ist eine wichtige Voraussetzung dafür, daß ScienceAtlas auf Anweisungen wie *Zeichne die Funktion f* korrekt reagieren kann. Denn nur, wenn *f* sich zeichnen läßt – also zum Beispiel mit `f(x)=sin(x)` definiert wurde – kann die Anweisung `plot(f, -2*pi, 2*pi)` zufriedenstellend ausgeführt werden[2].

[2] $pi = 3.14159$ ist im System als geschützte Variable vorgesehen und nicht veränderbar!

6.1.4 Frühere Befehlseingaben des Nutzers

Die bereits zuvor erwähnte Benutzer-History speichert vorangegangene Eingaben des Nutzers, ebenfalls permanent und abrufbar. So können Eingaben verarbeitet werden, die sich auf vorhergehende Befehle beziehen und diesem klar zugeordnet werden können. Anweisungen wie *Verkleinere den Definitionsbereich um 40%*, die eine vorher dargestellte Funktion betreffen, können bearbeitet und ausgeführt werden. [24]

6.2 Kommunikation mit ScienceAtlas

Mit dem Expertensystem ScienceAtlas läßt sich in vielerlei Hinsicht kommunizieren. Die weniger komplizierten Möglichkeiten – per Mausklick die Sichten ändern, Variablen aus dem Variablenraum aufrufen oder Befehle aus der Benutzer-History wiederverwenden – habe ich im vorigen Kapitel bereits erwähnt. Im Folgenden wird der der Parsingmechanismus erläutert, mit dem ScienceAtlas momentan die Texteingaben der Benutzer verarbeitet.

6.2.1 Der Parser

Zunächst wird der Eingabestring auf *systemspezifische Ausdrücke* wie Schleifen, Funktionsdefinitionen, Scriptdefinitionen und Bedingungsanweisungen untersucht. Wird hier keine Übereinstimmuing gefunden, werden die *Variablenräume* des Systems sowie die des Benutzers durchlaufen. Wird der Parser fündig, wird die Anweisung direkt (wenn nötig auch rekursiv) bearbeitet und das Ergebnis, z.B. der Wert einer Variable oder das Ergebnis eines Programmaufrufs, sogleich ausgegeben.

Wird kein systemspezifischer Ausdruck gefunden, wird der *Natural Language Processor* aufgerufen, der den Eingabestring mit Hilfe von regulären Ausdrücken auf im System definierte *Schemematches* untersucht. Beispiele für solche Schemematches sind

```
{W,w}ozu [braucht ,dient ][man ]§term(x)[?] und
[Zeig ,Zeige ,zeig ,zeige ][mir ][den ]{P,p}arameter §term(x)[!,.].
```

Auf diese Weise wird zunächst der Eingabestring als Ganzes bearbeitet. Wird keine Übereinstimmung gefunden, werden die einzelnen Komponenten – die durch Leerzeichen getrennten Worte – untersucht. Ist einer dieser beiden Wege erfolgreich, wird die Eingabe entsprechend verarbeitet, die Methode für das passende Schema wird ausgeführt.

Der Parser untersucht unter anderem, ob ihm der Typ eines Eingabeobjektes bekannt ist, in diesem Fall kann er diesen entsprechend behandeln. Eine Matrix wird z.B. in eine mathematische Grafik umgesetzt, eine natürlichen Zahl wird ohne besondere Formatierung ausgegeben und ein Script wird wieder anders behandelt. Unter anderem sind ScienceAtlas

die Typen NAT (Natürliche Zahl), REAL (Reelle Zahl), COMPLEX (Komplexe Zahl), MATRIX (Matrix), VARIABLE (Variable), FUNCTION (Funktion), SCRIPT (Script), POLYNOM (Polynom), EQUATION (Gleichung) und CLASS (Klasse) bekannt.

Sind auch diese letzten Versuche nicht erfolgreich, kann die Eingabe nicht verarbeitet werden und der Eingabestring wird unbearbeitet zurückgegeben. Auf eine Fehlermeldung wird explizit verzichtet, da das System darauf ausgerichtet ist, viele Möglichkeiten der Bearbeitung zu berücksichtigen.

6.2.2 Reaktionen auf die Nutzereingabe

ScienceAtlas reagiert auf erkannte Muster, Anweisungen und Ausdrücke im Allgemeinen dort, wo der Parser die Eingabe erkannt hat. In den oben genannten Methoden der definierten Schemata werden die nötigen weiteren Methodenaufrufe direkt ausgeführt und auch die Antwort für den Benutzer generiert. So wird z.B. durch die Eingabe von *Berechne die Determinante von A* erreicht, daß der Parser das Schemematch von

```
[B,b]erechne [den ,die ,das ]§term(x) von §term(y)[!,.]
```

erkennt und die entsprechende Funktion aufruft. Diese holt sich über den String und den Variablenraum des Nutzers die Typen der beiden Objekte (`§term(x)` und `§term(y)`), um zu kontrollieren, um welche Typen es sich handelt und ob die Determinante (`§term(x)`) von A (`§term(y)`) überhaupt berechnet werden kann. Ist dies möglich, wird die Funktion `det()` mit dem Argument `§term(y)` ausgeführt, das Ergebnis mit der virtuellen Maschine berechnet und schließlich dem Nutzer angezeigt.

Solche Methodenaufrufe können auch recht einfacher Natur sein wie z.B. die Anweisungen `Tree` oder `Plain`, die von der aktuellen Sicht auf die *TreeView* bzw. die *PlainView* umschalten und dem Nutzer dies mit einem Antwortsatz bestätigen. Oder aber sie sind komplexer Natur und haben die Ausführung eines Scripts oder eines Programms zur Folge.

In allen Fällen werden die Befehle in eine Befehlsfolge für die *Virtual Machine* des Systems umgesetzt, die dann dafür sorgt, daß entweder die Systemvariable für die verschiedenen Sichten umdefiniert oder ein komplexes Programm – gegebenenfalls rekursiv – ausgeführt wird. [24]

7 Entwurf eines Systems zur Verarbeitung natürlicher Sprache

Die natürlichsprachliche Eingabe des Nutzers soll syntaktisch wie semantisch analysiert werden, dabei sollen alle Komponenten der Wissensbasis berücksichtigt werden. Das System soll in einer angemessenen Zeit auf die Eingabe reagieren können, entweder direkt mit der Ausführung der gewünschten Funktion des Expertensystems oder mit einer Nachfrage.

