Task-based Parser Output Combination: Workflow and Infrastructure

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart
zur Erlangung der Würde eines Doktors der Philosophie (Dr. phil.) genehmigte Abhandlung

Vorgelegt von
Kerstin Eckart
aus Stuttgart-Bad Cannstatt

Hauptberichter:	Prof. Dr. Jonas Kuhn
1. Mitberichter:	Prof. Dr. Ulrich Heid
2. Mitberichter:	Prof. Dr. Andreas Witt

Tag der mündlichen Prüfung: 04.12.2017

Institut für Maschinelle Sprachverarbeitung der Universität Stuttgart
2018

Hiermit versichere ich, diese Arbeit selbständig verfasst und nur die angegebenen Quellen benutzt zu haben.

(I hereby declare that I have created this work completely on my own and used no other sources than the ones listed.)

_____

Kerstin Eckart

Für
Anna, Hermann und Käthe

Contents

List of abbreviations

ASR	Automatic Speech Recognition
B3DB	B3 database
CLARIN	Common Language Resources and Technology Infrastructure
CMDI	Component Metadata Initiative
CoNLL	Conference on Computational Natural Language Learning
CYK	Cocke-Younger-Kasami algorithm
DCMI	Dublin Core Metadata Initiative
DIRNDL	Discourse Information Radio News Database for Linguistic analysis
DRT	Discourse Representation Theory
EAGLES	Expert Advisory Group on Language Engineering Standards
GATE	General Architecture for Text Engineering
GOLD	General Ontology for Linguistic Description
GrAF	Graph Annotation Format
GUI	Graphical User Interface
HPSG	Head-driven Phrase Structure Grammar
ICARUS	Interactive platform for Corpus Analysis and Research tools, University of Stuttgart
ID	Identifier
IEC	International Electrotechnical Commission
INESS	Infrastructure for the Exploration of Syntax and Semantics
ISO	International Organization for Standardization
LAF	Linguistic Annotation Framework
LAUDATIO	Long-term Access and Usage of Deeply Annotated

	Information
LFG	Lexical Functional Grammar
OLAC	Open Language Archives Community
OLiA	Ontologies of Linguistic Annotation
OWL	Web Ontology Language
PASSAGE	Produire des Annotations Syntaxiques à Grande Échelle ('Large scale production of syntactic annotations')
PAULA	Potsdamer Austauschformat Linguistischer Annotationen ('Potsdam Exchange Format for Linguistic Annotations')
PID	Persistent Identifier
POS	part-of-speech
RDF	Resource Description Framework
ROVER	Recognizer Output Voting Error Reduction
SALTO	The SALSA Tool (referring to the SALSA XML format from the *The Saarbrücken Lexical Semantics Annotation and Analysis Project*)
SANCL	Syntactic Analysis of Non-Canonical Language (Workshop)
SFB	Sonderforschungsbereich (collaborative research centre)
SGML	Standard Generalized Markup Language
SPMRL	Statistical Parsing of Morphologically Rich Languages (Workshop)
SQL	Structured Query Language
STTS	Stuttgart-Tübingen-TagSet
TCF	Text Corpus Format
TEI	Text Encoding Initiative
UIMA	Unstructured Information Management Architecture
URL	Uniform Resource Locator
WaCky	The Web-As-Corpus Kool Yinitiative
WebLicht	Web-based Linguistic Chaining Tool
XLE	Xerox Linguistic Environment
XML	Extensible Markup Language

Abstract

This work introduces the method of task-based parser output combination as a device to enhance the reliability of automatically generated syntactic information for further processing tasks. The approach is based on two assumptions: (i) Parsing is not an isolated task, but often one step in a processing workflow. From this step, the necessary syntactic information is extracted and then applied in e.g. semantic analyses or information extraction. (ii) Parsing, as an automatic processing step to retrieve syntactic information, cannot be perfect, especially when perfect means a unique, correct and fully specified analysis for each utterance, which in addition also reflects the intention of the author. This is due to the fact that the correctness of an analysis can only be defined with respect to a syntactic theory or a grammar and it is also due to the ambiguity of natural language, which can often only be resolved by world knowledge or with the help of points of reference between the communicating individuals (common ground). Additionally, factors like creation time, genre and the relationship of the communicating individuals highly influence the lexical and structural choices.

Parsers, i.e. tools generating syntactic analyses, are usually based on reference data. Typically these are modern news texts. However, the data relevant for applications or tasks beyond parsing often differs from this standard domain, or only specific phenomena from the syntactic analysis are actually relevant for further processing. In these cases, the reliability of the parsing output might deviate essentially from the expected outcome on standard news text.

Studies for several levels of analysis in natural language processing have shown that combining systems from the same analysis level outperforms the best involved single system. This is due to different error distributions of the

involved systems which can be exploited, e.g. in a majority voting approach. In other words: for an effective combination, the involved systems have to be sufficiently different.

In these combination studies, usually the complete analyses are combined and evaluated. However, to be able to combine the analyses completely, a full mapping of their structures and tagsets has to be found. The need for a full mapping either restricts the degree to which the participating systems are allowed to differ or it results in information loss. Moreover, the evaluation of the combined complete analyses does not reflect the reliability achieved in the analysis of the specific aspects needed to resolve a given task.

This work introduces task-based parser output combination as a method and presents an abstract workflow which can be instantiated based on the respective task and the available parsers. The approach focusses on the task-relevant aspects and aims at increasing the reliability of their analysis. Moreover, this focus allows a combination of more diverging systems, since no full mapping of the structures and tagsets from the single systems is needed. The usability of this method is also increased by focussing on the output of the parsers: It is not necessary for the users to reengineer the tools. Instead, off-the-shelf parsers and parsers for which no configuration options or sources are available to the users can be included. Based on this, the method is applicable to a broad range of applications. For instance, it can be applied to tasks from the growing field of Digital Humanities, where the focus is often on tasks different from syntactic analysis.

To begin with, this thesis identifies some basic terms and concepts and gives an introduction to the technical background. Following this, related work regarding system combination is discussed, and a brief glance at task-based approaches is added.

A basic theoretical part of this work refers to interoperability of resources. Different notions of interoperability are contrasted and two levels of categories of interoperability are presented. These categories distinguish on the first level between representational interoperability and content-related interoperability and on the second level between structural aspects of interoperability and concept-related aspects of interoperability. Representational interoperability can

often be achieved, since it depends on technical decisions, but not on linguistic decisions. As a consequence, these representational aspects can be treated separately such that the focus is on the actual content-related differences: linguistic decisions regarding structure and concepts. Based on these categories, a classification of combination types is introduced, which captures combination settings ranging from the combination of very similar systems to the combination of systems diverging considerably in their applied structures and concepts. While several theoretical approaches can be linked to the presented categories of interoperability, the classification of combination types provides application-relevant grades for the expected dissimilarity of the combined systems.

To foster representational interoperability for several analyses, the approach of the relational database B3DB is proposed as an infrastructure. Differing analyses can be mapped onto the generic data structures of this database. The database also represents process metadata, such that the processes leading to the analyses can be captured and compared. The B3DB does thus not only support the combination approach but fosters also sustainability of the data.

All steps which are part of the workflow for task-based parser output combination are presented in detail, and the prerequisites for the application of the workflow are discussed. Subsequently, the abstract workflow is instantiated for two different example tasks to illustrate its application. Several types of combinations are evaluated based on the task. This shows that along the grades of the classification of combination types the outcome of the overall task can be enhanced as soon as the systems are sufficiently different. In the two example tasks, this is the case already for a combination of outputs with similar structures but different concepts.

In summary, this work contributes to research debates on interoperability and sustainability, combination approaches and task-based evaluation. Next to proposing refined categories of interoperability, a respective classification of combination types and a generic database as infrastructure, task-based parser output combination is introduced as a method. An abstract workflow for this method is presented in detail. Users can instantiate this workflow for several applications, tasks and available parsing systems, independently of the main focus of their task and their parsing expertise.

Zusammenfassung

Im Rahmen dieser Arbeit wird die aufgabenbasierte Kombination von Parserausgaben als Methode eingeführt, um die Zuverlässigkeit von automatisch erstellten syntaktischen Informationen für die weiterführende Prozessierung zu erhöhen. Der Arbeit liegen dabei zwei Annahmen zugrunde: (i) Eine syntaktische Analyse ist in vielen Fällen keine isolierte Aufgabe, sondern ein Schritt in einem Arbeitsablauf. Aus diesem Schritt werden die notwendigen syntaktischen Informationen extrahiert, die in weiterführenden Schritten, z.B. für darauf aufbauende semantische Analysen oder die Informationsextraktion, benötigt werden. (ii) Eine automatische syntaktische Analyse kann nicht perfekt sein, wenn wir unter einer perfekten Analyse genau eine, korrekte, vollständig spezifizierte und die Intention des Autors wiedergebende Darstellung für eine beliebige natürlichsprachliche Äußerung verstehen. Dem steht zum einen entgegen, dass sich die Korrektheit der Analyse immer auf eine bestimmte zugrundeliegende Theorie bezieht und zum anderen, dass natürliche Sprache einen hohen Grad an Ambiguität aufweist, die unter Umständen nur durch Weltwissen oder gemeinsame Annahmen der Kommunikationspartner aufgelöst werden kann. Dazu kommt, dass Faktoren wie Zeitpunkt, Genre und Beziehung der Kommunikationspartner zueinander entscheidenden Einfluss auf das verwendete Vokabular und die syntaktischen Strukturen haben.

Vorhandene Werkzeuge zur automatischen syntaktischen Analyse („Parser“) bauen meist auf bestimmten Referenzdaten, zumeist modernen Nachrichtentexten, auf. Für eine Vielzahl von Anwendungsgebieten, deren Datengrundlage davon abweicht, oder die nur Bedarf an der Analyse bestimmter syntaktischer Phänomene haben, kann die Zuverlässigkeit der entsprechenden Ausgabe von

den bei Auswertung der Gesamtanalyse auf kanonischen Nachrichtendaten zu erwartenden Werten stark abweichen.

Studien zu verschiedenen Analyseebenen in der Computerlinguistik haben gezeigt, dass eine Kombination von Analysesystemen einer Ebene nicht nur für die Syntaxanalyse die Qualität des Ergebnisses gegenüber den einzelnen beteiligten Systemen erhöht. Der Grund dafür ist, dass verschiedene Systeme meist auch eine unterschiedliche Fehlerverteilung aufweisen, was wiederum z.B. in einem Mehrheitsentscheid verschiedener Systeme ausgenutzt werden kann. Um dieses Verfahren erfolgreich einzusetzen, müssen die Systeme hinreichend unterschiedlich sein.

Für diese Kombinationsansätze werden für gewöhnlich die kompletten Ausgaben der einzelnen Systeme kombiniert und evaluiert, was zum einen die Verschiedenartigkeit der beteiligten Systeme beschränkt oder Informationsverlust zur Folge hat, da eine automatisierte Abbildung der verwendeten Strukturen und Konzepte in den Analysen gegeben sein muss. Zum anderen geben diese Analysen keinen Aufschluss darüber, inwiefern sich die Verbesserung des Gesamtergebnisses auf die für eine bestimmte Folgeaufgabe benötigten Aspekte bezieht.

Diese Arbeit führt die Methode der aufgabenbasierten Kombination von Parserausgaben ein und stellt einen abstrakten Ablaufplan zur Verfügung, der passend für die jeweils vorliegende Anwendung sowie die verfügbaren Parser instantiiert werden kann. Dabei wird speziell die Zuverlässigkeit der Analyse der anwendungsrelevanten Aspekte erhöht. Desweiteren ermöglicht die Methode gerade durch den Fokus auf bestimmte Aspekte die Kombination zuvor nicht kompatibler Systeme, da keine komplette Abbildung der verwendeten Strukturen und Konzepte gefordert wird. Schließlich wird die Anwendbarkeit der Methode auch dadurch erhöht, dass sich die Kombination auf die Ausgaben der Parser bezieht, vom Anwender also kein Eingreifen in das Werkzeug selbst gefordert wird. Damit wird zum einen ermöglicht, dass verschiedene verfügbare Standardwerkzeuge direkt eingebunden werden können, ebenso wie Werkzeuge, zu denen den Anwendern kein Quellcode und keine Konfigurationsmöglichkeiten zur Verfügung stehen. Zum anderen steht die Methode damit auch für ein weites Spektrum an Aufgabengebieten, z.B. aus den sich

entwickelnden Digital Humanities zur Verfügung, da als eigentlicher Fokus die Anwendung und nicht die Syntaxanalyse im Mittelpunkt steht.

Die vorliegende Arbeit befasst sich zunächst mit der Bestimmung einiger grundlegender Konzepte sowie der Einführung des technischen Hintergrunds als Arbeitsgrundlage. Danach werden verwandte Arbeiten zu Kombinationsstudien diskutiert, an die sich eine kurze Betrachtung zu aufgabenbasierten Ansätzen anschließt.

Als theoretische Grundlage wird der Begriff der Interoperabilität von Ressourcen untersucht und eine zweistufige Kategorisierung von Interoperabilitätsaspekten vorgestellt. Dabei wird zunächst die repräsentationsbezogene Interoperabilität, die nicht von linguistischen sondern nur von technischen Entscheidungen motiviert ist, von inhaltlicher Interoperabilität unterschieden. Da Erstere für die Kombinationsansätze oft herstellbar ist, hilft die Kategorisierung der Unterschiede zur Abtrennung dieser rein repräsentationellen Aspekte und zur Fokussierung auf die relevanten linguistischen Unterschiede in Strukturentscheidungen und verwendeten Konzepten. Um die einzelnen Aspekte besser herausarbeiten zu können, wird dazu noch zwischen der Kategorie der Strukturinteroperabilität und der Kategorie der Interoperabilität von Konzepten unterschieden. Auf dieser Grundlage wird eine Klassifikation von Kombinationstypen vorgestellt, die von einer Kombination sehr ähnlicher Systeme bis zur Kombination von Systemen mit abweichenden Strukturen und Konzepten reicht. Während sich in der Kategorisierung der Interoperabilitätsaspekte verschiedene vorhandene Ansätze verorten lassen, gibt die Klassifikation von Kombinationstypen direkt Aufschluss über die zu erwartende Abweichung der kombinierten Systeme untereinander.

Um repräsentationsbezogene Interoperabilität für verschiedene Ausgaben bereits mithilfe der verwendeten Infrastruktur herstellen zu können, wird der Ansatz der relationalen B3-Datenbank (B3DB) vorgeschlagen. Auf deren generische Datenstrukturen können nicht nur die Analysen abgebildet werden, es ist ebenfalls möglich Prozessmetadaten abzulegen, so dass Abläufe zur Erstellung der Analysen geeignet nachvollzogen und verglichen werden können. Die Datenbank trägt damit nicht nur zur Kombinierbarkeit der Analysen, sondern auch zur Nachhaltigkeit der erstellten Daten bei.

Alle Schritte im abstrakten Ablaufplan für die zentrale Methode, die aufgabenbasierte Kombination von Parserausgaben, werden im Detail vorgestellt, und Voraussetzungen für die Anwendbarkeit der Methode werden diskutiert. Zuletzt werden zwei unterschiedliche Beispielanwendungen vorgestellt, für die der abstrakte Ablaufplan instantiiert und mit Kombinationsansätzen verschiedenen Typs aufgabenspezifisch evaluiert wird. Dabei zeigt sich, dass sich entlang der auf Basis der Interoperabilitätskriterien definierten Kombinationstypen eine Verbesserung der Ergebnisse für die jeweilige Anwendung erreichen lässt, sobald die Systeme ausreichend verschieden sind. In den vorgestellten Beispielanwendungen ist dies bereits bei der Kombination von Ausgaben mit ähnlichen Strukturen aber unterschiedlichen Konzepten der Fall.

Insgesamt trägt die vorliegende Arbeit damit zu Forschungsdebatten über Interoperabilität und Nachhaltigkeit, Kombinationsansätze sowie aufgabenbasierte Evaluation bei und stellt neben einer detaillierten Kategorisierung von Interoperabilitätsaspekten, einer darauf aufbauenden Klassifikation für Kombinationsansätze und einer generischen Infrastruktur insbesondere die Methode der aufgabenbasierten Kombination von Parserausgaben zur Verfügung. Anhand des Ablaufplans für diese Methode kann der Ansatz von Anwendern mit unterschiedlichem Fokus für verschiedene Anwendungen und verschiedene verfügbare Parser instantiiert werden.