Ein Entwurf zur Implementation wird am Beispiel der Funktionen `plot(f, x1, x2)` und `plot3D(f, x1, x2, y1, y2)` erläutert, die das System beinhaltet. Ein Nutzer des Expertensystems *ScienceAtlas* soll die Möglichkeit erhalten, einen Plot einer Funktion ausgegeben zu bekommen, ohne sich mit der Syntax der zwei genannten Befehle auseinandersetzen zu müssen.

Die Funktionen `plot()` und `plot3D()` benötigen folgende Parameter:

- `f`: eine Funktion im Sinne des Expertensystems.[1]
- `x1` und `x2`: den Definitionsbereich $\{x_1 \dots x_2\}$ der Funktion, wenn es sich um eine Funktion mit zwei Dimensionen handelt und somit `plot()` genutzt wird.
- `y1` und `y2`: den Definitionsbereich $\{y_1 \dots y_2\}$ der Funktion, wenn `plot3D()` genutzt wird, um eine dreidimensionale Darstellung zu erhalten.

Ziel ist es, einen Satz in natürlicher Sprache, der als Inhalt eine Anweisung oder ganz allgemein Informationen enthält, die mit der Darstellung einer Funktion zu tun haben, richtig zu interpretieren und die Anweisung des Nutzers auszuführen. Hat der Nutzer genügend Informationen angegeben, so kann die Funktion direkt geplottet werden. Sind nicht genügend Informationen vorhanden, kann das System zunächst versuchen, diese mit Hilfe der Wissensbasis zu ergänzen. Andernfalls müssen diese fehlenden Werte nachgefordert werden. Das Expertensystem soll auf verschiedene Sätze reagieren können, die mit dem Plotten von Funktionen zu tun haben. Hier einige Beispiele:

[1] Funktionen einer oder mehrerer Veränderlichen können einfach definiert werden in der Form `f(x)=3*x^2` bzw. `g(x,y)=x*y` und sind dann in üblicher Form nutzbar, etwa durch den Aufruf `f(3)` oder `g(4,5)`

1. *Zeichne die Funktion f.*
   Die Funktion f muß definiert sein. Ein Definitionsbereich ist nicht angegeben. Um nicht sofort den Nutzer nach dem Definitionsbereich fragen zu müssen, kann zuerst in einer Liste nachgesehen werden, ob vielleicht für diese Funktion eine entsprechende Voreinstellung existiert. Ansonsten könnte eine spezielle Funktion einen Bereich vorschlagen, den der Nutzer nachher angleichen kann.

2. *Zeichne die Funktion f mit dem Definitionsbereich (-5, 5).*
   Die Funktion f muß definiert sein.

3. *Vergrößere den Definitionsbereich um 50%.*
   Die letzte erfolgreich gezeichnete Funktion wird vom System gespeichert und auf diese Anfrage erneut geplottet, allerdings mit einem veränderten Definitionsbereich.

4. *Zeichne eine Sinusfunktion.*
   Die Funktion und der Definitionsbereich der Sinusfunktion sind im System gespeichert, so daß dieses auf die Funktion `f(x)=sin(x)` sowie auf einen voreingestellten Definitionsbereich $(0, 2\pi)$ zurückgreifen kann.

7.1 Der Parser und die Grammatik

Der Parser sollte die Eigenheiten der deutschen Sprache mit einbeziehen, seine Grammatik muß erweiterbar sein und er darf nicht allzu lange für seine Analysen brauchen. Besonders geeignet ist ein für Unifikationsgrammatiken modifizierter Chart-Parser und die Lexikalisch-funktionale Grammatik (LFG).

Dieser Parser liefert die funktionale Struktur der Eingabe, die relevante syntaktische und semantische Informationen in Form einer Attribut-Wert-Matrix enthält. Da ein Chart-Parser nicht deterministisch arbeitet, versucht er, alle möglichen Konstituentenstrukturen zu ermitteln. Mit Hilfe der funktionalen Beschreibungen der LFG können Mehrdeutigkeiten jedoch auf ein Minimum reduziert werden.

Grammatikalisch nicht korrekte Sätze wie *Zeichnen das Funktionen z mit Bereichen 8 von 22* können als nicht richtig erkannt werden und dem Nutzer kann eine entsprechende Mitteilung angezeigt werden. Im besten Fall könnte das System eine Korrektur vorschlagen, allerdings kann ich mir vorstellen, daß ein Benutzer des Systems sich ungern von einer Maschine korrigieren läßt. So wäre es sicher besser, die relevanten Daten aus dem Satz zu extrahieren und die Funktion zu zeichnen.

Weiterhin ist zu beachten, daß das Zeitlimit, das die Nutzer von web-basierten Anwendungen wie *ScienceAtlas* fordern, vielleicht mit einer solchen Korrektur übeschritten wird, 10 Sekunden wären hier schon sehr lange.

Es ist also wahrscheinlich sinnvoll – zu Ungunsten der Korrektheit und zu Gunsten des Benutzers – während der Analyse die Genauigkeit ein wenig schleifen zu lassen. Inwieweit sich das positiv für den Nutzer bzw. negativ für das Verständnis des Systems auswirkt, wäre in der Praxis zu erproben.

Im Folgenden werde ich am Beispiel des Satzes *Zeichne die Funktion f* und der Grammatik

$$\mathcal{G}_4 = \begin{pmatrix} \mathcal{T} &= \{\text{V, DET, N, P, [TYP]}\}, \\ \mathcal{N} &= \{\text{SATZ, NP, VP, NOM, PP}\}, \\ \mathcal{S} &= \text{„}\textit{Zeichne die Funktion f}\text{“}, \\ \mathcal{P} & \end{pmatrix}$$

mit den Produktionsregeln $\mathcal{P}$ in Abbildung 9.1 und dem Lexikon $\mathcal{X}$ in Abbildung 9.2 den Zusammenhang zwischen Parser und systemeigenen Funktionen und Objekten erläutern.

Die Grammatik basiert im Wesentlichen auf der LFG, einige systemspezifische Erweiterungen sind hinzugekommen. Das Terminal [TYP] bezeichnet systemeigene Objekte z.B. vom Typ [FUNCTION][2]. Ein Typ [RANGE] ist im System noch nicht vorhanden und muß neu eingeführt werden. Er bezeichnet den Definitionsbereich mit den zwei Werten x_1 und x_2.[3] Systemeigene Typen werden im Folgenden mit eckigen Klammern [ ] gekennzeichnet.