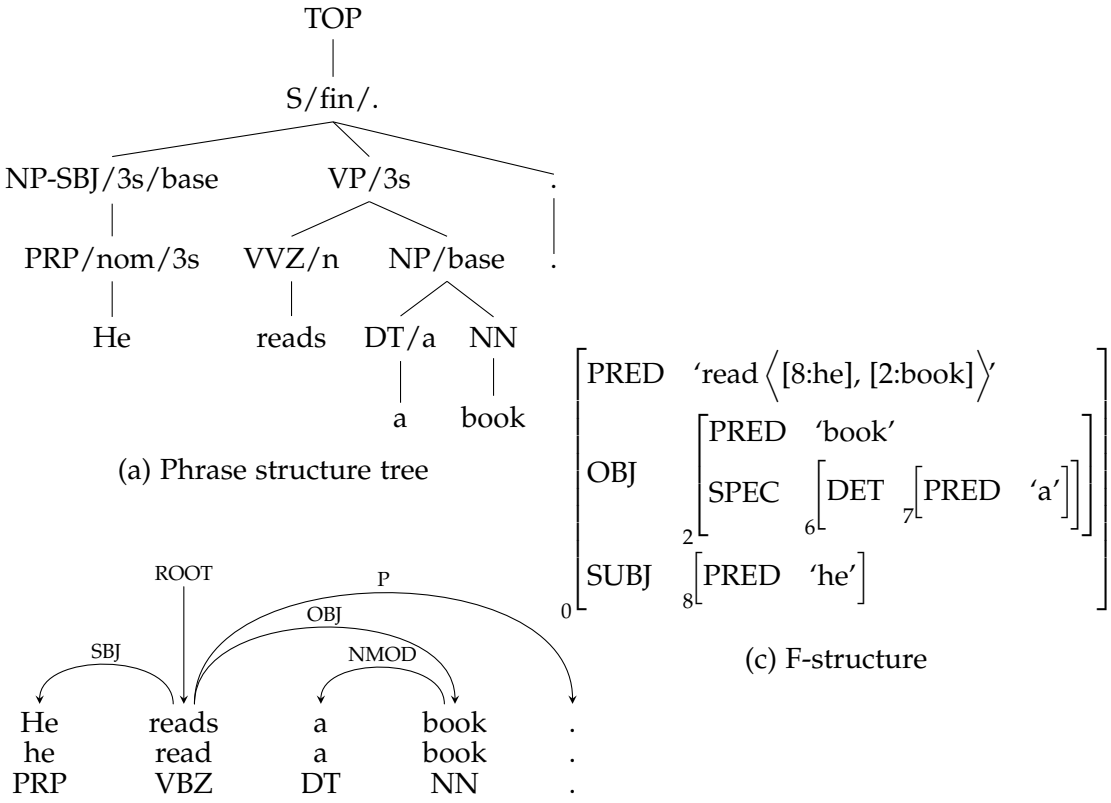
Chapter 1

Introduction

One of the main objects of research in computational linguistics is how to model aspects of language analysis by taking computational methods into account. Traditionally, these analyses treat language as a multi-layer system, splitting it into phenomena-based description levels such as phonology, morphology, syntax or semantics.

Parsing, in computer science, is a term for methods to determine and validate the structure of an input expression based on a formal specification, especially in the context of the syntactic analysis of source code with respect to the formal grammar of a programming language (Duden Informatik: p. 650). In computational linguistics the term describes analysis processes identifying structure of natural language input on several layers such as morphological parsing, syntactic parsing or semantic parsing. When the term appears without closer specification in a computational linguistic context, it usually refers to syntactic parsing, i.e. the process of identifying the syntactic structure of a sentence. The term is applied in that way throughout this thesis. And since (syntactic) parsing and automatic parsing systems, called parsers, are of major importance for this work the following shows some examples.

Figure 1.1 shows three different examples for syntactic analyses, each produced by an automatic parsing system for the same input sentence shown in Example (1.1) but based on different formalisms and theoretical paradigms describing sentence structure. Figure 1.1a is a phrase structure tree, an analy-



(a) Phrase structure tree

(c) F-structure

(b) Dependency tree

Figure 1.1: Syntactic analyses generated by different parsers.

sis generated by the constituency parser BitPar (Schmid 2004, 2006) based on a probabilistic context-free grammar. Figure 1.1b shows an analysis from a data-driven dependency parser which is part of the Mate Tools (Bohnet 2010). The analysis in Figure 1.1c is an F-structure, generated by an LFG parser via XLE-Web (INESS, Rosén et al. 2012).

(1.1) He reads a book.

In computational approaches, findings from the analysis of one level of linguistic description usually influence the analysis of other levels. The output of a parser is for example often used as the basis for semantic analyses. Moreover the identified syntactic features, such as information on the subject (SBJ) and

the object (OBJ) from Figure 1.1b, often serve as an input for other language-related applications, e.g. machine translation, automatic question answering, text summarization and search engines. Hence, the first assumption on which this work is based, is that **parsing is not an isolated task**, since syntactic information is further needed in applications and linguistic studies based on corpus data.

Under formal considerations one might expect that each parser output would represent the (i) single, (ii) correct, and (iii) fully specified syntactic analysis of the input sentence. However, even if competent speakers are asked to assign syntactic analyses, this requirement is difficult to meet, not even including the problem of finding an appropriate computational modelling. So before discussing the influence of the computational aspect, we will first have a short look at each of these three required aspects in general. First of all, language contains ambiguities. And since it is not even clear if human language producers or recipients necessarily resolve all these ambiguities in a given context, it might be hard to identify an analysis as the only appropriate one. The second aspect, correctness, is also hard to specify. Ambiguities, if resolved, might yet be resolved differently by the producer and a recipient depending on their knowledge and on the situation. Moreover there have been decades of discussion and various linguistic approaches to find a common description of natural language syntax, not only including the search for a general description of specific syntactic phenomena, but also taking into consideration which phenomena should be subject to a syntactic analysis at all. The same linguistic considerations influence the third aspect, full specification. Thus, the notion of a correct or fully specified analysis cannot be discussed in a general way, but depends on the specific syntactic theory or setting applied.

Adding a computational aspect brings another edge to the situation. The advantage of computational processing is usually that in a given span of time, a lot more data can be analysed computationally than manually by humans. Furthermore, computational analyses tend to be more consistent, whether in success or failure, since they do not suffer from fatigue or training bias, unless these aspects are simulated intentionally. However, computational methods utilizing even the largest sets of processors and storage capacities have not gone

beyond the Turing Test[1], and for the time being it is hard for a computational tool to even roughly simulate the flexibility of human language processing.

An automatic syntactic analysis is usually based on a computationally processable grammar or a statistical language model and sometimes supported by additional knowledge bases (e.g. lexicons) or analyses from other linguistic levels (e.g. morphology). When evaluating the results of these systems, the underlying theory, grammar or annotation scheme are not questioned but it is assessed how well the system was able to reproduce an expected analysis.

But even if the theoretical setting is not under consideration, it is often hard to find exactly the expected analysis in a large space of candidate analyses. Especially ambiguities constitute a major problem for automatic systems. Parsers usually only take the sentence context into account and do not have access to any representation of world knowledge. Thus important factors for a disambiguation are not available to the system. Additionally, the expected analysis, which is fixed independently in a manual annotation process prior to the evaluation, is not necessarily uncontroversial, despite the decision for a specific theory.

Another aspect is that natural language is not static and thus the notion of what is grammatical and the range of language which is produced or considered understandable by humans shifts over time and varies based on the communication context. For example, abbreviations used in today's online communication would not have been considered understandable in the 1980s and a parsing system whose statistics are based on newspaper text from the late twentieth century or whose rules are based on the grammaticality of this time will not be able to sufficiently represent syntactic information neither from text produced in online chats – nor from a drama of 1808. Register, creation time, genre and degrees of orality are therefore also aspects which influence the expectations towards a syntactic analysis, especially an automatic one.

[1] In a Turing Test, it is the objective of a machine to mislead a judge or interrogator into thinking it is a human by imitating human behaviour in answering questions. The interrogator is usually confronted with two participants, the machine and a human. After a specific amount of time, a decision has to be made based on the conversation with both participants, which is which. The many versions of this test are based on the test proposed by Turing (1950).

Hence, the second basic assumption of this thesis is that **parsing, as an automatic processing step to retrieve syntactic information, cannot be perfect**, since our notion of the perfect syntactic analysis depends on too many context variables. Furthermore the quality of the automatic analysis heavily depends on the system implementation and the data the system is applied to.

However, if parsing is not an isolated task, as was the first assumption, further computational linguistic tasks rely on exactly these non-perfect syntactic analyses, i.e. on the output of the parsers and on the quality of the analyses they produce.

In practice, many parsing systems have been designed and implemented, based on different technical approaches and different linguistic theories. Their output has been used in research and applications, and they have shown different strengths and weaknesses. Many of these parsing systems are valuable tools in a specific theoretical or practical context. Systems which build on statistical analyses of huge data sets tend to be robust, i.e., even if the input does not have a regular structure they are able to produce an analysis. At the same time they heavily depend on the type and quality of the data sets they base their statistics on. Systems which implement linguistic grammars or sets of rules incorporate many well defined linguistic constraints, but are likely to produce no analysis at all, if the input is considered ungrammatical by the underlying theory or if a specific syntactic phenomenon is not covered by the rule set.

In the past there have been approaches to combine available parsing systems to profit from their differing error distributions and to achieve better overall results, cf. Henderson and Brill (1999), Zeman and Žabokrtský (2005), and McDonald and Nivre (2011) amongst others.

This work takes up the direction of the combination approaches and poses the following research question:

Q *In which way can different automatic syntactic analyses be combined to increase the reliability of information propagated to further processing steps?*

That is, how to increase the analysis quality of exactly those bits of information which are utilized in the following processing? This question includes different aspects, which are paraphrased by Q 1 to 4 in the following.

As mentioned above, the usefulness of the combination depends on the different error distributions of the involved systems and consequently on the difference of these systems. Figure 1.1 exemplifies such diverse systems but illustrates also, that they are not interoperable as is, i.e. they do not necessarily identify the same structures or apply the same concepts and are also represented differently. This leads to Q 1:

Q 1 *How can the different systems become interoperable (i) with respect to the linguistic assumptions they are based on and (ii) with respect to the different representations of the linguistic information they apply?*

As also discussed above, it can be hard to determine a single perfect analysis of an input, which leads to Q 2:

Q 2 *If perfectness depends on so many context variables, how can the combination approach be evaluated?*

And since this work targets generic and adaptable solutions, the quintessence is:

Q 3 *Which steps are necessary to find a combination approach which benefits from synergetic effects of the participating systems, and can these steps be defined in an abstract way?*

Additionally, an infrastructural point of view naturally comes into play when a certain amount of data from several systems is included. For the combination approach, multiple analyses for the same input are to be stored and retrieved when needed. This is reflected by Q 4:

Q 4 *What are the requirements an infrastructure needs to fulfil in order to support and document complex procedures as they are targeted by a combination approach?*

Related to the overall research question, of how to combine analyses to increase their reliability, this work opts for a task-based approach. Taking the task into account which actually employs the syntactic information focusses the notion of perfectness of the syntactic analysis on the ability to be helpful for the task (Q 2). It also limits the need for ambiguity resolution and for interoperability of analysis content to those aspects which are directly relevant for the targeted task (Q 1). Nevertheless, as mentioned above, this work targets generic solutions, i.e. abstract realizations which can then be instantiated for a task at hand (Q 3) and a generic infrastructure (Q 4), which does not prefer specific analysis structures, but fosters interoperability where this can be achieved without information loss (Q 1). With respect to infrastructure, this work also argues for the importance of a detailed tracking of workflow steps as part of a thorough documentation.

As main contributions of this work (i) a set of categories of interoperability aspects and a pertaining classification of combination approaches is proposed and (ii) an abstract workflow for task-based parser output combination is developed. A relational database based on generic data structures provides an implementation of the requirements which support the combination approach. The prerequisites and application possibilities of the abstract workflow for task-based parser output combination are discussed by means of two different tasks: a study on the classification of readings of German *nach*-particle verbs (Task I) and the identification of phrases for information status annotation (Task II).

1.1 Outline

The text structure of this book consists of chapters, sections and subsections. Each chapter is devoted to a specific topic; sections and subsections divide the topic of the chapter into several aspects, either sequentially or to present different points of view. Figure 1.2 provides a visual outline of this book in a structural overview based on the chapters and contributions.

In the remainder of this introductory chapter, Section 1.2 presents concepts which are not specific to the topic of a particular chapter but relevant to several

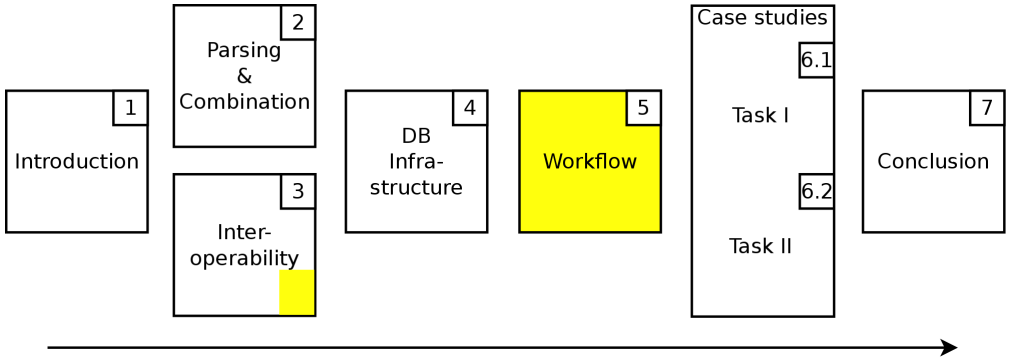


Figure 1.2: Chapter overview.

chapters or to make my view on the background explicit. Since it is exactly the definition of the more general concepts which are often taken for granted, it is easily overlooked that they can still vary in their definition. In addition to the above discussion of the main research questions and thus the objective of this work, Section 1.3 provides a motivation for why this work chooses task-based parser output combination for approaching the objective to find an adequate syntactic analysis for further processing tasks.

The second and third chapter present specific background regarding parsing and interoperability of resources respectively, which are two major topics this work builds on. Based on their similar role in providing specific background and related work, these chapters are displayed in parallel in Figure 1.2. However Chapter 3 additionally works towards one of the main contributions of this work, denoted in Figure 1.2 by the filled box within the chapter box. This contribution is a classification of combination types, based on the interoperability of systems which take part in a combination. The classification is a device to structure the trade-off between combination effort and increase of reliability and is applied in the case studies of task-based parser output combination presented in this work.

The fourth chapter is devoted to the infrastructural contribution. It expands the ground work of Eckart (2009) and describes a database which supports the combination of different systems due to its capacity of handling different

analyses and due to its means of workflow tracking. Chapter 4 can also be read independently of the rest of the work as a documentation regarding requirements for and implementation of the database denominated as B3 database.

Chapter 5 is highlighted in Figure 1.2 to visualize that it presents the other main contribution of this work: an abstract workflow for task-based parser output combination, which is then applied to case studies for two real application scenarios from a linguistic and a computational linguistic framework in Chapter 6. The two extensive corpus linguistic experiments are denoted Task I and Task II. Task I focusses on German *nach*-particle verbs, while Task II concerns phrases for information status annotation.

The final chapter concludes this work with an overview of the contributions and discusses their applicability as well as the contribution of this work to the research debates on interoperability and sustainability, system combination, and extrinsic evaluation.

Many examples in this work will show syntactic analyses which are generated manually or by automatic parsing systems. The relevant parts of these examples will be explained in the text and the utilized tool or tagset will be stated. However, to avoid redundancy, the explanation of the individual tags denoting the linguistic concepts applied in the annotation is given centrally in Appendix A. Similarly, the details of the utilized resources, such as version number or persistent identifier (PID) where available, are given in Appendix B in the order of their appearance in the text. Information on the examples from Figure 1.1 in the introduction is thus given at the top of Appendix B.1.

This work is embedded in the *Sonderforschungsbereich 732 (SFB 732)*[2], a collaborative research centre with four thematic areas and several sub-projects. Parts of this work consequently include collaborations within and between several projects of SFB 732, which allows me among other things to present tasks from these collaborations as case studies. Joint work is highlighted in the introduction of the respective chapters.

[2] *Sonderforschungsbereiche* are funded by the *Deutsche Forschungsgemeinschaft (DFG)*. SFB 732 with the title "Incremental specification in context" was funded in three phases from 2006 to 2018.

Furthermore, parts of this work, some previous work and work with respect to the applied resources has been published in the ways listed in the following.

The ground work for the infrastructural aspect and the B3 database discussed in Chapter 4 has been described in:

- Eckart, K. (2009). Repräsentation von Unterspezifikation in relationalen Datenbanksystemen. Diplomarbeit, Institut für Parallele und Verteilte Systeme, Universität Stuttgart, Germany.
- Eckart, K., Eberle, K., and Heid, U. (2010). An Infrastructure for More Reliable Corpus Analysis. In Bel, N., Hamon, O., and Teich, E., editors, *Web Services and Processing Pipelines in HLT: Tool Evaluation, LR Production and Validation*, pages 8–14, Valletta, Malta. LREC 2010 Workshop.
- Eckart, K. (2012). A standardized general framework for encoding and exchange of corpus annotations: The Linguistic Annotation Framework, LAF. In Jancsary, J., editor, *Proceedings of KONVENS 2012*, pages 506–515. ÖGAI. SFLR 2012 workshop.

The contribution to the *Proceedings of KONVENS 2012* (Eckart 2012) is based on an invited talk in the *Workshop on Standards for Language Resources* and mainly describes work by Nancy Ide and Keith Suderman (e.g. Ide and Suderman 2012). A description of the B3 database is included as a use case.

The approach to interoperability presented in Chapter 3 has been published in:

- Eckart, K. and Heid, U. (2014). Resource interoperability revisited. In *Proceedings of the 12th edition of the KONVENS conference*, volume 1, pages 116–126, Hildesheim, Germany.

Parts of the case studies for Task I (Section 6.1) have been published in the following papers, for which this author provided task-based parser output combination as a method, and set up and discussed the combination rules in collaboration with Wolfgang Seeker.

- Haselbach, B., Eckart, K., Seeker, W., Eberle, K., and Heid, U. (2012a). Approximating theoretical linguistics classification in real data: the case of German *nach* particle verbs. In *Proceedings of COLING 2012*, Mumbai, India.

- Eckart, K. and Seeker, W. (2013). Task-based Parser Output Combination. ESSLLI-13 Workshop on Extrinsic Parse Improvement (EPI). Düsseldorf, Germany.

A pilot study to the task described in Section 6.1, which did not employ parser combination, has been published in the following paper, for which this author provided the database part:

- Haselbach, B., Seeker, W., and Eckart, K. (2012b). German "nach"-Particle Verbs in Semantic Theory and Corpus Data. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey.