Das Verb „zeichnen“ fordert ein direktes Objekt vom Typ [FUNCTION] wie „f“ oder eines vom Typ FUNKTION wie „Funktion“, das wiederum einen Bezeichner vom TYP [FUNCTION] fordert. Ein indirektes Objekt ist optional und wird in den Regeln mit runden Klammern gekennzeichnet[4]. Der Nominalkomplex vor einem [TYP] ist ebenfalls nicht unbedingt erforderlich, denn für die Darstellung wird nur der Typ selbst benötigt, nicht unbedingt die Wörter „Funktion“ oder „Definitionsbereich“. Beispiele für diese Fälle sind *Zeichne meiner Mutter die Funktion f*, *Zeichne f* sowie *Zeichne f mit (-10, 10)*. Wenn allerdings „Funktion“ vorkommt, benötigt es einen Bezeichner vom Typ [FUNCTION] und „Definitionsbereich“ fordert einen Bezeichner vom Typ [RANGE], denn *Zeichne die Funktion* macht keinen Sinn, wenn nicht spezifiziert ist, welche Funktion gemeint ist.

Die Produktionsregeln unterscheiden sich von denen in Kapitel 3.3 – dort wurde die lexikalisch funktionale Grammatik vorgestellt – in der Form, daß sie Sätze im Imperativ erkennen,

[2] Der Typ FUNKTION bei „Funktion“ sowie der Typ BEREICH bei „Definitionsbereich“ sind keine Systemtypen. Diese Angabe dient nur der Kontrolle, ob „Funktion“ auch wirklich ein Objekt vom Typ [FUNCTION] bezeichnet. Sonst könnte man auch schreiben *Zeichne mir die Matrix f*, auch wenn *f* ein Objekt vom Typ [FUNCTION] ist. Der Parser könnte darauf nicht reagieren.

[3] Gemeint ist hier der Darstellungsbereich der Funktion in der Graphik und nicht den Definitionsbereich der Funktion im mathematischen Sinn!

[4] *Wem* die Funktion gezeichnet wird, ist an dieser Stelle nicht relevant. So kann die Funktion auch „meiner Mutter“ oder „mir“ gezeigt werden.

was für die Anweisungen an ScienceAtlas nötig ist.

Die Lexikoneinträge „f“ und „(-5, 5)“ sowie andere Namen und Inhalte von bereits dem System bekannten Objekten werden aus dem System gewonnen und vor der eigentlichen natürlichsprachlichen Analyse in das Lexikon eingetragen, so daß deren Werte dem Parser zur Verfügung stehen.

7.2 Das Parsing

Wie in Kapitel 3.3 vorgestellt, erstellt der Parser aus den Produktionsregeln und dem Lexikon eine Konstituentenstruktur, an deren Knoten die funktionalen Schemata notiert sind. Die Konstituentenstruktur des Satzes *Zeichne die Funktion f* sieht aus wie in Abbildung 9.3, mit eingetragenen funktionalen Schemata ergibt sich die K-Struktur wie in Abbildung 9.4.

Aus dieser Konstituentenstruktur wird mittels der *funktionalen Beschreibungen* die funktionale Struktur oder F-Struktur generiert, wie in Abbildung 9.5 zu sehen ist. Diese Attribut-Wert-Matrix muß folgende drei Wohlgeformtheitsbedingungen erfüllen: sie muß *vollständig*, *kohärent* und *konsistent* sein. Das heißt, daß in der funktionalen Struktur weder von den lexikalischen Formen geforderte Werte fehlen, noch daß zuviele davon existieren dürfen. Außerdem muß jedes Attribut der Matrix genau einen Wert haben.

Die F-Struktur wird vom Parser entwickelt und der semantischen Analyse zur Weiterverarbeitung zur Verfügung gestellt. Der Parser sollte die F-Struktur im Normalfall nur weitergeben, wenn diese wohlgeformt ist. Da die semantische Analyse jedoch unvollständige F-Strukturen unter Umständen ergänzen kann, werden diese nicht vom Parser, sondern von der semantischen Analyse bearbeitet.

7.3 Die semantische Analyse

Die semantische Analyse untersucht die F-Struktur auf Wohlgeformtheit, wobei die Analyse bestimmte Fehler tolerieren sollte, wie schon oben besprochen. Wenn also *Zeichne der Funktion f mit (-3, 3)* eingegeben wurde, sollte sie sich nicht damit aufhalten, den Nutzer zu korrigieren.[5] Drei Ergebnisse können grob unterschieden werden, weitere ähnliche Fälle sind möglich und entsprechend zu behandeln:

[5] Es sei denn, das Expertensystem soll eine Art „Oberlehrer-Persönlichkeit“ bekommen ☺

1. **Die F-Struktur ist wohlgeformt**

   Ist die funktionale Struktur wohlgeformt, sind im Fall der Beispieleingabe *Zeichne mir die Funktion f mit dem Definitionsbereich (-5, 5)* alle geforderten Argumente vorhanden, die alle direkt aus der F-Struktur entnommen werden können, siehe Abbildungen 9.6, 9.7 und 9.8.

   LINK verweist auf die zu benutzende Funktion des Systems, könnte jedoch auch auf eine Routine verweisen, die die Daten der F-Struktur weiterverarbeitet und erst dann die Systemfunktionen aufruft. Wahrscheinlich ist das im Fall von ScienceAtlas die bessere Methode, denn hier kann der Code für die virtuelle Maschine generiert werden und zusätzlich können etwaige zusätzliche Routinen aufgerufen werden, um dem unterschiedlichen Verhalten verschiedener Verben gerecht zu werden.

   Als Argumente sind die Objekte „f“ vom Typ [FUNCTION] und „(-5, 5)“ vom Typ [RANGE] mit deren Attributen NAME, X_1RANGE und X_2RANGE vorhanden. Im Prinzip kann die semantische Analyse jetzt direkt `plot(f, x1range, x2range)` aufrufen. Für eine dreidimensionale Funktion wird allerdings ein weiterer Definitionsbereich benötigt. Die semantische Analyse würde dieses erkennen und gegebenenfalls auf einen Standardwert zurückgreifen. Im Idealfall ist der Typ [FUNCTION] polymorph. Jeder Subtyp von [FUNCTION], zum Beispiel [2D_FUNCTION] und [3D_FUNCTION], kann so die Methode `plot()` passend implementieren. Dann würde die Routine `plot` des jeweiligen Objektes aufgerufen und die Grammatik müßte diesbezüglich nicht erweitert werden.

2. **Die F-Struktur ist nicht vollständig und kann vom System sinnvoll ergänzt werden**

   Sollte in der Struktur der Definitionsbereich fehlen, kann versucht werden, diese Daten aus der Wissensbasis zu ergänzen. Existiert z.B. eine Liste mit Funktionen, Funktionsnamen und Vorschlägen für Definitionsbereiche, wie in Tabelle 7.1 gezeigt, kann der Definitionsbereich ergänzt werden, wenn eine entsprechende Funktion in der Liste existiert. Die Funktionsnamen in dieser Liste können genutzt werden, um vom Lexikon – zum Beispiel von dem Wort *Sinusfunktion* – einen Verweis auf diese Liste zu setzen, um gegebenenfalls auf die Anweisung *Zeichne eine Sinusfunktion* reagieren zu können und so die richtige Funktion definieren und mit dem Definitionsbereich aus der Tabelle den Graph zeichnen zu können. Falls keine Funktion in dieser Liste mit der Eingabefunktion übereinstimmt, kann immer noch ein allgemeiner Defaultwert benutzt werden – oder eine Routine des Systems errechnet einen sinnvollen Bereich – bevor der Nutzer aufgefordert wird, doch bitte einen Definitionsbereich anzugeben.

   Fehlt der Bezeichner für die Funktion wie in *Vergrößere den Definitionsbereich um 50%*, kann versucht werden, diesen aus der Benutzer-History zu ergänzen. Das Analysemodul könnte die letzte erfolgreich gezeichnete Funktion annehmen.

Funktion	Name	Def. Ber.
$f(x) = sin(x)$	SIN	$(0, 2\pi)$
$f(x) = cos(x)$	COS	$(0, 2\pi)$
$f(x) = x^2$	QUAD	$(-8, 8)$
$\vdots$	$\vdots$	$\vdots$

Tabelle 7.1: Eine Liste, mit deren Hilfe der Definitionsbereich ergänzt werden kann.

3. **Die F-Struktur ist nicht wohlgeformt und kann nicht ergänzt werden**

   Wenn das Verb fehlt oder andere notwendige Strukturen nicht vorhanden sind, wird dem Nutzer eine entsprechende Meldung ausgegeben. Das System kann anhand der F-Struktur die fehlenden bzw. übeflüssigen Komponenten sehr genau lokalisieren. Der Nutzer bekommt eine sehr spezifische Fehlermeldung.

   Bei der Erweiterung des Entwurfs ist zu beachten, daß beispielsweise ein fehlendes Verb auch bedeuten kann, daß ein anderer Sinnzusammenhang vorliegt und dies nicht unbedingt ein Eingabefehler ist.

7.4 Die Antwort des Systems

Die Antwort des Systems ist im besten Fall der Graph einer Funktion. Ist die F-Struktur nicht wohlgeformt und können die fehlenden Komponenten nicht aus der Wissensbasis ergänzt werden, muß das System auf die jeweilige Situation reagieren. Eine Nachfrage an den Nutzer muß im Rahmen dieser Arbeit mit einigen vorgefertigten Sätzen behandelt werden, die auf die Situationen zugeschnitten sind, die bei der semantischen Analyse vorkommen können.

Eine Sprachgenerierung im klassischen Sinn ist sehr komplex und würde zum einen noch Erläuterung von sehr viel Grundlagenwissen erfordern und zum anderen müßten sehr viel umfangreichere semantische Strukturen vorliegen, z.B. unter Verwendung von semantischen Netzen, Schlußmethoden und Prädikatenlogik. Eine sehr gute Einführung findet sich in [23] und eine umfangreichere ist in [10] zu finden.

7.5 Erweiterbarkeit der Grammatik und Anwendung auf weitere Verben

Die Beispielgrammatik ist im Wesentlichen auf die zwei behandelten Beispielsätze beschränkt. In gleicher Form können andere Wörter in das Lexikon eingetragen werden. Für Verben mit

verschiedenen Subkategorisierungen müssen weitere Produktionsregeln erstellt werden und mit den entsprechenden Routinen des Systems verknüpft werden. Im Rahmen des Expertensystems ScienceAtlas ist das Verb *zeichnen* bis jetzt nur mit dem Plotten einer Funktion verknüpft. Sollen Verben implementiert werden, die mehrere Verwendungen innerhalb des Systems finden (was bei einer Implementierung die Regel sein wird), kann anhand der Typen [TPY] des Eingabesatzes festgestellt werden, wie mit diesen in Verbindung mit dem Verb und den Ergebnissen der F-Struktur-Analyse verfahren werden kann.

Sobald die Grammatik mit den Imperativsätzen korrekt arbeitet und Anweisungen an das System zufriedenstellend ausgeführt werden, kann die Grammatik mit Aussagesätzen erweitert werden. Eine Verknüpfung von natürlicher Sprache und einem Schlußfolgerungssystem könnte der nächste Schritt zur Erweiterung des Systems sein.

8 Zusammenfassung und Ausblick

Es wurde gezeigt, daß eine natürlichsprachliche Analyse für eingeschränkte Zusammenhänge im Rahmen des Expertensystems ScienceAtlas durchaus möglich ist. Natürlichsprachliche Eingaben können syntaktisch und semantisch so analysiert werden, daß eine Verknüpfung mit den Systemfunktionen sinnvoll umgesetzt werden kann.

Nach der Besprechung von Konstituentenstrukturen, Grammatiken und funktionalen Strukturen in Form von Attribut-Wert-Matritzen fällt auf, daß eine solche natürlichsprachliche Analyse aufwändig und sehr komplex ist.

Es ist durchaus möglich, die Funktionalität des entwickelten Entwurfs auch mit dem momentan genutzten Parser zu implementieren. Das bisherige System könnte mit Schemata wie

```
{Z,z}eichne [mir, uns, meiner Mutter ][die ][Funktion, Abildung] §term(x)[?]
```

erweitert werden. Die Funktion des Systems wäre so gleichbleibend schnell und sogar die Kommunikation mit dem Benutzer wäre – wenn dieser die Eigenheiten des Systems kennt – gar nicht so unkomfortabel.[1]

Diese Art des Parsings ist jedoch sehr schlecht erweiterbar. Wenn es nicht nur darum geht, Anweisungen des Nutzers auf Übereinstimmung mit den Schemematches zu prüfen und diese Auszuführen, sondern den Nutzer richtiggehend zu *verstehen*, ist eine Erweiterung nicht realisierbar.