The corpora and subcorpora applied in the case studies are described in the following papers: Faaß and Eckart (2013) refers to SdeWaC, which was set up by Gertrud Faaß. This author was involved in the procedure to find parsable sentences by means of their analysis by the dependency parser FSPar[3] and responsible for some final steps with respect to character encoding. Eckart et al. (2012) refers to the DIRNDL corpus for which this author rendered possible a joint querying of several manual and automatic annotations which were based on deviating token layers. This was achieved by means of the database mentioned above, in which a flexible linking between the token layers could be established. Cf. Sections 6.1.1 and 6.2.1 for short descriptions of the resources applied in the case studies, including SdeWaC and DIRNDL.

- Faaß, G. and Eckart, K. (2013). SdeWaC – A Corpus of Parsable Sentences from the Web. In Gurevych, I., Biemann, C., and Zesch, T., editors, *Language Processing and Knowledge in the Web*, volume 8105 of *Lecture Notes in Computer Science*, pages 61–68. Springer-Verlag Berlin Heidelberg, Germany.

[3]Cf. Section 2.1.4 for a short description of the parser.

- Eckart, K., Riester, A., and Schweitzer, K. (2012). A Discourse Information Radio News Database for Linguistic Analysis. In Chiarcos, C., Nordhoff, S., and Hellmann, S., editors, *Linked Data in Linguistics. Representing and Connecting Language Data and Language Metadata*, pages 65–75. Springer-Verlag Berlin Heidelberg, Germany.

1.2 Concepts

This section describes some concepts on which this work is based, and which are relevant to several aspects thereof. While the following subsections can be used as a reference and each can be read on its own, they are ordered in a way which develops an illustration of the background.

1.2.1 A broad definition of linguistic resources

Throughout this work a broad interpretation of the notion of **linguistic resource** will be employed, covering not only corpora and linguistic knowledge bases, but also all other components which play a role in a computational linguistic workflow, or are generated as a result thereof, e.g. processing tools, single annotation layers or language models.

Figure 1.3 shows a set of components in a typical computational linguistic setting, each of which can be considered as a resource of its own. There is, for example, linguistic **primary data**, which is data related to linguistic utterances and which does not include an explicit representation of a linguistic interpretation. Primary data can consist of text, images, audio or video signals, etc.; refer to different modalities, e.g. spoken, written, signed, etc.; and might originate in the form of collected corpora, fieldwork material, experimental data, survey data, etc.[4]

Often primary data serves as an input for some kind of processing (chain), in which a linguistic interpretation is made explicit by adding (multiple layers of)

[4]Figure 1.3 visualizes primary data from the DIRNDL corpus, cf. Sections 1.2.2 and 6.2.1. The waveform is a screenshot from Wavesurfer (`http://www.speech.kth.se/wavesurfer/`).

annotations.[5] These annotations can be created manually or can be generated by automatic **processing tools**, which may come in the form of downloadable systems or encoded as web services. Most of these processing tools incorporate **knowledge bases**, such as grammars, lexicons, word nets, lemma lists, frequency lists, language models, etc. These knowledge bases can be based on linguistic theories as well as on statistical information extracted from other resources, e.g. another annotated corpus.

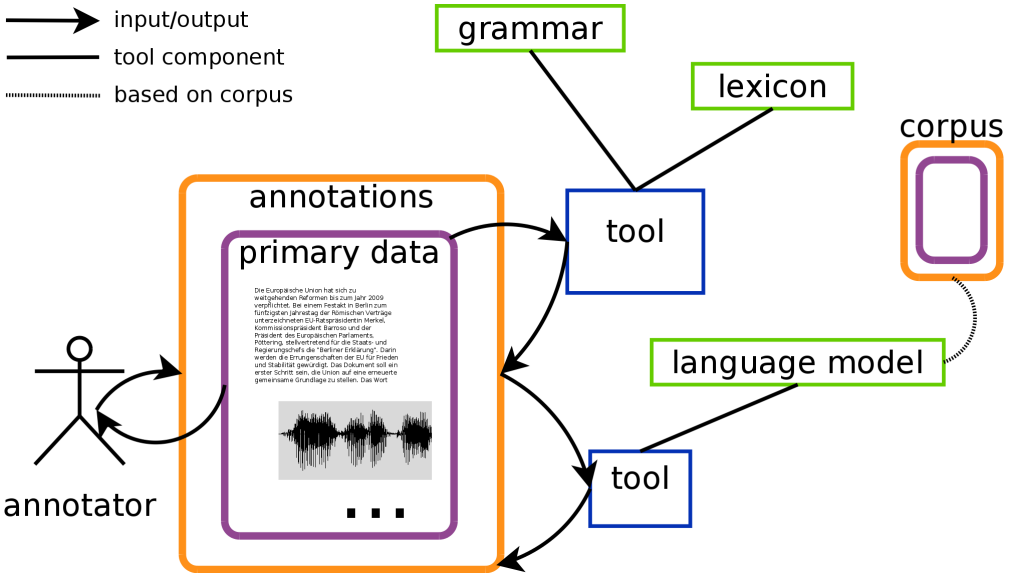


Figure 1.3: Some components of a computational linguistic setting.

As stated above, we use the term resource to cover all these different components. An annotated corpus, a specific tool, or a knowledge base can be part of a computational linguistic workflow but can also be the result of such a workflow, and sometimes even both, cf. the discussion on bootstrapping workflows in Section 1.2.4. Additionally, a processing tool might not only generate annotations, but also conduct other processing steps, e.g. extract relevant data or produce a conversion from one data format into another.

Of course there can be the need to subclassify resources into categories like tools, corpora and lexical resources or along other dimensions. Witt et al. (2009)

[5] Cf. (McEnery and Wilson 2001: pp. 32f.) and Leech (1993). They state that one has to be aware that each annotation act is actually an interpretation of the structure or the content of the data. Thus it is to be noted that an annotation need not be consensual among linguists.

propose a taxonomy of language resources based on two independent features: they distinguish (i) between **static resources** and **dynamic resources** and (ii) between **text-based resources** and **item-based resources**. In their taxonomy, they describe static resources as data inventories, such as corpora and ontologies, and dynamic resources as generators of new data, such as parsers and data extraction tools. Their second feature depends on the size of the base unit of linguistic objects under consideration. Item-based resources focus on distinct, individual objects, such as a lexicon, while text-based resources relate to a combination of base units, like corpora or tools for statistical machine translation which take phrases or sentences into account.[6]

Besides the classification proposed by Witt et al. (2009), there are others as well, cf. also Section 3.1 for classifications based on aspects of interoperability. However, different classifications arrange resources according to orthogonal criteria and thus might even leave out single resources which either fit into multiple categories or into no categories at all. One theme of this work is to find strategies for a **generic** treatment of different resources, e.g. in Chapter 4 all data from a linguistic workflow is treated in the same way in a relational database, and in Chapter 5, syntactic annotations from different parsers can be inserted into the combination workflow. Accordingly, this work employs the broad definition of the concept of a resource.

1.2.2 Distinction of data layers

While this work makes use of a broad concept of resources themselves, we will nevertheless make use of a strict classification when it comes to data which is associated with a resource. We will make a clear distinction between data which is part of the resource, e.g., primary data, annotations, or source code, and

[6] In this distinction, the second feature depends of course on the definition of the base unit. Since in their examples, Witt et al. (2009) describe (part-of-speech) taggers and morphology systems as dynamic item-based resources and parsers as dynamic text-based resources, the base unit is on the lexical level. Using another level of base units, e.g. a sublexical level, can thus change the attribution of the resources.

data which describes a resource, e.g., **metadata**, or other instances of separate documentation[7].

Figure 1.4 shows an example of this distinction, based on an annotated corpus. As already stated above in Section 1.2.1, primary data is resource data where no linguistic interpretation is made explicit. All explicit linguistic information added to the primary data is seen as annotation, which is then (an important) part of the resource itself. Data comprising facts *about* the resource, however, is not seen as part of the resource but as separate documentation, including metadata, user manuals, and published papers dealing with the resource.

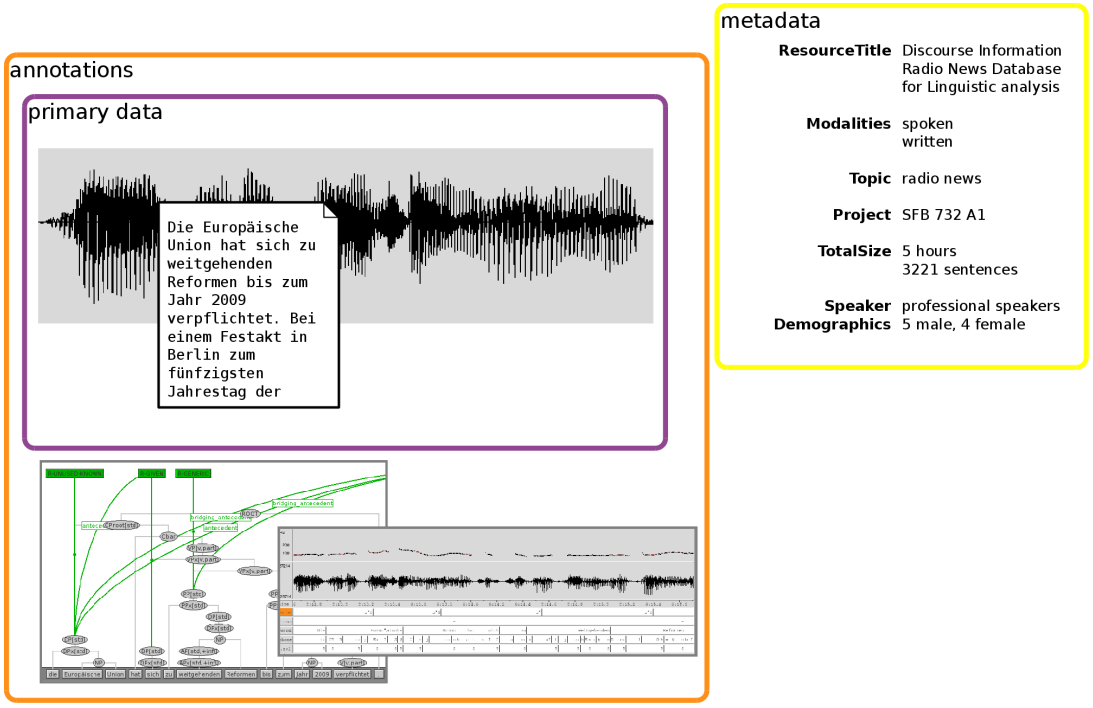


Figure 1.4: Data layers related to the DIRNDL corpus.

The example corpus here is the *Discourse Information Radio News Database for Linguistic analysis* (short: DIRNDL, Eckart et al. 2012; Björkelund et al. 2014b),

[7]This refers to the distinction between integrated and separate documentation as adopted from software engineering: integrated documentation is equivalent to comments within the source code, separate documentation is that part of the software which is not included into the source code (cf. Ludewig and Lichter 2013: p. 259). In our case the source code is equivalent with the resource data.

which is described in more detail in Section 6.2.1. For now we will only take into account that DIRNDL consists of two sets of primary data, i.e. audio recordings of German radio news broadcasts and textual manuscripts thereof, and several layers of annotation, based on these primary data sets.[8]

The metadata box in Figure 1.4 contains information about DIRNDL encoded in a formal structure of feature value pairs. In this sense, metadata is a structured type (and often part) of resource documentation. Metadata can be automatically interpretable by means of e.g. a document grammar, while documentation as such can consist of unstructured data as well, e.g. in user documentation such as manuals.

Since metadata and in particular the type of metadata which we call **process metadata** plays an important role for the infrastructural aspect of our combination approach, we go into more detail on these concepts in Sections 1.2.3 to 1.2.5.

1.2.3 Metadata

As described in Section 1.2.2 metadata are structured information about the resource, but are not part of the resource itself. Keeping this accurate distinction is often an advantage, since there are linguistic resources for which the resource data are not or cannot be made freely available, e.g. due to the rights of the authors of the primary data, or the protection of subjects in a linguistic study. However, metadata which do not contain parts of these resource data, but only information about the resource, can be made publicly available and can thus foster collaborations between researchers interested in the same topic.

[8]Figure 1.4 shows syntactic and semantic annotation layers in a screenshot from SALTO (Burchardt et al. 2006), PID: `http://hdl.handle.net/11858/00-246C-0000-0005-BD14-0`, as well as phonetic and prosodic annotation layers in a screenshot from Wavesurfer, `http://www.speech.kth.se/wavesurfer/`. The waveform in the primary data box is also a screenshot from Wavesurfer. This author has arranged the figure and has used it in several presentations. Due to its display of characteristic settings, it has also been provided (in its original color scheme) to be used in work regarding public relations of the *Institut für Maschinelle Sprachverarbeitung* of the University of Stuttgart and can be found, e.g. in the *FORSCHUNG LEBEN* magazine (08.2017) of the University of Stuttgart.

This is for example done by adding metadata sets to online databases or catalogues, e.g. the LRE Map[9] or the CLARIN Virtual Language Observatory[10]. There, the metadata elements constitute search criteria, on the basis of which interested researchers can identify resources which might fit their needs. For example, when searching for a speech corpus of professional speakers, the metadata set displayed for the DIRNDL corpus in Figure 1.4 identifies DIRNDL as a suitable corpus. Of course such shallow searches might not point out the most fitting resource for a task at hand, but they are able to filter for a certain set of possibly fitting resources; on the basis of the full metadata set, potential users can then assess the usefulness of the resource with respect to their specific task.

Based on this small use case, we can identify two other criteria which are important for the sensible use of metadata. First, they have to come in a format which is on the one hand interpretable by automatic systems, such as being automatically searchable, and which is on the other hand human readable, such that the detailed information which the users might not have searched for directly, is nevertheless accessible to them. Second, they should take requirements of different user groups into account: a diachronic corpus consisting of historical documents might be of interest for linguists inspecting language change as well as for historians extracting descriptions of specific events. However, these two groups might use different search criteria, e.g. historians could be interested in the name and profession of the author, while linguists might rather want to know if the dataset is large enough to find a sufficient number of instances of certain types of words. Nevertheless they might both be interested in the time coverage of the data, i.e. the period of time in which the primary data was created.

Over time, several initiatives have proposed formats for a structured representation of such metadata information. Thereby the focus can be on a specific type of resource or on a broad range of resources, and the power of description ranges from small and restricted sets of metadata elements, which are easy to

[9] `http://www.resourcebook.eu`
[10] `https://catalog.clarin.eu/vlo/`

query, to large and flexible ones, which are harder to query but which also take domain-specific details into account.

Examples of such initiatives and formats are the **Dublin Core** elements from the *Dublin Core Metadata Initiative* (DCMI)[11], the format from the *Open Language Archives Community* (OLAC)[12], which emerged from DCMI, the header of the XML format specified by the *Text Encoding Initiative* (TEI)[13] and the *Component Metadata Initiative* (CMDI)[14], which originates from the CLARIN Research Infrastructure[15].

However, most metadata schemes focus on facts about the static resource. Metadata on the creation process of the resource can add important information for a potential user, thus Section 1.2.4 describes process metadata in more detail.

1.2.4 Process metadata for workflow tracking

As described in Section 1.2.3, the creation and publication of metadata fosters the reuse of data, which is an important factor in resource sustainability. The advantages for the research community are immediately evident: creating a resource is costly, and by reusing and further developing already existing resources, new users do not have to invest the creation costs again, while the costs of the creators are honoured by means of citations.

However, next to the information about the static resource encoded in the metadata schemes as described above (cf. Section 1.2.3), it can be highly relevant to also encode information about the creation process of a resource or the workflow of a study. This is the case, because even a small detail somewhere in the workflow can have an enormous effect on the result, and is thus a crucial information for the reuse of the data.

As an example of this, we refer to a study by Elming et al. (2013), which investigates the influence of four different tree-to-dependency conversions on several tasks such as negation resolution, statistical machine translation and

[11] `http://dublincore.org/`
[12] `http://www.language-archives.org/`
[13] `http://www.tei-c.org/`
[14] `https://www.clarin.eu/content/component-metadata`
[15] `https://www.clarin.eu/`

sentence compression amongst others. For these tasks, which are often called downstream tasks with respect to the parsing step, the conversion is only a small detail in the respective workflows. With all tree-to-dependency conversions, a constituency-based syntactic analysis as in Figure 1.1a is converted into a dependency structure as in Figure 1.1b, but each conversion scheme is based on some different linguistic considerations. While the conversions are only a minor step in the workflow, the overall results of the tasks vary with the decision on the converter.[16] Thus, as a first step it is important to inform the users that such a conversion happened and furthermore which conversion scheme was applied.

Another example could be the creation of a corpus resource: at some point, the creators might decide to delete sentences, e.g. because they are too short or too long for their processing tools, or because they are not relevant for their task, etc. However, this decision might influence future studies taking the frequency of specific phenomena into account, and it should thus be included in the resource documentation.

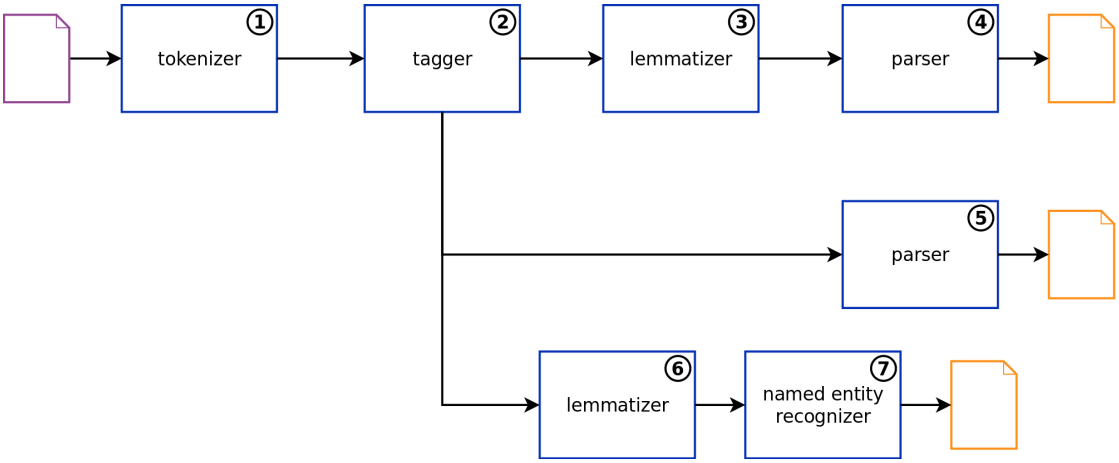
Of course, creators should not be restricted in creating a resource which fits their needs, thus a thorough documentation still supports the sensible reuse of the resource: New users should receive as much information as possible about the decisions made in the process of creating a resource, such that they can make an informed decision whether the resource fits their needs.

The same is true with respect to reproducibility and comparability of studies conducted with the help of linguistic resources: Knowing exactly how each resource or result was created helps researchers to assess if the outcomes can be compared to each other, as seen in the example of the downstream tasks above.

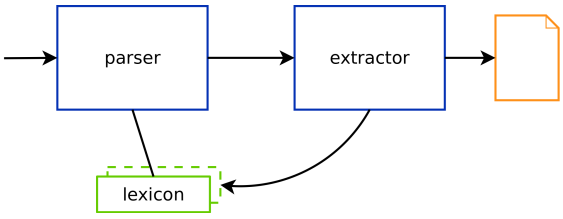
Keeping track of these creation or processing steps can be done by collecting process metadata, comprising information on applied resources and their precise version, and on the relations and dependencies between steps of a workflow, cf. Section 1.2.5.

[16]The evaluation of the statistical machine translation with BLEU (Papineni et al. 2002) did not show significant differences, but the actual translations were reported to be very different nevertheless.

Three typical workflow types are linear workflows, also called **pipeline**, **branching** workflows and **bootstrapping**. Figure 1.5 shows examples for each of them.[17]



(a) Pipeline and branching



(b) Bootstrapping

Figure 1.5: Workflow types.

A pipeline consists of several workflow steps which are processed one after the other. The output of one step is the input of the next step and each step adds to the overall result. The sequence of steps ①, ②, ③, ④ is a typical pipeline in natural language processing, comprising segmentation of input text (tokenizer), annotation of part-of-speech tags to the segments (tagger), annotation of a dictionary form to the segments (lemmatizer) and creation of a syntactic analysis (parser).

Branching happens whenever alternative steps are inserted such that two pipelines can be executed independently of one another. This can happen in

[17] Cf. also Ulusoy (2014).

cases where the results of both branches are to be compared, but also, when one branch is neglected at some point and a new branch is created, starting from a previous step. In Figure 1.5a step ⑤ introduces an additional parser, which, as opposed to the parser in step ④, does not use any lemma information. The output of steps ④ and ⑤ might be compared afterwards. If it is found in the workflow that the syntactic information from neither parser is helpful, these branches might be neglected and another branch may be started with a different lemmatizer ⑥ and a tool which annotates specific entities such as names of persons, cities or institutions (named entity recognizer, step ⑦). Of course a branch can contain several steps, be branched again or merged back into one of the other branches.

The third workflow type, bootstrapping, describes an iterative process, where the outcome of one step provides additional knowledge for the reapplication of the same step. When to stop the bootstrapping process can for example be determined by the quality of the output. Figure 1.5b shows a part of a workflow, where a parser uses an additional lexicon as knowledge base (cf. Section 1.2.1). After the parsing step, new information from the output is extracted and added to the lexicon. That way, the knowledge base of the parser grows, and when the parser is applied again on the same input, the output might change due to the new information in the lexicon. For a respective setting see Eberle et al. (2008).

However, to be able to include all relevant process metadata into the documentation of a resource means to track it already in the course of the work. Going back to the similarity to software documentation, at which we already hinted in Section 1.2.2, Ludewig and Lichter (2013: pp. 259, 267) state that development and documentation are inseparable: If the decisions are not recorded during development, they are likely to not be accessible later, e.g. because they are simply forgotten, or because the respective person has left the development along with the information. Thus anyone who has to change the software later (in our case: work with or further develop the resource later), has the task of an archaeologist or is bound to speculations which tend to be wrong.

Nevertheless, collecting process metadata along the way supports the workflow itself, since it helps to keep track of complicated processing steps, allows for backtracking to every point of the already executed workflow, e.g. for branch-

ing, and makes it easier to assess the quality of the available results and their interpretation. We will track and make use of process metadata based on the database infrastructure, cf. Chapter 4, applied in our combination workflow. This database is also able to capture process metadata for all analysis relations presented in Section 1.2.5.

1.2.5 Analysis relations

Linguistic annotations are usually created in an analysis process consisting of several steps, cf. Section 1.2.4 for examples of respective workflow types. Eberle et al. (2012) describe three dimensions of analysis relations, which can be illustrated for prototypical corpus annotation workflows in Figure 1.5. Anticipating the discussion on interoperability in Chapter 3 the following paragraphs will also shortly state some connections between interoperability issues and these three dimensions of analysis relations.

Vertical analysis relations. When analysing language data, the analysis steps often reflect a multi-layered structure of language. For textual data the usual (automatic) processing steps include segmentation into sentences and tokens (Figure 1.5a, ①), annotation of part-of-speech tags and lemmas (Figure 1.5a, ②+③), generation of syntactic trees representing the structure of each sentence (Figure 1.5a, ④) and maybe some further annotation produced e.g. by named entity recognition, coreference resolution, etc. Thus, there are relations which exist between so-called 'higher' and 'lower' annotation layers, based on the idea of abstraction from surface form to abstract representations: a 'higher' annotation layer usually depends on the information from a 'lower' level. Due to the view of 'higher' and 'lower' these relations can be seen as vertical analysis relations.[18]

[18]The denomination of the analysis relations is based on the idea of 'higher' and 'lower' annotation levels and is independent of the visualization. The visualization of the workflows in Figure 1.5 illustrates this: it contains vertically related analyses which are the outcome of the workflow steps which are displayed horizontally. Since workflows from real settings can become large graphs, the orientation of a fitting workflow visualization varies between several possibilities, e.g. left-to-right, top-to-bottom, bottom-to-top or centre-outwards.

The content of a 'lower' level output, e.g. annotation guidelines and tagset, needs to fit the requirements of the 'higher' level. In an automatic processing chain, these content-related interoperability requirements come with additional ones in terms of representational interoperability (cf. the discussion on categories of interoperability in Section 3.2): a parser which expects tabular information from segmentation and part-of-speech tagging will not be able to handle XML input, even if the tagset is interpretable by the parser.

Horizontal analysis relations. For most analysis layers, there are several competing or complementing proposals in the literature and in implemented tools how to annotate them, starting from different decisions on what a token is, up to the distinction between phrase-structure trees and dependency graphs (cf. Figure 1.1). Horizontal analysis relations thus exist between alternative analyses from the same linguistic description layer for the same input, e.g. between the outputs of the two different parsers in Figure 1.5a, ④ and ⑤.

Since there are many approaches to combine information from different annotations on the same analysis layer, cf. Section 2.2, interoperability is an important factor for the combination of these annotations.

Temporal analysis relations. Annotation schemes, annotation tools and knowledge bases may evolve over time. This is catered for by temporal analysis relations. If the same input is annotated by two different instances of a given analysis process, e.g. two versions of a tagger, or with different lexicon versions such as in the bootstrapping in Figure 1.5b, the resulting analyses might also differ and are thus subject to a temporal relation. In an optimal setting, the quality of the output increases over time, however this is not always the case and might lead to changes in the development of a tool or to a halt in a bootstrapping workflow.

This relation type is important for the constant development and enhancement of linguistic resources, but is usually rather uncritical with respect to interoperability, since the outputs of these steps tend to be similar and often do not interact.

Similar relations exist of course between the resources which are created with vertical, horizontal and temporal analysis steps. An audio corpus which has been processed by two different systems for the same level of annotation thus yields two resources which are related by a horizontal analysis relation. Documentation on the creation process of a resource therefore often helps to assess if two resources can be applied together. Information about vertical, horizontal and temporal analysis relations can be captured by means of process metadata (cf. Section 1.2.4), stating which input has been processed by which tools and in which version of the tools.

1.2.6 Reliability of annotated linguistic information

In this section we shift the focus from monitoring what has been done with the data (process metadata, Section 1.2.4) to what can further be done with it based on the quality of the annotation. **Reliability** of extracted information is an important concept for this work and in this context it is to be understood from the users' perspective: the higher the reliability of the parser output, the more the user can rely on the correctness of the syntactic information passed on to further processing steps.

However, with the task-based focus of this work, its approach to reliability does not intend to improve complete sets of annotation, but to increase the benefit the annotated information can bring to subsequent processing. Thus, we do not talk about reliability of the whole output, but about reliability of those parts of the output which are relevant for the task.

While reliability is the concept which characterizes output from the users' perspective, the tool perspective is captured by the related but distinct concept of **confidence**. A parser is usually in the situation that it can create several analyses for the same sentence. Thus, to decide on an analysis it uses indicators such as probability values to decide on the next analysis step, or to chose one analysis from a list of possible ones. These lists are often n-best list, where the analyses are ranked based on the confidence of the tool with respect to each analysis. The more confident the tool is regarding an analysis, the higher this

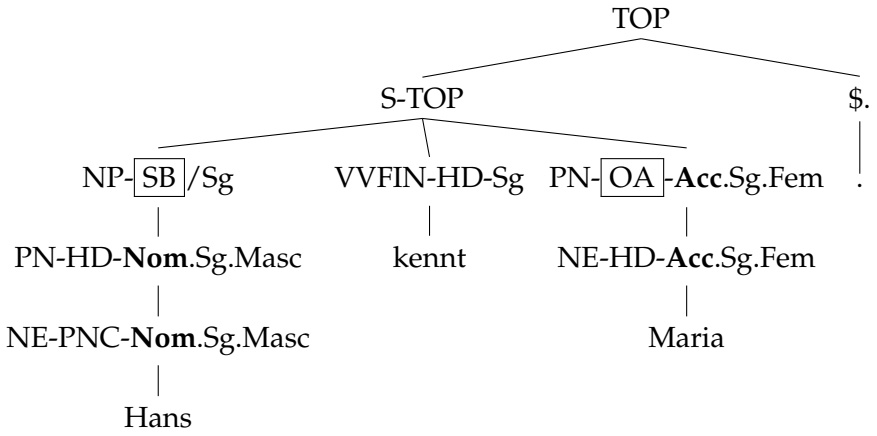
analysis is ranked in the list. The analysis on rank one is then returned as the output of the parser.

(1.2) *Hans kennt Maria.*
Hans knows Maria

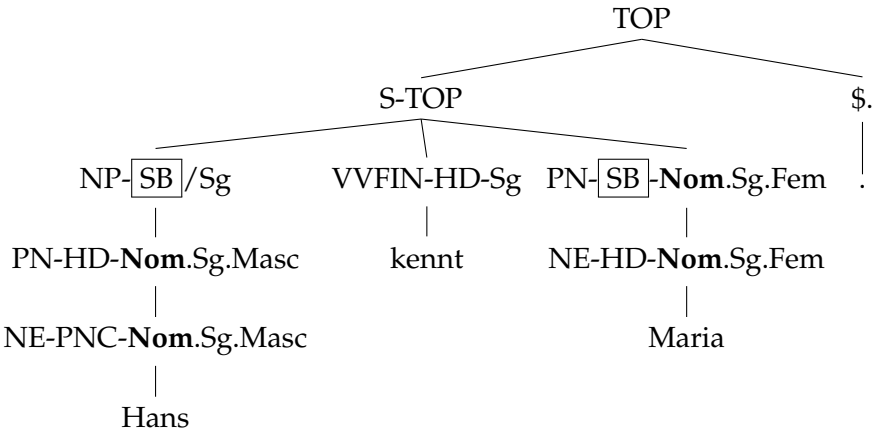
To compare reliability and confidence, Figure 1.6 shows three analyses from the parser BitPar for the German sentence in Example (1.2).[19] This sentence has two readings, (i) *Hans* is the subject of the sentence and in nominative case, while *Maria* is the object of the sentence and in accusative case and (ii) *Maria* is the subject of the sentence, in nominative case, and *Hans* is the object, in accusative case. Without context, the standard reading is (i), which corresponds to the analysis BitPar ranked first (Figure 1.6a). It is in fact ranked two places higher than the analysis shown in Figure 1.6b, which is incorrect because in this analysis both, *Hans* and *Maria*, are analysed as nominative case. So far the confidence of the tool supports the reliability of the decision. However, the first analysis which displays the correct reading alternative (ii), is ranked on position eleven (Figure 1.6c), i.e. seven ranks lower than the incorrect analysis from rank three. The higher confidence of the wrong analysis does thus not support the reliability of the output in this case.

So far, in this example we took the single-tool perspective into account. Section 6.2 will show, among other things, how information from several analyses of an n-best list can be combined with output from other parsers to increase the reliability of the output parts relevant for the task. Section 1.2.7 introduces the denomination of exactly those parts.

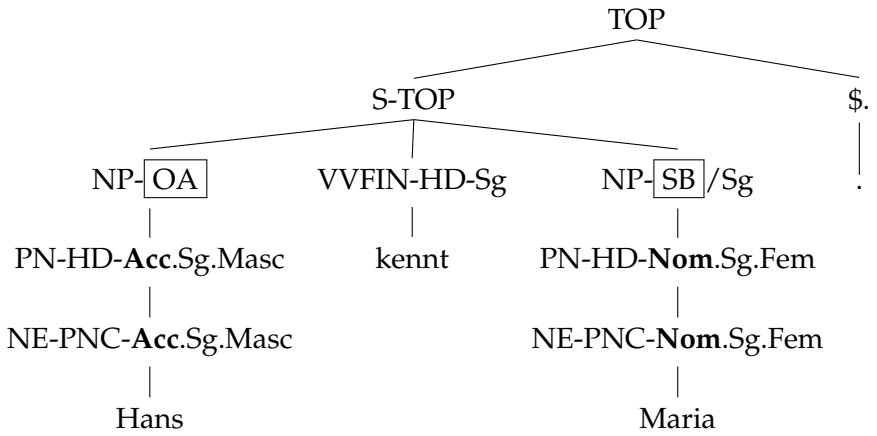
[19] The highlighted parts of the annotation are: SB – subject, OA – object in accusative case, Nom – nominative case, Acc – accusative case.



(a) Rank 1



(b) Rank 3



(c) Rank 11

Figure 1.6: Parse trees from n-best list.

1.2.7 Syntactic features

In this work, the parts of information which are passed on from the parsing output, i.e. the syntactic analysis, to the further processing steps of the task are called **syntactic features**. Section 1.2.6 already mentioned that it is the reliability of exactly these features which this work wants to increase.

The term 'feature' is however widely used in machine learning settings, which also applies to parsers based on machine learning approaches. Therefore 'feature' or 'syntactic feature' often appear in the literature regarding respective parsers (e.g. Bohnet 2010) or their combination (e.g. McDonald and Nivre 2011). In this work, the term is used in the sense of the above definition and does thus neither restrict the approach to a specific type of parser or combination nor does it necessarily imply a machine learning setting as part of the task. Nevertheless, the use of features in this work is similar to their use in machine learning in the respect that they are extracted as relevant parts to inform a processing step.

1.2.8 Precision and recall

This last section refers to evaluation metrics and differs from the other described concepts in the respect that there is of course a formal definition for these metrics available. It is added here however, since the concepts of precision and recall are referred to in several places throughout this work.

For the evaluation of parsing results, precision, recall and their harmonic mean (called F-score, F_1 score or F-measure) are typical metrics. They originate from the field of information retrieval and include four possible cases for each item in the result set: (1) true positives (TP), i.e. correct results; (2) false positives (FP), i.e. elements which are part of the result set, but should not be; (3) true negatives (TN), i.e. elements which are correctly not part of the result set; and (4) false negatives (FN), i.e. elements which should be part of the result set, but are not. Precision (P), cf. Equation (1.3a), is then the relation between the true positives and all positives, i.e. it describes how many of the elements in the result set are correct. Recall (R), cf. Equation (1.3b), is the relation between the

true positives and the set of true positives and false negatives, i.e. it describes how many of the correct elements have been found.

$$P = \frac{TP}{TP + FP} \tag{1.3a}$$

$$R = \frac{TP}{TP + FN} \tag{1.3b}$$

For a specific task, it might be more important to ensure that the retrieved features from the data set are correct than finding all available features, cf. Section 6.1; for another task it might be more important to find as many of the features as possible, at the expense of having some false positives included, cf. Section 6.2. In cases where there is no clear preference for recall or precision, the harmonic mean of the two values can be computed, evaluating how close a result comes to the impossible case of mutual optimization. Depending on the features to be extracted or the task they are applied for, there can be other measures involved, such as accuracy, or attachment scores utilized to evaluate dependency relations.

1.3 Motivation of task-based parser output combination as a device to reach the objective

The introduction of this chapter has set the background and presented the objective of finding an adequate syntactic analysis for further processing tasks. This section will now shift the focus and motivate why exactly task-based parser output combination is proposed as this work's device to reach the objective.

Therefore we again take up the two assumptions this work is based on:

i Generating syntactic analyses of language data is not an isolated task.

ii Parsing is (or: cannot be) perfect.

We discussed in the introduction that (ii) is due to the many context factors which influence the notion of perfection. However, if the result of parsing cannot be optimized in a global sense, can we at least increase its reliability for

further automatic or manual processing steps, since these steps typically rely heavily on the quality of the parsing output?