Das System könnte bei Weiterführung dieses Entwurfs aus den ihm zugänglichen Daten komplexere Zusammenhänge erkennen, diese speichern und darauf zurückgreifen und auch Schlüsse daraus ziehen. Aus diesem dann sehr viel komplexeren Wissen können auch komplexe Antworten *generiert* werden. Der Nutzer kann sich mit dem System *unterhalten*. Ein weiterer Schritt im Bereich der Kommunikation zwischen Mensch und Maschine könnte dann sein, dem virtuellen Gegenüber eine Art Persönlichkeit zu verleihen und ihm tatsächlich zu erlauben, den Nutzer bei jeder Gelegenheit zu korrigieren.[2]

[1] In diesem Zusammenhang ist die Arbeit von Lutz Prechelt aus dem Jahr 1989 interessant, in der er einen „Fallschablonenzerteiler für Deutsch“ entwickelt [25] – ein sehr schnelles System zur Einordnung von deutscher Sprache in ein sehr umfangreiches Schablonensystem.

[2] Ein solches Feature sollte tunlichst abschaltbar sein ☺

Der grundsätzlich natürlichsprachliche Ansatz des Expertensystems ScienceAtlas konnte weitergeführt und durchaus Perspektiven für weitere natürlichsprachliche Ansätze erarbeitet werden. Eine Implementation des entwickelten Entwurfs könnte nun Schritt für Schritt erweitert, das Lexikon um weitere Vokabeln ergänzt und diese mit den Funktionen des Systems verknüpft werden. Wird auch die Grammatik entsprechend erweitert und die sprachlichen Möglichkeiten des Systems durch einen Sprachgenerator ergänzt, könnte die natürlichsprachliche Ein- und Ausgabe mit einem Schlußfolgerungssystem kombiniert werden, so daß ein mathematisches Dialog- und Expertensystem entsteht.

9 Anhang

(r1)	SATZ	→	V ↑=↓	(NP) (↑IOBJ) =↓ (↓KAS) = DAT	NP (↑DOBJ) =↓ (↓KAS) = AKK
(r2)	NP	→	N ↑=↓		
(r3)	NP	→	(NOM) ↑=↓	[TYP] ↑=↓	(PP) (↑POBJ) =↓ (↓BEZ TYP) = [RANGE]
(r4)	NOM	→	DET ↑=↓	N ↑=↓	
(r5)	PP	→	P ↑=↓	NP ↑=↓	

Abbildung 9.1: Die Produktionsregeln $\mathcal{P}$ der Grammatik $\mathcal{G}_4$. Angaben in runden Klammern () sind optional.

				Empfänger Thema Bereich
zeichne:	V,	(↑PRÄD)	=	„zeichnen <(↑ IOBJ) (↑ DOBJ) (↑ POBJ)>“
		(↑TEMPUS)	=	PRÄS
		(↑MODUS)	=	IMP
		(↑PER)	=	2
		(↑IOBJ KAS)	=	DAT
		(↑DOBJ KAS)	=	AKK
		(↑DOBJ TYP)	=	[FUNCTION] | FUNKTION
		(↑POBJ TYP)	=	[RANGE] | BEREICH
		(↑LINK)	=	`plot()`
die:	DET,	(↑SPEZ)	=	DEF
		(↑NUM)	=	SG
		(↑GEN)	=	FEM
		(↑KAS)	=	AKK
Funktion:	N,	(↑PRÄD)	=	„Funktion <(↑ BEZ)>“
		(↑NUM)	=	SG
		(↑PER)	=	3
		(↑GEN)	=	FEM
		(↑KAS)	=	AKK
		(↑TYP)	=	FUNKTION
		(↑BEZ TYP)	=	[FUNCTION]
f:	TYP,	(↑TYP)	=	[FUNCTION]
		(↑NAME)	=	„f“
mit:	P,	(↑PRÄD)	=	„mit“
dem:	DET,	(↑SPEZ)	=	DEF
		(↑NUM)	=	SG
		(↑GEN)	=	MAS
		(↑KAS)	=	DAT
Definitionsbereich:	N,	(↑PRÄD)	=	„Definitionsbereich <(↑ BEZ)>“
		(↑NUM)	=	SG
		(↑PER)	=	3
		(↑GEN)	=	MAS
		(↑KAS)	=	DAT
		(↑TYP)	=	BEREICH
		(↑BEZ TYP)	=	[RANGE]
(-5, 5):	TYP,	(↑TYP)	=	[RANGE]
		($\uparrow X_1$VALUE)	=	„-5“
		($\uparrow X_2$VALUE)	=	„+5“

Abbildung 9.2: Das Lexikon $\mathcal{X}$ der Grammatik $\mathcal{G}_4$.

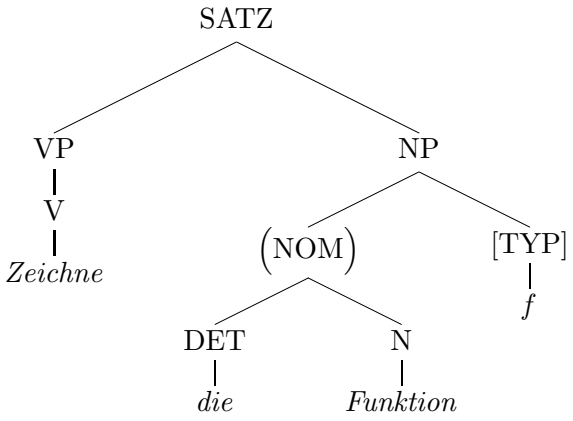

Abbildung 9.3: Die Konstituentenstruktur von Zeichne die Funktion f.