Literature shows that a combination of systems often yields better results than the best single system. This can be found to hold across different tasks, e.g. automatic speech recognition (Fiscus 1997), part-of-speech tagging (Tapanainen and Voutilainen 1994) and parsing (Henderson and Brill 1999); for different structures, e.g. constituents (Henderson and Brill 1999) and dependency (Zeman and Žabokrtský 2005); and also for different approaches, from voting over classifiers choosing from different parser outputs (Surdeanu and Manning 2010) to extending a parser to become a combination classifier itself (McDonald and Nivre 2011).[20]

So parser combination is a safe point to start from. However there are two aspects of parser combination to look at in more detail. First, the combination gain is based on the fact that systems come with different error distributions. Combining systems which make the same errors will propagate these errors through the combination step. Fortunately, different parsing systems usually show different error distributions and the more different the systems are, the more different are the error distributions expected to be. This is however limited by the fact that a combination can only take results into account which it can actually compare. That is, the systems need a certain degree of interoperability to take part in a combination approach.

The second aspect is that parser combination techniques so far were aimed at finding the best overall result, i.e. the structure which maximizes the quality for the whole input sentence. However the relevant syntactic information depends on the task, and thus the approach of maximizing the quality of the overall parse might go at the expense of the quality of exactly the syntactic information needed for a specific task. For a little excursus on different task-relevant information let us inspect this fact in conjunction with some example tasks, gathered from different projects of the collaborative research centre described in Section 1.1.

One task was the disambiguation of German nominalizations ending in *-ung*. These nominalizations can appear with up to three readings: an object reading,

[20] Cf. Section 2.2 for more details on this related work.

an event reading and a state reading. Indicators to decide on the reading in a specific sentence can be extracted from the context in the form of modifiers and selectional preferences (Spranger and Heid 2007; Eberle et al. 2009); thus, they can be extracted from a syntactic analysis (Kountz et al. 2007).

In a corpus study on passives of reflexives, Zarrieß et al. (2013) extracted candidates based on the syntactic analysis showing reflexive pronouns with their heads being passivized verbs. And in a study on German *nach*-particle verbs, Haselbach et al. (2012a) had to identify such verbs also in cases where the particle is separated from the verb. Furthermore their study is based on dative and accusative arguments appearing with these verbs.

To give another example, the process of annotating information status according to the RefLex scheme (Baumann and Riester 2012) requires detailed information on phrase embedding, because in the hierarchical scheme subphrases can be annotated with labels which are different from the phrase in which they are embedded.

What we easily see here, is that relevant syntactic information for one task might be irrelevant for another one. Thus the basic principle of this work is to take the task into account when increasing the reliability of the syntactic information. While this accounts for the fact that different tasks rely on different parts of the analysis, it also provides a broader spectrum for the aspect of interoperability mentioned above: instead of full interoperability of the systems, the combination only needs to be able to compare the parts which are relevant for the task, which in turn even allows for the combination of more different systems.

A more detailed description of the *nach*-particle verb and information status tasks will be given in Section 6.1 and Section 6.2, respectively, since these two tasks have been included as case studies for this work.

So far we motivated the use of task-based parser combination. The last part to add is the focus on the output. Combination of the output excludes the possibility of going into a system to change it, e.g. as in McDonald and Nivre (2011) where a parser becomes a combination classifier. However this work introduces this restriction to promote an architectural concept which comes along with the concentration on the task: with this restriction the parser can be handled as a

plug-in module in the workflow of the overall task. As a consequence, different parsers can easily be inserted into and can be switched within the workflow, and the main contribution of this work, the abstract workflow for task-based parser output combination, can be applied independently of the availability of open source systems or the user's expertise regarding the specificities of parsing systems. It is thus applicable in a broader range of settings.

The following chapters will present the aspects of task-based parser output combination in detail.

Chapter 2

Technical background: Combining syntactic information

In this work, the output of several different parsing systems takes part in a combination approach. The result of the combination provides information for tasks such as corpus analytical studies. To understand the differences between the parsing systems, Section 2.1 gives an introduction to the notion of parsing as applied in this work; the categories along which the different systems are classified; and the systems themselves. These differences between the parsing systems are also used later in this work to categorize the combination approaches, and we will show in which way they are central to the performance of combined systems.

Section 2.2 discusses related work on parser combination and compares the approaches with respect to their basic structures to be combined and their actual combination method. Section 2.3 adds some related work on task-based parsing.

This chapter thus describes the technical background regarding the parsing approaches applied in this work, as well as related work on parser combination and task-based parsing.

2.1 Parsing: automatic syntactic analysis

Chapter 1 gave a first definition of the term **parsing** within this work where it is understood as automatically generating a syntactic analysis of a natural language expression, stated similarly in Kübler et al. (2009: p. 1). This functional view defines an input and an output of the parsing process. Thereby the input expression is usually a sentence of written text and the output is a formal representation of a complete analysis of the input expression, in many cases a hierarchical syntactic structure which spans every part of the input sentence. A **parser** is then an executable implementation of the process of syntactic analysis.

Taking a closer look at the specification of the input expression, its scope is not clearly defined with respect to natural language. While it will often be a sentence, there are also many cases where the input is not a sentence, yet it makes sense to provide a syntactic analysis. Examples are headlines, transcriptions of spoken utterances or user generated content on the web. Thus we do not restrict the input to sentences here, to provide for a realistic setting. Even though non-sentence input expressions might affect the performance of individual parsing systems, we assume the overall combination procedures in the following to be applicable to input expressions in the above sense.

Moreover, we also base our discussion on the assumption that the input expression is a completely determined sequence of discrete symbols (cf. Langer 2004) called **tokens**. For parsing, the tokens are usually similar to words; we use an intuitive notion of "word" here, and do not attempt to provide a theoretically motivated definition. While we assume this sequence of input symbols to be determined in advance, works like Seeker and Çetinoğlu (2015) propose combined approaches for input segmentation and parsing, building syntactic structures over a word hypothesis graph.[1]

In our approach the parsing always works on a static token layer, however this does not mean that the tokens have to be identical for every parser in the combination. Section 2.1.1 discusses the preprocessing of the input prior to the actual parsing step in more detail.

[1] This is particularly relevant in languages and settings in which the syntactic analysis builds on units which are not identical to the surface "words" separated by whitespace.

Having discussed the input expression, we shift our focus to the output. Depending on the theory and framework on which the parser is based, the result of the analysis process contains different features. As a hierarchical structure (prototypically a tree), the generated syntactic analysis contains information on relations between the input symbols (e.g. belonging to the same phrase or being the dependent of a specific head). When classifications are applied on top, or instead of the structure, they are usually marked by labels in the analysis, denoting the roles and functions of the symbols in the overall expression.

As stated above, we define the output of parsing to be a complete analysis, including every symbol of the input expression. Besides full parsing, there is also the process of chunking, which provides partial syntactic structures for connected components, but not necessarily for the complete input expression. Chunking systems have not been included in the approach of this work.

In cases where it is not possible to find a sensible and theoretically motivated analysis for the complete input, the requirement of a complete analysis usually leaves the systems the option to include unattached symbols at a generic place in the structure or to not return any structure at all. The former device is also often applied with respect to punctuation symbols.

Figure 2.1 sums up our introduction so far. We presented parsing as an automatic process, and we highlighted its input and output structures, i.e. a sequence of discrete symbols and a hierarchical structure respectively.

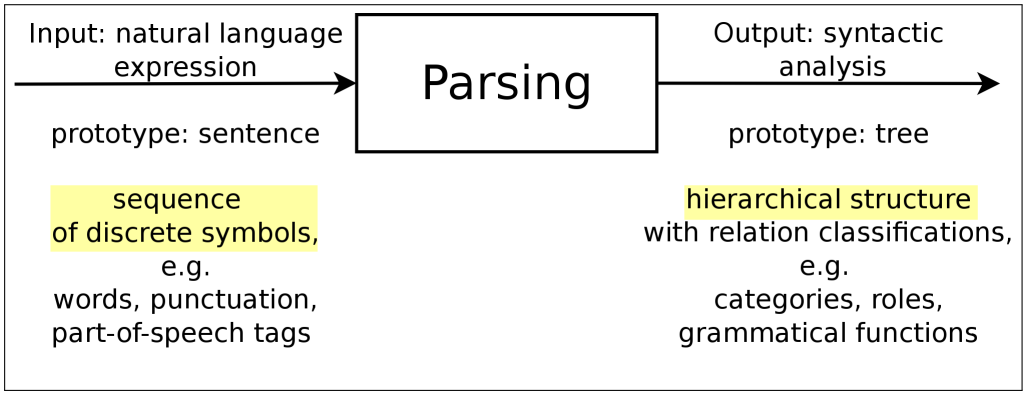


Figure 2.1: Parsing as a function: characteristics of input and output.

Two more aspects were added to Figure 2.1, one regarding the symbols in the input sequence and the other regarding the prototypical structure of the output:

Although the tokens, i.e. words and punctuation symbols, are usually part of the resulting syntactic analysis, parsers sometimes make use of word-level labels as input symbols, e.g. part-of-speech tags. In most cases, this leads to a substantial reduction of the size of the alphabet of input symbols the parser has to deal with, and allows for a joint handling of tokens which belong to the same class, e.g. being an adjective.

The other addition concerns the prototypical syntactic analysis: it has the formal structure of a tree. Like the input expression which is not always equivalent to a sentence, also the result of the syntactic analysis process is not always rendered a tree. See Section 2.1.2 for a discussion on representation models for the parsing output.

One more thing which becomes evident in Figure 2.1 is that by taking the functional view so far, we introduced parsing by the characteristics of its input and output, and did not say much about the parsing process as such. This is in line with the objective of this work not to make any changes within existing parsers, but to explore the possibilities of improvement based on the output. Nevertheless, in the following, we take a closer look at the objectives and frameworks of parsing, to be able to introduce in Section 2.1.3 the categories for the parsers in the combination studies of this work. We will however not go into details of parsing which go beyond this need to classify the parsers utilized in the studies.

Figure 2.2 presents a view on the concepts and objectives of a syntactic analysis. On the one hand, there is the linguistic setting, in which a syntactic analysis is based on a formalization of the allowed relations between the input symbols, e.g. a grammar containing rules to derive all well-formed syntactic structures. On the other hand, there is the algorithmic realization leading to a practical computational tool, which can actually generate a syntactic analysis. However, there is a connection between these two, since all parsing is based on underlying linguistic considerations, either directly, when some rules of a formalized descriptive grammar are implemented in the parser, or indirectly,

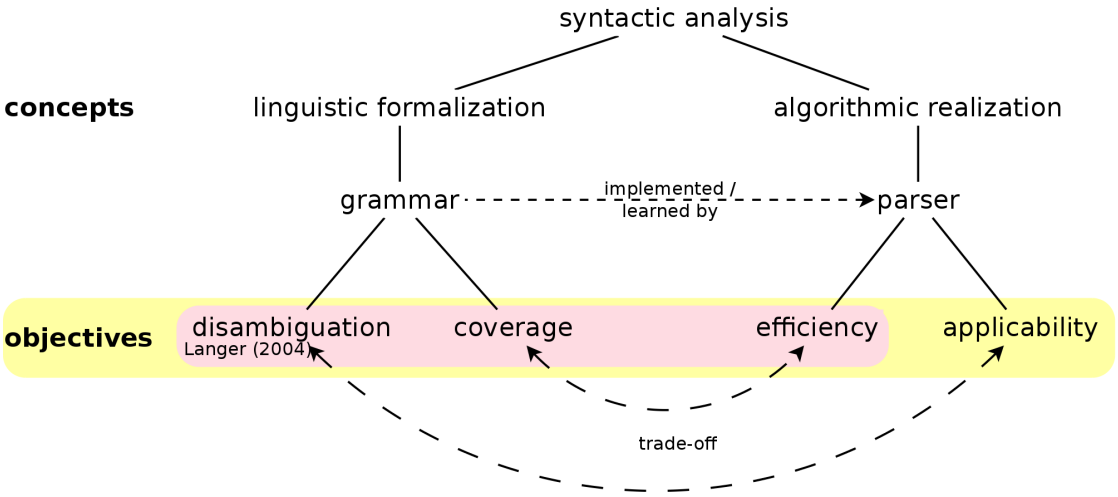


Figure 2.2: Concepts and objectives of a syntactic analysis.

when a parser is supposed to learn from a large set of available syntactic analyses, which in turn usually adhere to some theoretical considerations. There is also the case of probabilistic grammars, defining a probabilistic distribution over possible output structures which can be assigned to the input.

The next paragraphs illustrate some objectives of the syntactic analysis. According to Langer (2004), three important challenges for parsing are: (i) disambiguation, (ii) coverage and (iii) efficiency. We add applicability as a fourth one, referring to the usefulness of the generated analyses for further examination and processing.

As discussed in Chapter 1, natural language is subject to ambiguity on many levels. Formal analysis of the syntactic level thus has to deal with syntactic ambiguity. In human analysis, context and world knowledge of discourse partners usually provide the basis for functional communication[2], i.e. information from many levels is applied in disambiguation. Within formal analysis systems, however, only parts of this information are available, thus ambiguity might remain in cases where human analysis would have ruled

[2]And also humour, e.g. when ambiguities are consciously used to create a particular pragmatic effect.

out certain readings. Actually some of these readings even may not be easily evident for the human reader, cf. the discussion on the syntactically ambiguous sentence in Example (2.2) in the following paragraph.

Examples (2.1) to (2.3) show three syntactically admissible sentences. Example (2.1) is syntactically ambiguous between the reading in a, where the telescope is with the man and the reading in b, where the agent uses it as a device for seeing. The same is true for Example (2.2) but while in Example (2.1) human readers also cannot decide on the intended reading, the analysis in Example (2.2)b is usually ruled out by readers' semantic or world knowledge.[3] To bring out this point of syntactic adequateness even clearer, Example (2.3) shows a famous example[4] of a sentence which would be perfect to a parser, while rather mysterious to a human reader.

(2.1)

a. She sees [the man [with the telescope]].
b. She sees [the man] [with the telescope].

(2.2)

a. She sees [the man [with the toothbrush]].
b. She sees [the man] [with the toothbrush].

(2.3)

Colorless green ideas sleep furiously.

(Chomsky 1957)

The goal of a parser is however to exploit its available information for disambiguation where possible or at least for partial disambiguation up to the point where no knowledge is left. Example (2.4) shows a French example which is ambiguous between a reading where *ferme* is the verb of the sentence and a

[3] Note that in case no actual context is provided, there is always room for a context, i.e., if no context is given, a human reader is in many cases able to make up a context in which also a ruled out reading may apply. Think for example of a new kind of telescope resembling the form of a toothbrush ...

[4] At the time of writing this thesis, the sentence coined by Chomsky (1957) even has its own article in the community-based online lexicon Wikipedia: `https://en.wikipedia.org/wiki/Colorless_green_ideas_sleep_furiously`

reading where *porte* is. However, with the coordination in Example (2.5), only the first reading remains, i.e. for this example, syntactic information from the sentence context is sufficient to lead to disambiguation.[5]

(2.4)

a. *Le pilote ferme la porte.*
the pilot closes.VERB the door

b. *Le pilote ferme la porte.*
the pilot steadfast her carries.VERB
'The steadfast pilot carries her.'

(2.5) *Le pilote ferme la porte et la feñetre.*
the pilot closes the door and the window

The second aspect on Langer's list is coverage: As stated above, all parsing is based on grammar, either learned (data-driven parsing) or manually implemented (rule-based parsing, cf. Section 2.1.3 for details on this distinction). However, no grammar is able to cover the range of varieties of syntactically possible phenomena or their changes over time, while still ruling out all unused constructions. One approach taken in the development of parsers is to be very prescriptive in what to allow as a valid syntactic construction. This normally leads to the rejection of all those input expressions which do not comply with these rules and eventually to a loss of data which cannot be processed further. Another approach are data-driven parsers, which are usually more robust against phenomena not conforming to frequently observed patterns, but these tools therefore allow more constructions and possibly decide for non-well-formed syntactic structure.

This immediately leads to Langer's third aspect: there are efficiency criteria for parsing systems. In practice the objective is not to create a full interpreter of a specific formalism, but to find a restricted number of likely analyses in a sensible amount of time, and with a defined amount of memory capacity. This introduces a trade-off between coverage end efficiency. Thus Langer states that the parser is seen as a performance system, which restricts the

[5] Actually, the part-of-speech information is disambiguated by means of syntactic information. It is thus assumed that the parser is aware of all possible part-of-speech tags of each token.

competence system that is the grammar, including its properties of complexity and decidability.[6] In other words, in favour of producing a result at all, a parser will probably not implement a full grammar, but an efficient one.

The additional objective of applicability refers to the fact that not only during the actual parsing process there are usage-oriented criteria. Also the output has to be processable, by human inspection or for further automatic analyses. This in turn leads to another trade-off between the objectives. Even if there is not enough evidence to fully disambiguate an input expression, it might be mandatory for further applicability that the output consists of one simple, fully disambiguated analysis. Parsers thus often restrict their search space or their result structures by some sort of **forced guessing**.

2.1.1 Preprocessing

Most parsers rely on more linguistic information for their syntactic analysis than a grammar or a trained model. As discussed in the introduction to Section 2.1 the first level of information needed is segmentation information: which are the boundaries of the input segments (sentence-like) and discrete symbols (tokens). Further information can comprise lemmas, part-of-speech and morphological features. In most architectures this information has to be present in the input already before the parsing step starts, thus we subsume them as **preprocessing steps**.