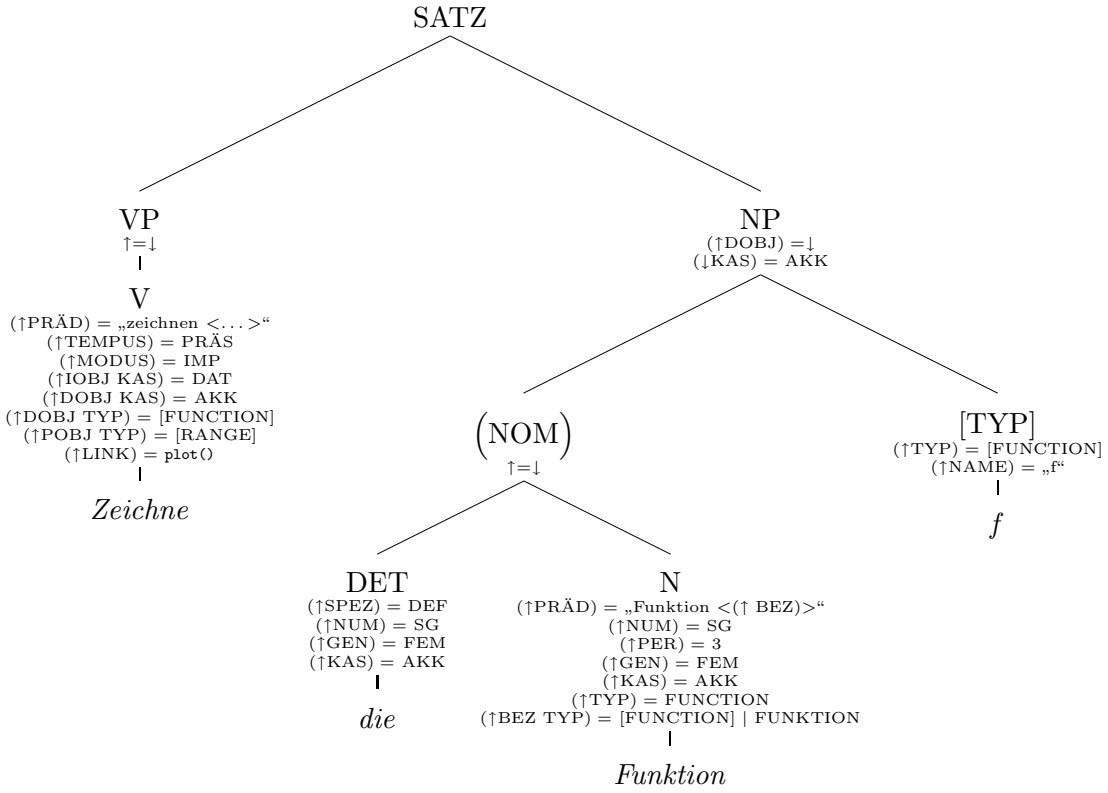

Abbildung 9.4: Die Konstituentenstruktur von Zeichne die Funktion f *mit anotierten funktionalen Schemata.*

$$
f\colon \begin{bmatrix}
\text{PRÄD} & \text{„zeichnen} \left\langle \overset{Empfänger}{\left(f\ \text{IOBJ}\right)}\ \overset{Thema}{\left(f\ \text{DOBJ}\right)}\ \overset{Bereich}{\left(f\ \text{POBJ}\right)} \right\rangle\text{“} \\
\text{DOBJ} & g\colon \begin{bmatrix}
\text{PRÄD} & \text{„Funktion} \left\langle \left(g\ \text{BEZ}\right) \right\rangle\text{“} \\
\text{BEZ} & h\colon \begin{bmatrix} \text{TYP} & \text{[FUNCTION]} \\ \text{NAME} & \text{„f“} \end{bmatrix} \\
\text{SPEZ} & \text{DEF} \\
\text{NUM} & \text{SG} \\
\text{PER} & 3 \\
\text{GEN} & \text{FEM} \\
\text{KAS} & \text{AKK} \\
\text{TYP} & \text{FUNKTION}
\end{bmatrix} \\
\text{TEMPUS} & \text{PRÄS} \\
\text{MODUS} & \text{IND} \\
\text{PER} & 2 \\
\text{LINK} & \texttt{plot}()
\end{bmatrix}
$$

Abbildung 9.5: Die funktionale Struktur des Satzes Zeichne die Funktion f.

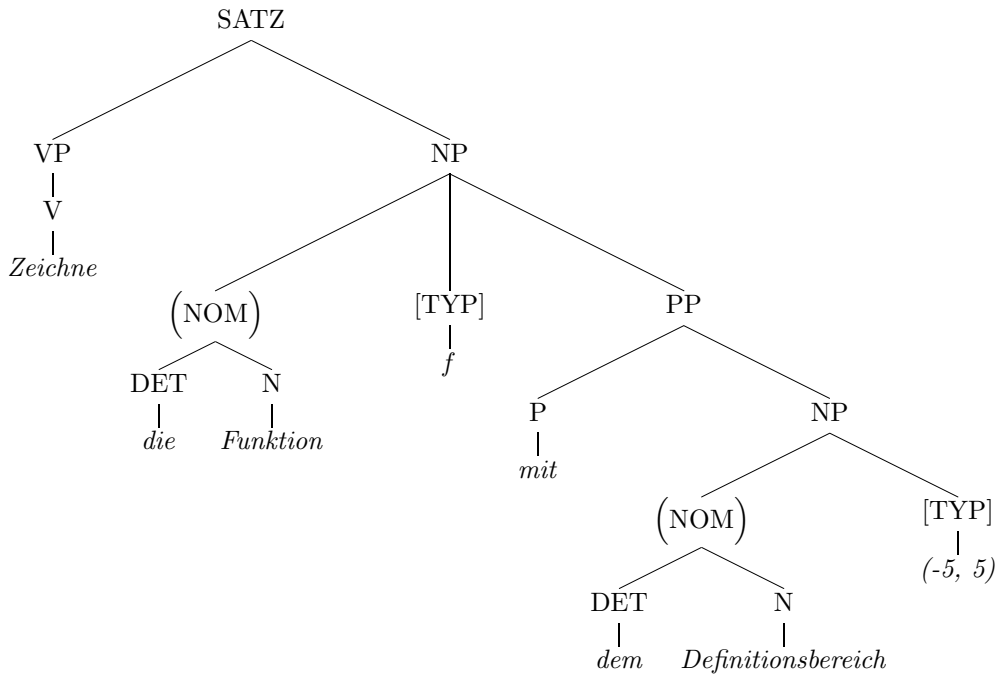

Abbildung 9.6: Die Konstituentenstruktur von Zeichne die Funktion f mit dem Definitionsbereich (-5, 5).