There are system architectures which provide their own preprocessing utilities together with and tailored to the parser: we call these **parsing pipelines** here. Other architectures only provide the actual parser and come with a specification about which information has to be present in the input and in which representation format. Many systems are somewhere in between, meaning that they come with some own preprocessing utilities, but require e.g. an external segmentation; or they allow to override their native preprocessing in favour of an external analysis and utilize only the parsing step. However, a tailored combination of preprocessing pipeline and a specific parser can be advantageous in many cases, e.g. over independent tools which have been developed

[6] Approximate translation from (Langer 2004: p. 261).

for or trained on different domains. A statistical parser can even learn to handle preprocessing errors, i.e. build correct structures on erroneous input, when the same preprocessing is constantly applied. Lastly there are also the integrating architectures as mentioned in the introduction to Section 2.1 (cf. Langer 2004; Seeker and Çetinoğlu 2015), which generate a syntactic analysis conjointly with other information, e.g. like disambiguation preferences from the part-of-speech and parsing level for Example (2.5).

Parsing pipelines and preprocessing are based on a model of 'lower' and 'higher' annotation layers as described in Section 1.2.5. To generate an annotation layer, an analysis system can take information from lower layers into account and generate a higher layer, i.e. the higher layer depends on the information of the lower layer, but not vice versa. In a strict pipeline approach, there is no feedback from higher layers to lower layers, e.g. a syntactic analysis can depend on part-of-speech information but the part-of-speech tagging cannot take information from the syntactic analysis into account. This implies that there is an order in which the analysis steps are processed and allows errors from lower levels to propagate to the higher ones without the chance of correction. For example if the part-of-speech analysis decides for Example (2.5) that *ferme* is an adjective and *porte* is a verb, the syntactic analysis will build a respective structure to which the last sentence part cannot be correctly attached. Iterative and integrating architectures are based on a model, where information from one layer can influence (all) other layers, introducing corrections or helping in disambiguation: information from a syntactic analysis can e.g. restrict possible part-of-speech values of a token, thus leading to a reprocessing of the part-of-speech analysis.

For the case studies in this work, we apply parsing pipelines, including and requiring different amounts of preprocessing. We discuss this in more detail in Section 2.1.4 and together with the case studies in Chapter 6.

2.1.2 Output models

In this work we focus on output combination, i.e., even if we distinguish different categories of parsers (cf. Section 2.1.3) to decide which ones to combine,

the actual combination will happen on the output. Therefore it is important to know which models of parser output we might want to deal with.

The standard structure of an automatically generated syntactic analysis is a tree or **parse tree**. The formal properties of parse trees are usually those of directed rooted trees: consisting of sets of n nodes and (n-1) directed edges, a dedicated root node, such that there is a directed path either from each node to the root, or from the root to each node, and thus no cycles. One additional constraint which applies to parse trees, is that the nodes which represent the tokens are fully ordered by means of their input sequence.

A set of parse trees which represent several different syntactic analyses for the same input segment is called a **parse forest**.

However, the output model of a parser can also be a directed acyclic graph, for example when additional information is encoded into the parse tree, e.g. secondary edges denoting missing constituents in sentence coordinations (Albert et al. 2003), or in cases where ambiguity is represented by several attachment points within the same structure.

2.1.3 Categories of parsing tools and techniques

In this work, several different parsing systems have been utilized for the combination case studies. As will be shown in Section 2.2, framework-related and technology-related differences between the parsers play an important role for the success of the combination. We will thus discuss in the following some framework-related and technical categories according to which the parsing systems utilized in the case studies can be classified and which make the different setups evident.

The systems can be classified into constituency parsers and dependency parsers, and data-driven approaches can be distinguished from rule-based approaches. These two classifications are orthogonal to each other. There are preferences, which are however mainly due to historical reasons, e.g. most dependency parsers tend to be data-driven. Some additional classification criteria, e.g. the ways in which the parsers handle ambiguity, will be discussed together with the utilized parsers in Section 2.1.4.

Constituency parsers. Constituency parsers are based on the concept of phrase structure grammars. The terms **phrase** or **constituent** refer to the complex segments of the input expression introduced by the grammar (Langer 2004). They are complex insofar as their size is between that of an atomic input symbol and the size of the whole expression, and because they contain their own internal structure.

Phrase structure grammars are usually context-free grammars. In this sense, $G = \{N, \Sigma, P, S_0\}$ is a grammar, with N being a set of non-terminal symbols, Σ being the alphabet of terminal symbols, P being the set of production rules and S_0 being a start symbol, and with the property that there is a representation of P such that the left side of each rule consists of exactly one symbol from the set of non-terminals N. Figure 2.3 shows an example of a context-free grammar and a derivation of an input expression.

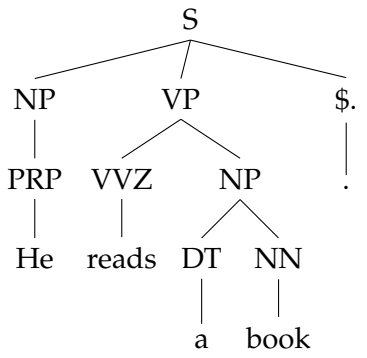
For constituency parsing, the non-terminal symbols denote the category of a phrase, e.g. noun phrase or verb phrase, and thus the relation between their sub parts, while the terminal symbols of alphabet Σ are either words or atomic categories denoting morpho-syntactic properties of a word, e.g. part-of-speech tags. The root node spans the whole input expression and the terminal nodes relate to an input symbol each. Each application of a production rule is marked by a father node in the hierarchical structure.

$N = \{S, NP, VP, PRP, VVZ, DT, NN, \$.\}$
$\Sigma = \{He, reads, a, book, .\}$
$P = \{$

S	→	NP VP \$.	PRP	→	He
NP	→	DT NN	VVZ	→	reads
NP	→	PRP	DT	→	a
VP	→	VVZ NP	NN	→	book
			\$.	→	.

$\}$
$S_0 = S$

(a) Small context-free grammar



(b) Constituency tree

Figure 2.3: A context-free grammar and a derivation tree for the sentence from Example (1.1) on Page 22.

A main aspect, which is however not necessarily marked explicitly in the phrase structure tree, is the lexical head of a phrase. It is the main part of each phrase and determines its features (Langer 2004), for example case or number agreement of noun phrases. In the tree, it is often implicitly referred to by the category of the phrase, e.g. noun phrase, verb phrase or prepositional phrase.

In Figure 2.3b the non-terminal categories comprise the start symbol S for the constituent which spans the whole sentence, VP for a verb phrase, NP for a noun phrase and the part-of-speech tags PRP, VVZ, DT, NN and $. for personal pronoun, verb, determiner, noun and punctuation respectively. The alphabet consists of the lexical items.[7]

A typical parsing algorithm to produce phrase structure trees is the Cocke-Younger-Kasami algorithm (CYK algorithm). It requires the grammar to be in Chomsky normal form, i.e. the left-hand side of each rule to consist of exactly one non-terminal symbol and the right-hand side of each rule to consist either of two non-terminal symbols or of exactly one terminal symbol. The data structure which the CYK algorithm applies is an $n \times n$ matrix, with n being the number of input tokens. This matrix, also called **chart** (e.g. Schmid 2004), is filled with the non-terminal symbols from which a span of the input can be produced.

Actually, the CYK algorithm decides the word problem for context-free grammars in Chomsky normal form, i.e. it decides whether the input segment is a formal word in the language defined by the grammar; this is true, when the start symbol appears in the leftmost column of the last row of the chart. What can be used in addition for parsing however, is that all possibly utilized production rules can be extracted from the chart, such that it is possible to extract all derivation trees for the input segment from it, i.e. the parse forest.

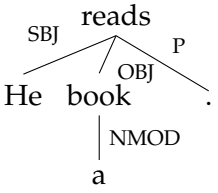
Parsing algorithms making use of such tabular data structures are sometimes called **Chart Parser**; another example is the algorithm by Earley (1970).

Dependency parsers. Dependency parsers are based on the concept of dependency relations between input symbols, e.g. an object acting as the dependent of a verb, or an adjective acting as the modifier of a noun. The main aspect of

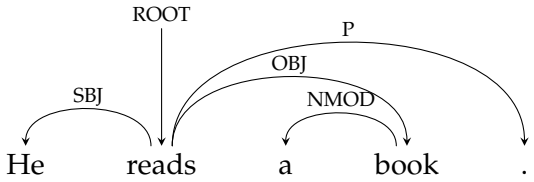
[7] The analysis is based on the constituency tree from Figure 1.1a (Page 22) but with simplified categories.

dependency structures is thus the fact that the syntactic structure is built by words, which are connected by binary, asymmetrical[8] relations (cf. Kübler et al. 2009: p. 2). In these relations one word is the dependent, which syntactically depends on the other word, the head. Thereby each dependent has exactly one head, but each head can have one or more dependents.

Since a head can in turn be the dependent of another head, the dependency analysis also creates a hierarchical structure up to a "highest" head, which is not a dependent itself. Contrary to a constituent structure, no additional nodes are introduced. In dependency structures, the common expectation is that the verb determines the syntactic structure of an expression, while all other words directly or indirectly depend on the verb (Langer 2004). Thus the root of the dependency analysis is usually a verb.



(a) Nested dependencies



(b) Dependency arcs over input sequence

Figure 2.4: Two representations of dependency graphs.

Since the notion of head plays such an important structural role, heads are explicitly marked in the analysis. Figure 2.4 shows two types of representations for the same dependency analysis. Figure 2.4a highlights the hierarchical relations between the input symbols by building a tree structure from the top most head, the verb, to the dependents which are no heads themselves. Figure 2.4b implements the relations over the ordered token sequence of the

[8]The relation is unidirectional, and for all words w1 and w2 the following holds: If w1 is related to w2 then w2 cannot be related to w1.

input expression, thus moving the focus from the hierarchy to the relations. The representation from Figure 2.4a is called **nested tree**, because it fulfils the nested property (Kübler et al. 2009: p. 17) stating that the nodes of every subtree of the tree are a contiguous sequence from the input.

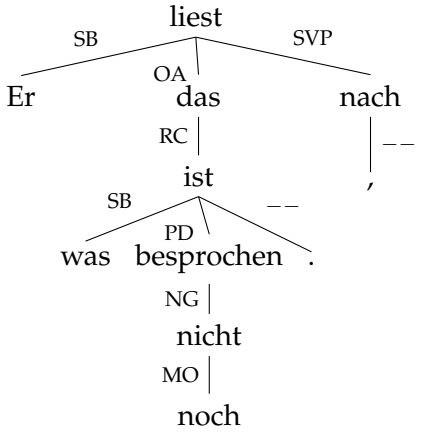
A dependency tree is called **projective**, if its arc representation can be drawn without crossing edges. Kübler et al. (2009: p. 17) show that the nested property holds for all projective dependency trees. **Non-projective** dependency trees are however still trees and could thus be drawn in a hierarchical representation without crossing edges, also if the nested property does not hold. Figure 2.5 shows two representations of the same non-projective dependency analysis by the mate parser (cf. Bohnet (2010) and Section 2.1.4) for the sentence in Example (2.6).[9] Figure 2.5a shows a tree representation, for which the nested property does not hold, Figure 2.5b shows the arc representation with ordered token sequence and crossing edges. The crossing edges are due to the attachment of the relative clause (RC) and the fact that verb particles can appear separated from their base verb in German (SVP). However, there are also non-projective structures in e.g. English analyses, cf. Kübler et al. (2009: Figure 2.1).

(2.6) *Er liest das nach, was noch nicht besprochen ist.*
he reads that after what still not discussed is
'He reads up on what is not discussed yet.'

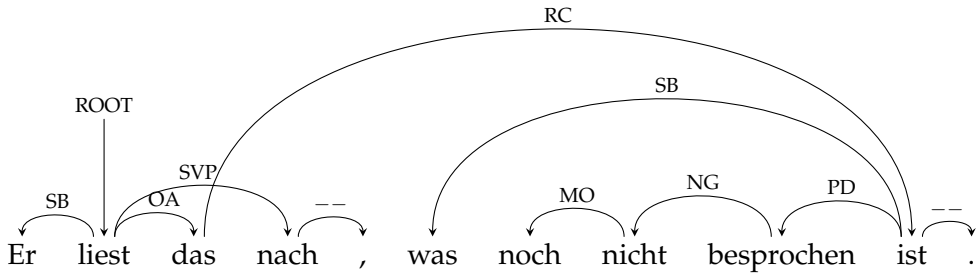
An purely representational issue is the notation of the directionality of the edges. For tree representations as in Figure 2.4a, there is a clear intuition that the edges start at the head and end at the dependent, to conform to tree properties. Representations of the dependency edges as in Figure 2.4b, do not predetermine the direction of the edges. While we follow most of the notions and representations utilized in Kübler et al. (2009), regarding the orientation of the dependency edges in the arc representation, we choose the other alternative for the following examples[10] and represent dependency edges as directed from the dependent to the head. This decision only concerns the

[9] This example was inspired by a sentence from the SdeWaC corpus. The corpus is applied in the case studies for Task I (Section 6.1.1).

[10] Figures 1.1b, 2.4b and 2.5b applied the direction of head to dependent.



(a) Tree without crossing edges



(b) Arc representation with ordered token sequence and crossing edges

Figure 2.5: Two representations of a non-projective dependency tree.

representational level and serves to provide a smooth introduction to the tabular output representation of the dependency parsers utilized in this work. In the tabular formats the position of the head and the label of the relation are listed as an annotation for each token such that the head position is more intuitively the target of an outgoing edge. In the following, both, the tabular format and the arc representations will be utilized in examples, depending on the relevant aspects to show.

Another representational artefact is that often all tokens in the sentence have to be handled in the same way and thus also each node has to be represented as a dependent. Therefore there is often a virtual additional symbol assigned to

the output (e.g. a node called TOP or called ROOT as in Figures 2.4b and 2.5b), which simulates the head for all those tokens which otherwise would not be a dependent of any other token of the input expression (e.g. the main verb or the sentence final punctuation, if punctuation symbols are included in the output structure). Illustrations of the dependency analyses occurring in this work will thus also show additional edges to a virtual root node.

Rule-based and data-driven approaches. Rule-based parsers rely on a set of rules, such as in a symbolic context-free grammar, which allows to derive analyses of input sequences, or on a set of constraints, such as in constraint grammar (Karlsson 1990), which restricts prespecified readings until no further constraint is applicable.

As already mentioned in Chapter 1, an analysis based on rule-based approaches evaluates the well-formedness of the input with respect to the applied grammar. However, the systems are not robust with respect to unexpected input. Phenomena of domain or register change as well as corrupted input sequences are highly likely to produce no analysis at all. Of course a grammar can evolve over time, rules or constraints can be added or modified when new phenomena are encountered, but since not all phenomena will be seen, only parts can be covered. Setting up a respective grammar requires conceptual and manual effort, even more so, when further development is envisioned.

Data-driven systems base their analyses on data they have 'seen', i.e. on the basis of which they were able to extract statistical information. A probabilistic context-free grammar can be extracted from an annotated treebank, i.e. a large set of syntactic analyses, by taking the frequency of applied rules into account. Parsing models can also be trained on such treebanks, by defining features which typically allow them to build analyses for input sequences even if they are outside of the scope of the grammar the treebank was based on.

While data-driven approaches are more robust to unexpected input than rule-based approaches, their search space includes wrong analyses and they do not evaluate (grammatical) well-formedness of the input (Langer 2004) besides formal criteria such as a tree constraint. However, data-driven approaches allow for forced guessing, and can thus propose a single output even in highly

ambiguous cases. They automatically specify a disambiguation strategy, which has to be designed separately for rule-based approaches.

For many data-driven approaches at least some annotated data is necessary, i.e. some annotation guidelines have to be defined, and the data has to be annotated. Or, as in active learning, the system output can be corrected by the user to improve the system by application. There is thus also conceptual and/or manual effort included and the trained systems reflect a domain specific bias of the data they were trained on.

While both types of approaches, rule-based as well as data-driven, can be beneficial based on the application scenario, a promising concept are hybrid approaches, in which some rules or constraints are added to robust data-driven systems to provide linguistic information, e.g. for disambiguation or relabeling (Seeker et al. 2010).

2.1.4 The parsers utilized in this work

In this section we will introduce those parsing systems, which have been utilized in the case studies, cf. Chapter 6. The case studies operate on German data, thus parsers for German are presented here. Nearly all of the parsers have some connection to the TIGER treebank[11] (Brants et al. 2002, 2004), a newspaper corpus based on articles from several categories of the *Frankfurter Rundschau*. The corpus includes about 900,000 tokens and has been annotated with lemmas, part-of-speech tags, morphological features and syntactic structure. Cf. Albert et al. (2003) for the annotation scheme regarding the syntactic annotations. The data is available in several versions and encodings such as TIGER-XML, its own XML-based representation (König et al. 2003), which has also become the basis for the serialization format of the Syntactic Annotation Framework (SynAF), cf. ISO 24615-1:2014; ISO/PRF 24615-2.

This section makes use of several made-up example sentences, Examples (2.7) to (2.10),[12] by means of which the different analyses of the parsers are presented.

[11] PID: `http://hdl.handle.net/11022/1007-0000-0000-8E50-6`

[12] The sentences are partly inspired by sentences encountered in the corpora which are utilized in the case studies and/or aspects of specific parses which played a role in processing. Those cases have been combined into the examples here to be introduced in a more general fashion

The examples cover several syntactic structures which are either typical candidates for feature extraction, e.g. verb argument structure; or which tend to be treated differently by different parsers, e.g. coordination; or which pose problems to most parsing systems, e.g. disambiguation of the attachment of prepositional phrases. The example sentences are from the sports news domain. Since most of the parsers have been trained on or built for text from the news domain, these sentences do not confront the parsers with completely unexpected syntactic structures, while still including usage from a specific domain.

(2.7) *Stroh-Engel und Behrens legten nach.*
Stroh-Engel and Behrens layed after
'Stroh-Engel and Behrens scored also.'

(2.8) *Kempe konnte gegen St. Pauli punkten.*
Kempe could against St. Pauli score
'Kempe could score against St. Pauli.'

(2.9) *Es ist Marcel Hellers Schnelligkeit, mit der auf dem Platz*
it is Marcel Heller's speed with which on the field
nur wenige mithalten können.
only few keep up can
'It is Marcel Heller's speed, which only few can keep up with on the field.'

(2.10) *Wagner war bereits bei der*
Wagner was already at the
U-21-Europameisterschaft erfolgreich: 2009 schoß er
UEFA European Under-21 Championship successful 2009 kicked he
zwei Tore für die deutsche Elf.
two goals for the German eleven
'Wagner had already been successful at the UEFA European Under-21 Championship: in 2009 he scored two goals for the German football team.'

The mate parser. The mate parser[13] (also sometimes referred to in publications as the Bohnet Parser or Mate Tools Parser) is a data-driven dependency parsing

and with only a few example sentences. For the case studies, Chapter 6 shows real examples from the data sets.

[13] PID: `http://hdl.handle.net/11022/1007-0000-0000-8E4E-A`

system (Bohnet 2010; Bohnet and Nivre 2012; Bohnet and Kuhn 2012). It is part of the Mate Tools collection, which consists of modules to create a pipeline for natural language analysis. These modules comprise part-of-speech taggers, lemmatizers, morphological taggers, parsing modules and semantic role labellers. For the parser module there are two implementations: a graph-based dependency parser and a transition-based dependency parser. The modules are data-driven, i.e. for each module a trained model is needed to complete the pipeline[14]. The German models available with the tool have usually been trained on a dependency conversion of the TIGER treebank by Seeker and Kuhn (2012) and thus apply tags from the TIGER treebank.

For the analyes in Figures 2.6 to 2.9 a version of the pipeline of lemmatizer, part-of-speech tagger, morphological tagger and graph-based parser was applied.

Figure 2.6b shows the actual output representation of the mate parser, the tabular format from the 2009 CoNLL shared task (Hajič et al. 2009). The columns represent the following attributes from left to right: token id (ID), consisting of the sentence number and the token position separated by an underscore; token form (FORM); gold lemma (LEMMA) and predicted lemma (PLEMMA); gold part-of-speech (POS) and predicted part-of-speech (PPOS); gold morphological features (FEAT) and predicted morphological features (PFEAT), each consisting of a set of values separated by the vertical bar: |; head (HEAD) and predicted head (PHEAD), which are represented by the head's token position and 0 for the virtual root node; dependency relation (DEPREL) and predicted dependency relation (PDEPREL); and 2+n additional columns for the semantic role labelling part of the shared task, which do not play a role for the examples here. For the annotations, the gold column contains data of gold-standard (i.e. high) quality, on which a system can either be trained or evaluated. The predicted columns contain the output predicted by the automatic system, thus in Figure 2.6b the predicted columns are filled, and the gold columns are empty or contain the invalid head position -1. Further examples in this work will only show the actually filled columns for readability reasons. Sentence borders are denoted by an empty line after the analysis. The mate parser returns

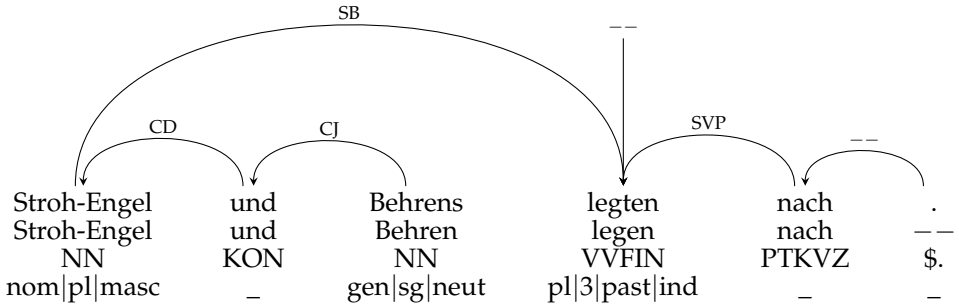
[14]Each of the modules provides the functionality to train respective models on input data.

one fully disambiguated analysis, which is the most probable one based on its model.

Figure 2.6a visualizes the tabular output from Figure 2.6b in an arc representation. The dependency edges are directed from dependent to head, such that the edge visualizes the representation from the tabular output where the head is listed as an annotation for a specific token.

Figure 2.6 shows that the coordination is represented as a chain, where the first conjunct is the head and that the separated particle of the verb *nachlegen* ('here: [to] score again, also: [to] put more of sth. into sth.') is attached to its base verb with a relation label SVP denoting a separated verb particle. The sentence end punctuation is attached directly to the preceeding token. In Figure 2.7 the split token[15] *St. Pauli* ('district St. Pauli of the city of Hamburg'), is connected by a PNC relation connecting components of a proper noun. Figure 2.8 shows the long distance relation to attach the relative clause RC and a correct attachment of the prepositonal phrase *auf dem Platz*. Figures 2.8 and 2.9 show that also sentence-internal punctuation is attached to the preceeding token.

[15]The tokenization is not part of the mate analysis, but expected to be conducted in advance, such that the input is tokenized.



(a) Arc representation of the analysis

1_1	Stroh-Engel	_	Stroh-Engel	_	NN	_	nom|pl|masc	-1	4	_	SB	_	_
1_2	und	_	und	_	KON	_	_	-1	1	_	CD	_	_
1_3	Behrens	_	Behren	_	NN	_	gen|sg|neut	-1	2	_	CJ	_	_
1_4	legten	_	legen	_	VVFIN	_	pl|3|past|ind	-1	0	_	−−	_	_
1_5	nach	_	nach	_	PTKVZ	_	_	-1	4	_	SVP	_	_
1_6	.	_	−−	_	$.	_	_	-1	5	_	−−	_	_

(b) Output in CoNLL2009 format

Figure 2.6: Analysis of Example (2.7) by the mate parser.

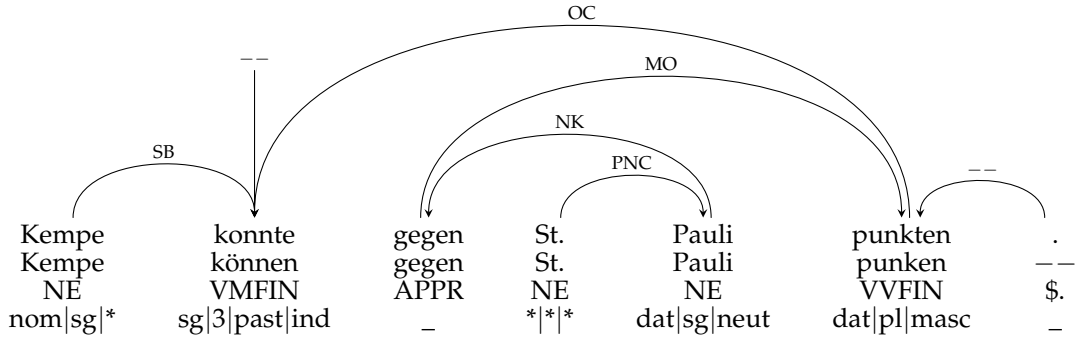


Figure 2.7: Analysis of Example (2.8) by the mate parser.

Figure 2.8: Analysis of Example (2.9) by the mate parser.

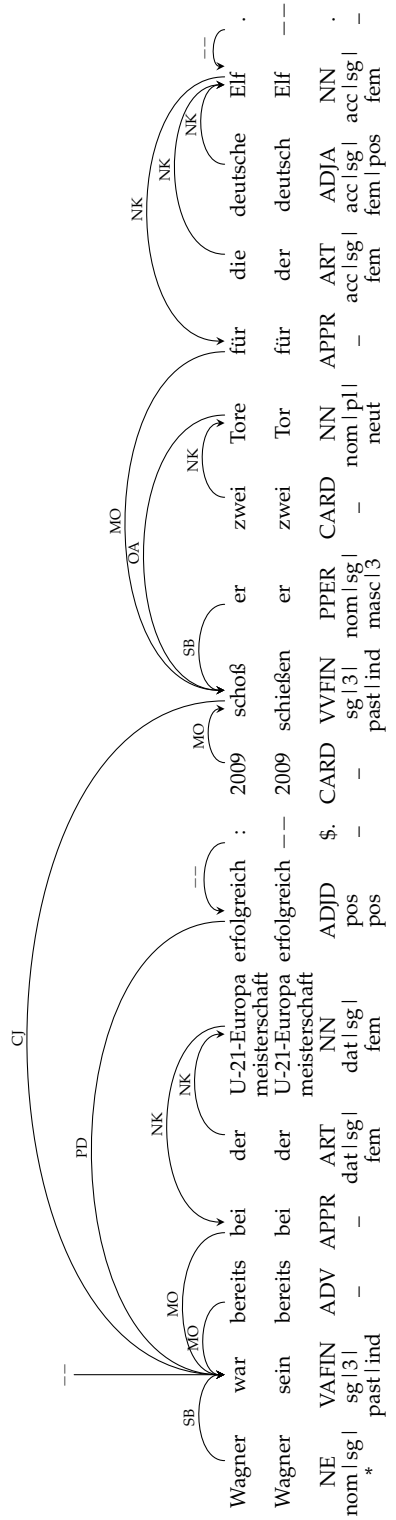


Figure 2.9: Analysis of Example (2.10) by the mate parser.

FSPar. FSPar[16] is a rule-based dependency parser with a large lexical knowledge base (Schiehlen 2003). The parser is based on Abney (1996)'s partial parsing with finite state cascades and consists of several transducers which handle the input regarding simple noun phrases, coordinated noun phrases and verbal phrases for verb-first, verb second and verb-final structures. Thereby a transducer can also process the input right-to-left, which is employed for maximal embedding in the verb-final transducer. Next to a lexicon-based tokenization process, in which the input is enriched with all information from the lexicon including several part-of-speech tags from which the tagger can then choose afterwards, the parsing step employs a large subcategorization lexicon. FSPar includes also a tagging step by the TreeTagger (Schmid 1994) employing a tagset based on the STTS (Schiller et al. 1999).

FSPar offers several output formats. A tabular format similar to the CoNLL format shown above[17] is shown in Figures 2.10 to 2.13. The columns are: token position, starting from 0; token form; part-of-speech; lexically enhanced lemma; morphological features; head; dependency relation. All following columns can have different interpretations, e.g. the last two columns are repeated to provide more possible analyses, for relative pronouns an additional (possibly underspecified) head can be given to attach the relative clause as a whole, and for most personal and possessive pronouns the last column provides possible antecedents. Among others, further output formats are machine-readable dependency tuples and trees of the substructures recognized by the transducers.[18] The examples from FSPar are presented in the tabular format in this work because FSPar outputs underspecified analyses and utilizes an extended head concept, such that for an analysis from FSPar there is no default arc representation.

Regarding ambiguous cases, FSPar returns underspecified analyses, e.g. with respect to different possible heads (two heads separated by two vertical bars: ||, in Figure 2.13, Token 5 in the second sentence, denoting different possibilities

[16] PID: `http://hdl.handle.net/11022/1007-0000-0000-8E60-4`

[17] And even more similar to the CoNLL2006 format, also called CoNLL-X format.

[18] For more details on the processing chain of the parser and its output formats see Eckart (2009).

of the attachment of the prepositional phrase), different possible relations (two relations separated by one vertical bar: |, in Figure 2.13, Token 4, denoting an object in genitive case NP:2 or accusative case NP:8) or a combination of both (Figure 2.12, Token 5).

Due to FSPar's extended head concept, combined heads are possible (two token positions combined with a slash: /, in Figure 2.11, denoting the complex verb structure *konnte punkten*) and coordinations do not have to decide on a single head: The conjunction in Figure 2.10 is represented differently from the analysis of the mate parser. Here both conjuncts are subjects NP:1 to *legen* and the head of the conjunction *und* consists of both conjuncts (0|2). While this prevents artificial chain relations, regarding a purely token-based analysis, the strict single head constraint is violated.

The relative clause in Figure 2.12 (Token 6) is ambiguously related to either *Marcel_Heller* or the correct token *Schnelligkeit*, to which also the relative clause from the mate parser analysis was related.

FSPar relies heavily on its own preprocessing pipeline and also has a strict sentence segmentation, so the input is split into two sentences at the colon in Figure 2.13. It also combines tokens and introduces an underscore for the respective whitespace in the input (Figure 2.11, Token 3 and Figure 2.12, Token 2), thus the token positions might deviate from analyses of other systems. Parts of the input which cannot be regularly attached to a head are attached to a virtual root node (-1)[19]. Sentence final punctuation is attached to the root node, sentence internal punctuation to the respective token. Tokens for which no head can be found are also attached directly to the root node. This convention can be exploited to estimate difficulties the parser had with a respective sentence.[20]

Some information from the lexical knowledge base is propagated to the lemma column of the output, e.g. particle verbs are marked by # between particle and base verb in their lemma (Figure 2.10, Token 3 and Figure 2.12, Token 12)

[19]This differs from the token numbering of the mate parser. The mate parser starts the first token at position 1 and the root node is 0, thus a -1 is invalid for a mate parser analysis, for FSPar the first token position is 0 and the root node is -1.

[20]Cf. Faaß and Eckart (2013).

and also additional geographical information is represented (Figure 2.11, Token 3 and Figure 2.13, Token 7 in the second sentence).

Since the German tagset utilized for the TreeTagger is based on the same tagset as the TIGER treebank, the same part-of-speech tags appear in the analyses of FSPar and the mate parser. An example is the tag PTKVZ for separated verb particles, which plays an important role in the case studies of Task I (Section 6.1).

```
<s>
  0  Stroh-Engel   NN     Stroh-Engel   Nom:M:Pl                                          3   NP:1
  1  und           KON    und           _                                                 0|2 KON
  2  Behrens       NE     Behrens:H     Nom:M:Sg|Dat:M:Sg|Akk:M:Sg|Nom:F:Sg|Dat:F:Sg|Akk:F:Sg 3   NP:1
  3  legten        VVFIN  nach#legen    1:Pl:Past:Ind|3:Pl:Past:Ind|3:Pl:Past:Konj        -1  TOP
  4  nach          PTKVZ  nach          _                                                 3   RK
  5  .             $.     .             _                                                 -1  TOP
</s>
```

Figure 2.10: Analysis of Example (2.7) by FSPar.

```
<s>
  0  Kempe      NE      Kempe:H                             Nom:Sg                             4/1  NP:1
  1  konnte     VMFIN   könnenI                             1:Sg:Past:Ind|3:Sg:Past:Ind         -1  TOP
  2  gegen      APPR    gegen                               Akk                                4/1  ADJ
  3  St._Pauli  NE      1._FC_St._Pauli|Sankt_Pauli:Stadt   Akk:N:Sg                             2  PCMP
  4  punkten    VVFIN   punkten                             Inf                                4/1  RK
  5  .          $.      .                                   |                                   -1  TOP
</s>
```

Figure 2.11: Analysis of Example (2.8) by FSPar.

<s>							
0	Es	PPER	es	Nom:N:Sg	3/1	NP:11	
1	ist	VAFIN	seinA	3:Sg:Pres:Ind	-1	TOP	
2	Marcel_Hellers	NE	Marcel_Hellers:H	Gen:M:Sg|Gen:F:Sg	3	GL	
3	Schnelligkeit	NN	Schnelligkeit	F:Sg	3/1	RK	
4	,	$,	,	_	3/1	PUNCT	
5	mit	APPR	mit	Dat	12/13||3	ADJ|PP/mit:4	
6	der	PRELS	d	Dat:F:Sg	5	PCMP	2||3
7	auf	APPR	auf	Dat	12/13||3||6	ADJ	
8	dem	ART	d	_	9	SPEC	
9	Platz	NN	Platz	Dat:M:Sg	7	PCMP	
10	nur	ADV	nur	_	12/13||11	ADJ	
11	wenige	PIS	wenig	Nom:Pl	12/13	NP:1	
12	mithalten	VVINF	mit#halten	Inf	12/13	RK	
13	können	VMINF	könnenI	1:Pl:Pres:Ind|3:Pl:Pres:Ind|3:Pl:Pres:Konj	3/1	ADJ	
14	.	$.	.	_	-1	TOP	
</s>							

Figure 2.12: Analysis of Example (2.9) by FSPar.

<s>							
0	Wagner	NE	Wagner:H	Dat|Nom:Sg	6/1	NP:1|NP:4|NP:1	
1	war	VAFIN	seinA	1:Sg:Past:Ind|3:Sg:Past:Ind	-1	TOP	
2	bereits	ADV	bereits	|	6/1||3	ADJ	
3	bei	APPR	bei	Dat	6/1	ADJ|ADJ|PP/bei:4	
4	der	ART	d	|	5	SPEC	
5	U-21-Europameisterschaft	NN	U-21-Europa#@meisterschaft	Dat:F:Sg	3	PCMP	
6	erfolgreich	ADJD	erfolg#@reich	|	6/1	RK	
7	:	$.	:	|	-1	TOP	
</s>							
<s>							
0	2009	NE	2009	Dat|Akk	1	NP:8|NP:4	
1	schoß	VVFIN	schießen	1:Sg:Past:Ind|3:Sg:Past:Ind	-1	TOP	
2	er	PPER	er	Nom:M:Sg	1	NP:1	0:1+
3	zwei	CARD	2	|	4	ADJ	
4	Tore	NN	Tor	Gen:N:Pl|Akk:N:Pl	1	NP:2|NP:8	
5	für	APPR	für	Akk	1||4	ADJ	
6	die	ART	d	|	8	SPEC	
7	deutsche	ADJA	De:Region:Adj	|	8	ADJ	
8	Elf	NN	Elf	Akk:F:Sg	5	PCMP	
9	.	$.	.	|	-1	TOP	
</s>							

Figure 2.13: Analysis of Example (2.10) by FSPar.