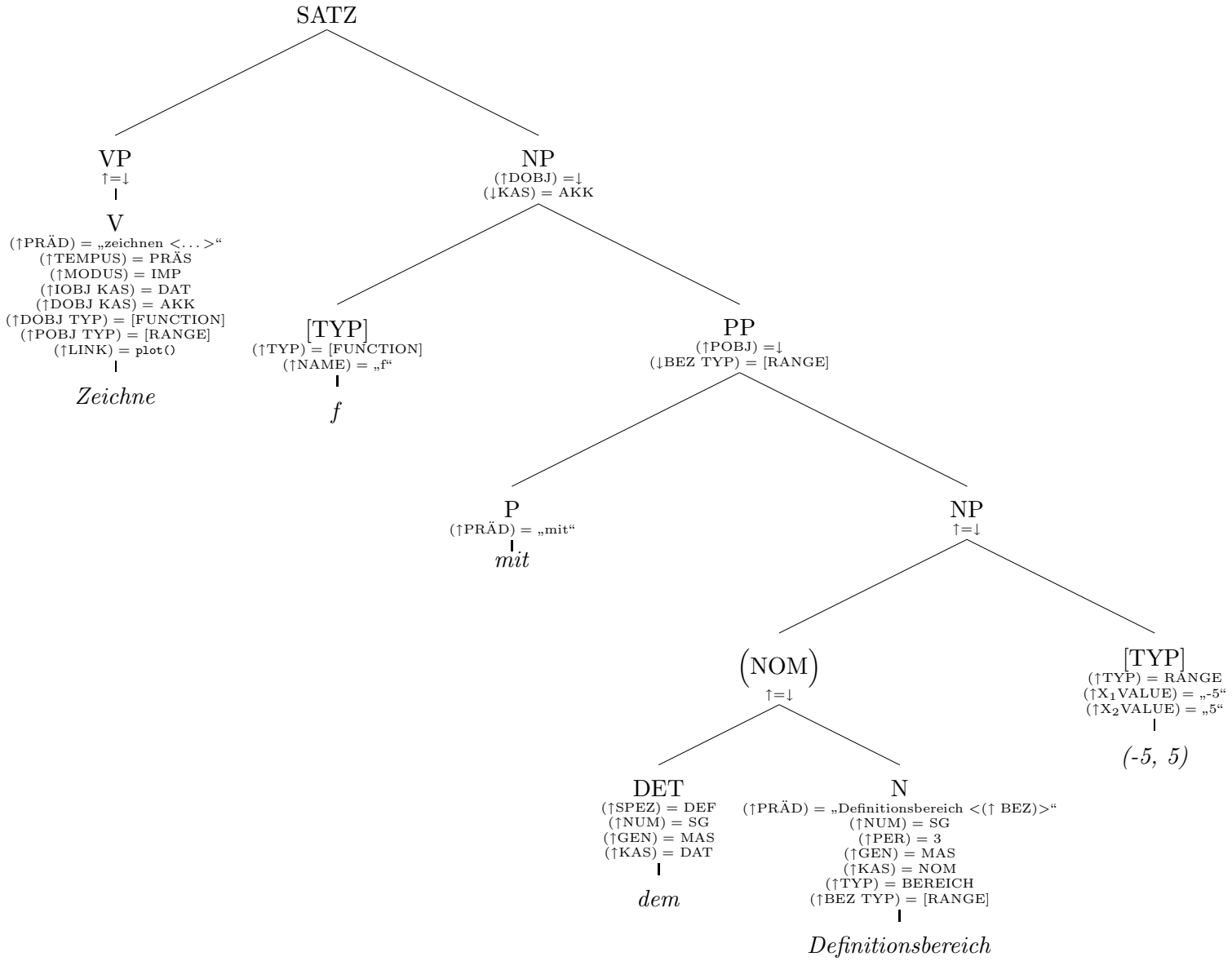

Abbildung 9.7: Die Konstituentenstruktur von Zeichne (die Funktion) f mit dem Definitionsbereich (-5, 5) *mit anotierten funktionalen Schemata. Die optionale NP wurde aus Platzgründen weggelassen.*

$$
f\colon \begin{bmatrix}
\text{PRÄD} & \text{„zeichnen} \left\langle \overset{Empfänger}{\left(f\ \text{IOBJ}\right)}\ \overset{Thema}{\left(f\ \text{DOBJ}\right)}\ \overset{Bereich}{\left(f\ \text{POBJ}\right)} \right\rangle\text{“} \\
\text{DOBJ} & g\colon \begin{bmatrix}
\text{PRÄD} & \text{„Funktion} \left\langle \left(g\ \text{BEZ}\right) \right\rangle\text{“} \\
\text{BEZ} & h\colon \begin{bmatrix} \text{TYP} & [\text{FUNCTION}] \\ \text{NAME} & \text{„f“} \end{bmatrix} \\
\text{SPEZ} & \text{DEF} \\
\text{NUM} & \text{SG} \\
\text{GEN} & \text{FEM} \\
\text{PER} & 3 \\
\text{KAS} & \text{AKK} \\
\text{TYP} & \text{FUNKTION}
\end{bmatrix} \\
\text{POBJ} & i\colon \begin{bmatrix}
\text{PRÄD} & \text{„Definitionsbereich} \left\langle \left(i\ \text{BEZ}\right) \right\rangle\text{“} \\
\text{BEZ} & j\colon \begin{bmatrix} \text{TYP} & [\text{RANGE}] \\ \text{X}_1\text{RANGE} & \text{„-5“} \\ \text{X}_2\text{RANGE} & \text{„5“} \end{bmatrix} \\
\text{PRÄP} & k\colon \begin{bmatrix} \text{PRÄD} & \text{„mit“} \end{bmatrix} \\
\text{SPEZ} & \text{DEF} \\
\text{NUM} & \text{SG} \\
\text{GEN} & \text{MAS} \\
\text{PER} & 3 \\
\text{KAS} & \text{DAT} \\
\text{TYP} & \text{BEREICH}
\end{bmatrix} \\
\text{TEMPUS} & \text{PRÄS} \\
\text{MODUS} & \text{IND} \\
\text{PER} & 2 \\
\text{LINK} & \texttt{plot}()
\end{bmatrix}
$$

Abbildung 9.8: Die funktionale Struktur des Satzes Zeichne die Funktion f mit dem Definitionsbereich (-5, 5).

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Batori, Prof. Dr. Istvan: *Formale Grammatiken für die Beschreibung natürlicher Sprachen.* Universität Koblenz-Landau, 2000.
<http://www.uni-koblenz.de/~compling/Lehre/Veranstaltungen/Skripts/fog.vorlesung.ss2000.ps>.

[2] Baum, Robert: *robertbaum://japan.japanisch.* 2004.
<http://www.robertbaum.de/japan/>.

[3] Benthem, Johan van und Alice ter Meulen (Herausgeber): *Handbook of logic and language.* Elsevier Science B.V., 1997.

[4] Briscoe, Ted und John Caroll: *Generalised probabilistic LR parsing of natural language (corpora) with unification-based grammars.* Computational Linguistics, 19.1:25-59, 1993.

[5] Chomsky, Noam: *Reflexionnen über die Sprache.* Suhrkamp Verlag, Frankfurt am Main, 1977.

[6] Chomsky, Noam: *Lectures on Government and Binding.* Forich, Dodrecht, 1981.

[7] Feldmann, Prof. Anja: *Einführung in die Informatik 4.* TU München, 2004.
<http://www.net.informatik.tu-muenchen.de/teaching/SS04/info4/folien/>.

[8] Gazdar, G., E. Klein, G. Pullun und I. Sag: *Generalized Phrase Structure Grammar.* Blackwell, Oxford, 1985.

[9] Gottlob, Georg: *Expertensysteme.* Springer Verlag, Wien, 1990.

[10] Görz, Günther: *Einführung in die künstliche Intelligenz.* Addison-Wesley, Bonn; Paris, 1995.

[11] Haegman, Liliane: *Introduction to Government and Binding Theory.* Blackwell, Oxford, 1994.

[12] Hopcroft, John D. und Jeffrey D. Ullman: *Formal Languages and their Relation to Automata.* 1969.

[13] Humboldt, Wilhelm von: *Werke: in fünf Bänden*, Band 3: Schriften zur Sprachphilosophie. Wissenschaftliche Buchgesellschaft, Darmstadt, 4., unveränderte Auflage, 1963.

[14] Incors: *Mainichi-Kanji*. INCORS GmbH, Berlin, 2001.
<http://mainichi.incors.de/>.

[15] Joshi, Aravind K.: *An Introduction to Tree Adjoining Grammars*. Technical Report No. MS-CIS-86-64, Department of Computer and Information Science, University of Pennsylvania, 1986.

[16] Klenk, Ursula: *Generative Syntax*. Gunter Narr Verlag, Tübingen, 2003.

[17] Künneth, Thomas: *Einführung in die Korpuslinguistik I*. Universität Erlangen, 2001.
<http://www.linguistik.uni-erlangen.de/kursmaterial/SS01_EMSV_Kuenneth/030501.pdf>.

[18] Lenders, Prof. Dr. Winfried: *Grundlagen der Computerlinguistik*. Universität Bonn, 2004.
<http://www.ikp.uni-bonn.de/dt/lehre/materialien/grundlCL/>.

[19] Müller, Stefan: *Deutsche Syntax deklarativ. Head-Driven Phrase Structure Grammar für das Deutsche*. Niemeyer, Tübingen, 1999.

[20] Naumann, Sven: *Generalisierte Phrasenstrukturgrammatik: Parsingstrategien, Regelorganisation und Unifikation*. Niemeyer, Tübingen, 1988.

[21] Naumann, Sven und Hagen Langer: *Parsing – eine Einführung in die maschinelle Analyse natürlicher Sprache*. Teubner, Stuttgart, 1994.

[22] Pollard, Carl und Ivan A. Sag: *Head-Driven Phrase Structure Grammar*. The University of Chicago Press, Chicago, 1994.

[23] Potthast, Roland: *Expertensysteme*. Universität Göttingen, 2004.

[24] Potthast, Roland: *ScienceAtlas – Science and Applications Management and Modelling Interface*. Universität Göttingen, 2004.
<http://www.ScienceAtlas.de>.

[25] Prechelt, Lutz: *Ein Fallschablonenzerteiler für Deutsch*. Universität Karlsruhe, 1989.
<http://page.mi.fu-berlin.de/~prechelt/Biblio/dipl.pdf>.

[26] Reif, Gerald: *Moderne Aspekte des Wissensverarbeitung – Ein interaktiver Lernbehelf für das Web Based Training.* Technische Universität Graz, 2000.
<http://www.iicm.edu/greif/thesis.ps.zip>.

[27] Rieger, Prof. Dr. Burghard: *Semiotisch Kognitive Informationsverarbeitung (SCIP).* Universität Trier, 2003.
<http://www.ldv.uni-trier.de/~scips/deutsch/background/SCIPtut1.pdf>.

[28] Stetter, Christian: *Schrift und Sprache.* Suhrkamp Verlag, Frankfurt am Main, 1977.

[29] Stolcke, Andreas: *An efficient probabilistic context-free parsing algorithm that computes prefix probabilities.* Forschungsbericht TR-93-065, International Computer Science Institute. SRI International, Berkeley, Kalifornien, 1994.

[30] Switzer, Robert: *Compilerbau.* Universität Göttingen, 2003.

[31] Tomita, Masaru: *Efficient Parsing for Natural Languages: A Fast Algorithm for Practical Systems.* Kluewer, Boston, 1986.

[32] Tomita, Masaru: *An Efficient Augmented-Context-Free Parsing Algorithm.* Computational Linguistics, 13.1/2:31-416, 1987.

Danksagung

Prof. Dr. Roland Potthast möchte ich an dieser Stelle für seine kontinuierliche Betreuung, für das interessante Thema und für eine Einführung in das Expertensystem *ScienceAtlas* danken.

Herrn Priv.-Doz. Dr. habil. Carsten Damm danke ich für die Zweitkorrektur sowie für seine freundliche Unterstützung während meines Studiums.

Prof. Dr. Robert Switzer danke ich herzlich für alle seine hervorragenden Vorlesungen, für seine detaillierten Scripte und für seinen feinen Humor.

Weiterhin danke ich meinen Korrekturleserinnen und Korrekturlesern insbesondere meiner Mitbewohnerin Ameli Stock, meinem Freund und Kommilitonen Frank Schwichtenberg und Antonia Blanke.

Zu guter Letzt möchte ich noch meinen übrigen Mitbewohnerinnen und Mitbewohnern danken für ertragene Launen und für mildernde Umstände beim Bestrafen von Nicht-Abwaschen und ähnlichen Vergehen, und nicht zuletzt meiner Tochter Ronja für ertragene Launen und meinem kleinen Freund Emil, der etliche zusätzliche Kitzel-Tiraden ertragen mußte. ☺