BitPar. BitPar[21] (Schmid 2004, 2006) is a constituency parser for probabilistic context free grammars. In this description, we refer to the tool as described by Schmid (2004). It is based on the CYK algorithm described in Section 2.1.3. However, it goes along the lines of Langer (2004)'s statement of efficiency: making use of a three dimensional chart which is implemented as two bit vectors reducing memory usage and runtime (Schmid 2004).

In this work, BitPar is used together with a probabilistic context-free grammar for German, extracted from the TIGER treebank. However, the treebank has been transformed and enhanced with some additional information, before the grammar was extracted.

The extracted information from the transformed treebank includes about 125,800 rules with a set of about 30,650 non-terminal symbols. The terminal symbols are the part-of-speech tags, since the lexical information is not included in the grammar but by means of an additional lexicon and a word-class recognizer for unknown vocabulary. However, the output of BitPar are constituency trees including the tokens (Figures 2.14 to 2.17). Nodes denoting constituents consist of a category (S, VP, NP), a function derived from TIGER treebank edges (HD, SB, MO)[22] and optional morphological or lexical features. Part-of-speech nodes consist of the part-of-speech tag (NN, VVFIN, PTKVZ), a function and morphological features where applicable. Since the function labels and part-of-speech tags originate from the TIGER treebank they can be found in the mate parser analyses and in BitPar analyses. The part-of-speech tags from the TIGER treebank originate from the same tagset the part-of-speech tags utilized by FSPar are based on. By default, BitPar outputs the most probable parse tree, based on its grammar. However an n-best list can be accessed.

The coordination in Figure 2.14 is combined under one constituent, a coordinated noun phrase CNP. The separated verb particle is marked by part-of-speech tag PTKVZ and function SVP but their connection is only visible through being on the same level, while being both attached to the constituent spanning the sentence. Since tokenization is also not part of the BitPar processing the tokenization is the same as applied for the mate parser. In Figure 2.15 *St. Pauli* is

[21] PID: `http://hdl.handle.net/11022/1007-0000-0000-8E53-3`

[22] The root node has no function, because it has no incoming edge.

combined under a proper noun constituent PN. The relative clause is marked as a sentence constituent with the relative clause function RC in Figure 2.16 and the example in Figure 2.17 is analysed as one sentence. Sentence final punctuation is attached to the root node and sentence internal punctuation (Figures 2.16 and 2.17) is attached to the respective (possibly coordinated) sentence constituent. Figure 2.17 shows an embedding within the prepositional phrase *für die deutsche Elf* and analyses the prepositional phrase as part of the noun phrase. This analysis differs from the analysis by the mate parser, while both attachments are proposed by FSPar.

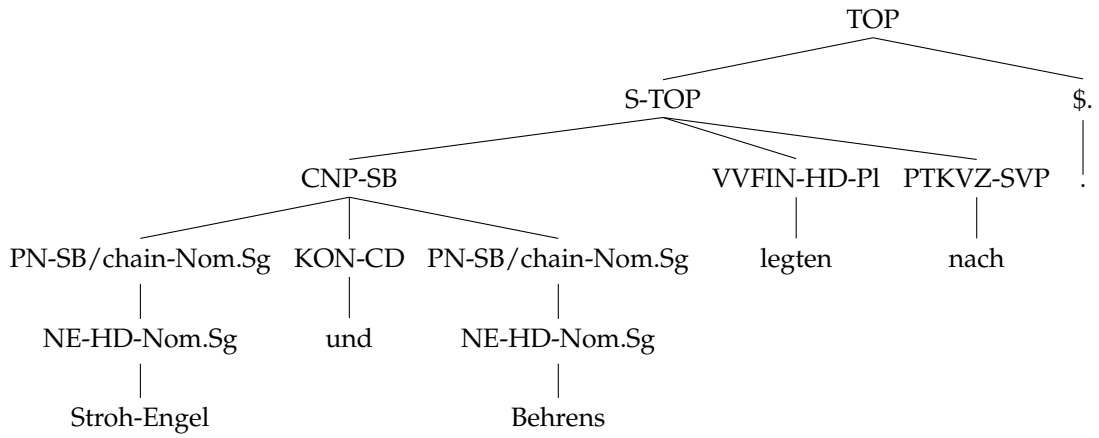


Figure 2.14: BitPar analysis of Example (2.7).

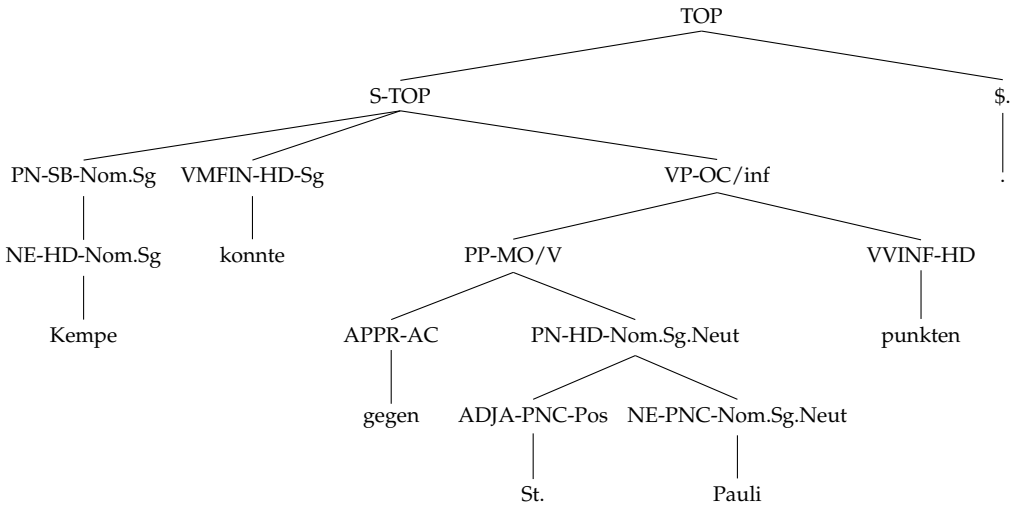


Figure 2.15: BitPar analysis of Example (2.8).

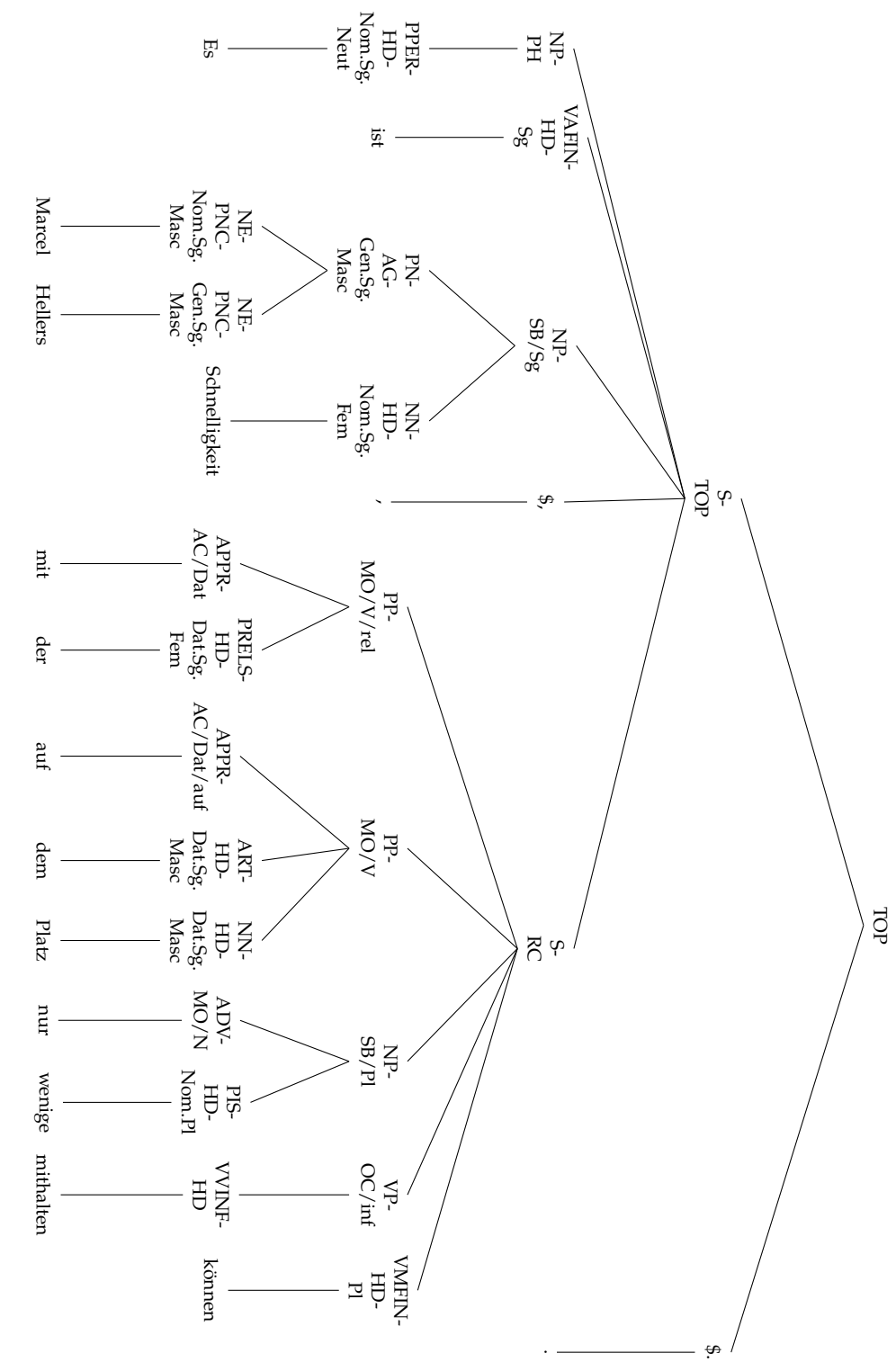


Figure 2.16: BitPar analysis of Example (2.9).

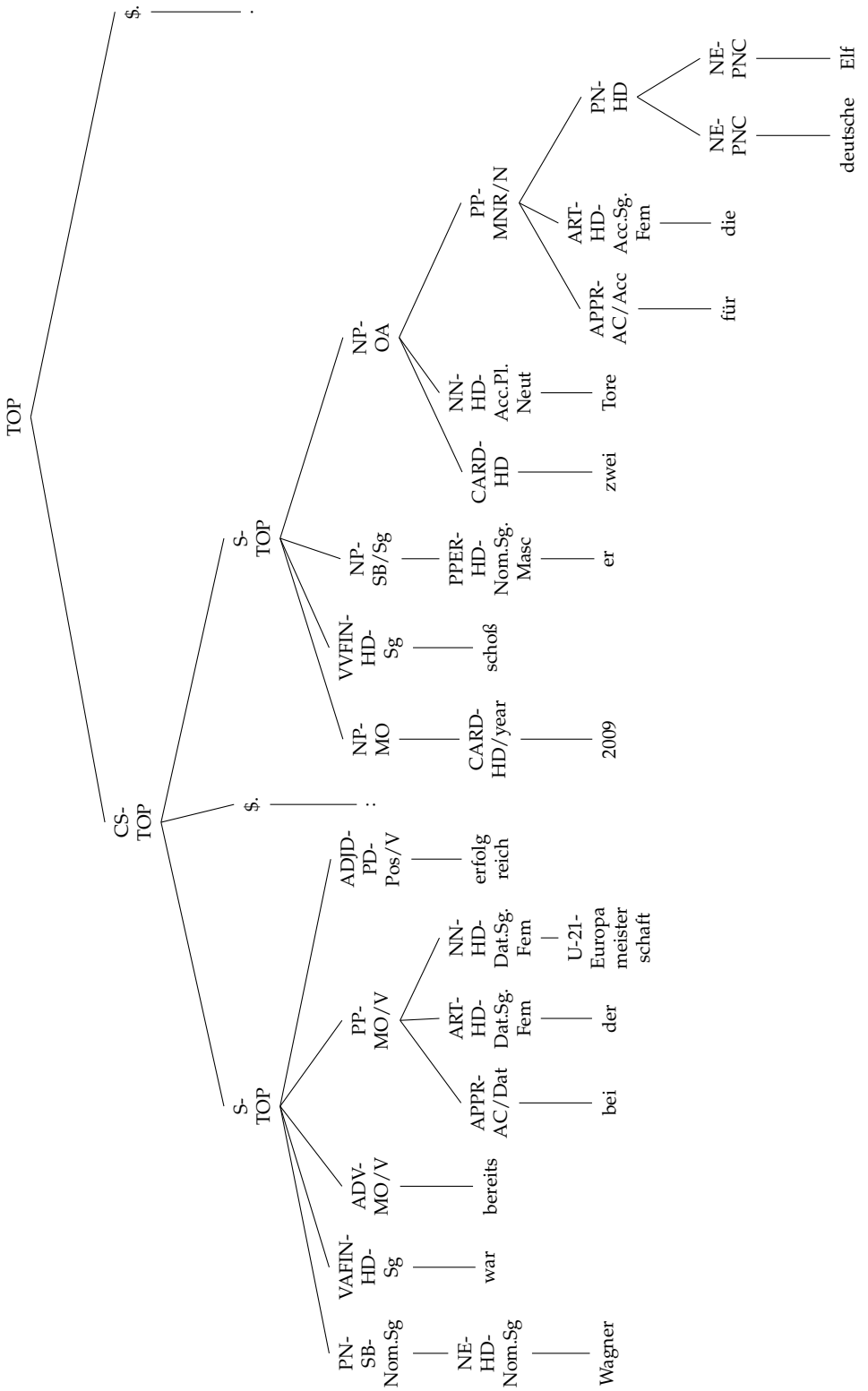


Figure 2.17: BitPar analysis of Example (2.10).

The IMS-SZEGED-CIS parser. The IMS-SZEGED-CIS parser (Björkelund et al. 2013a) took part in the shared task on statistical parsing of morphologically rich languages (SPMRL) 2013 (Seddah et al. 2013).[23] The group took part in the dependency as well as the constituency track of the shared task, however the case study in Section 6.2 applies the constituency system, so for this work the IMS-SZEGED-CIS parser refers to a data-driven constituency parser.

The system employs products of probabilistic context free grammars with latent annotations. Grammar training starts from different initializations on the same data set, and the product can then be seen as a combination approach within the parser. For preprocessing, the system applies morpho-syntactic tagging, such that rare words can be replaced by morphological tags, relying on the handling of unknown words by the tagger. After the product parsing a reranker is applied.

The input is expected to be tokenized, so for Figures 2.18 to 2.21 the same tokenization was applied for the IMS-SZEGED-CIS parser and BitPar. The output is a fully disambiguated analysis, chosen according to the reranker.

Again, the set of constituents and part-of-speech tags is based on the sets from the TIGER treebank, however without the functional information. The root node, which is called TOP in the BitPar output is called VROOT for the IMS-SZEGED-CIS parser. The analysis in Figure 2.18 is very similar to the one by BitPar, only missing the proper noun categories (PN) in the coordination. Figure 2.19 analyses the verb *punkten* erroneously as a separated particle and thus misses also the verbal phrase from the BitPar output. Figures 2.20 and 2.21 show that all punctuation is located directly under the root, and that overall the structures from the IMS-SZEGED-CIS parser are flatter than those of BitPar, cf. the noun phrase *Marcel Hellers Schnelligkeit* or the prepositional phrase *für die deutsche Elf*. Like BitPar, the IMS-SZEGED-CIS parser embeds the latter phrase within the noun phrase *zwei Tore für die deutsche Elf*.

[23] A further version took part in the SPMRL 2014 shared task (Seddah et al. 2014) as IMS-Wrocław-Szeged-CIS system (Björkelund et al. 2014a).

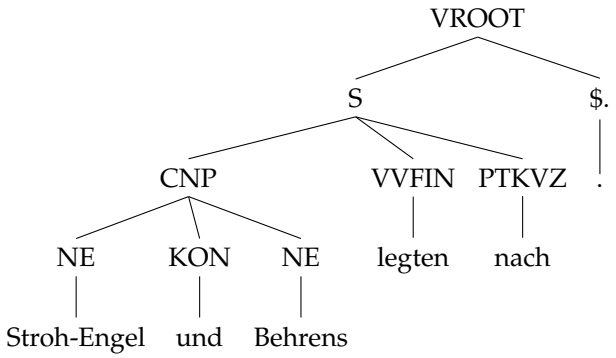


Figure 2.18: Analysis of Example (2.7) by the IMS-SZEGED-CIS parser.

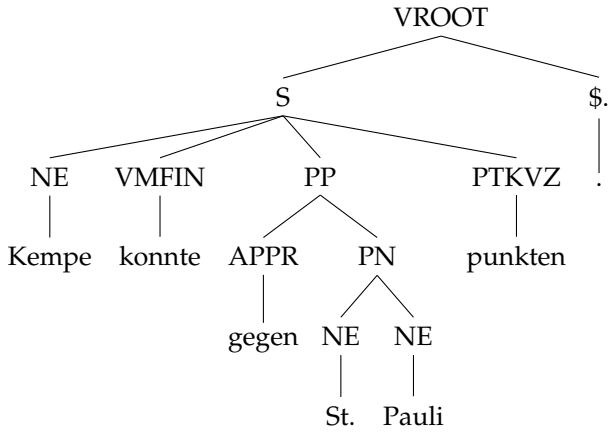


Figure 2.19: Analysis of Example (2.8) by the IMS-SZEGED-CIS parser.

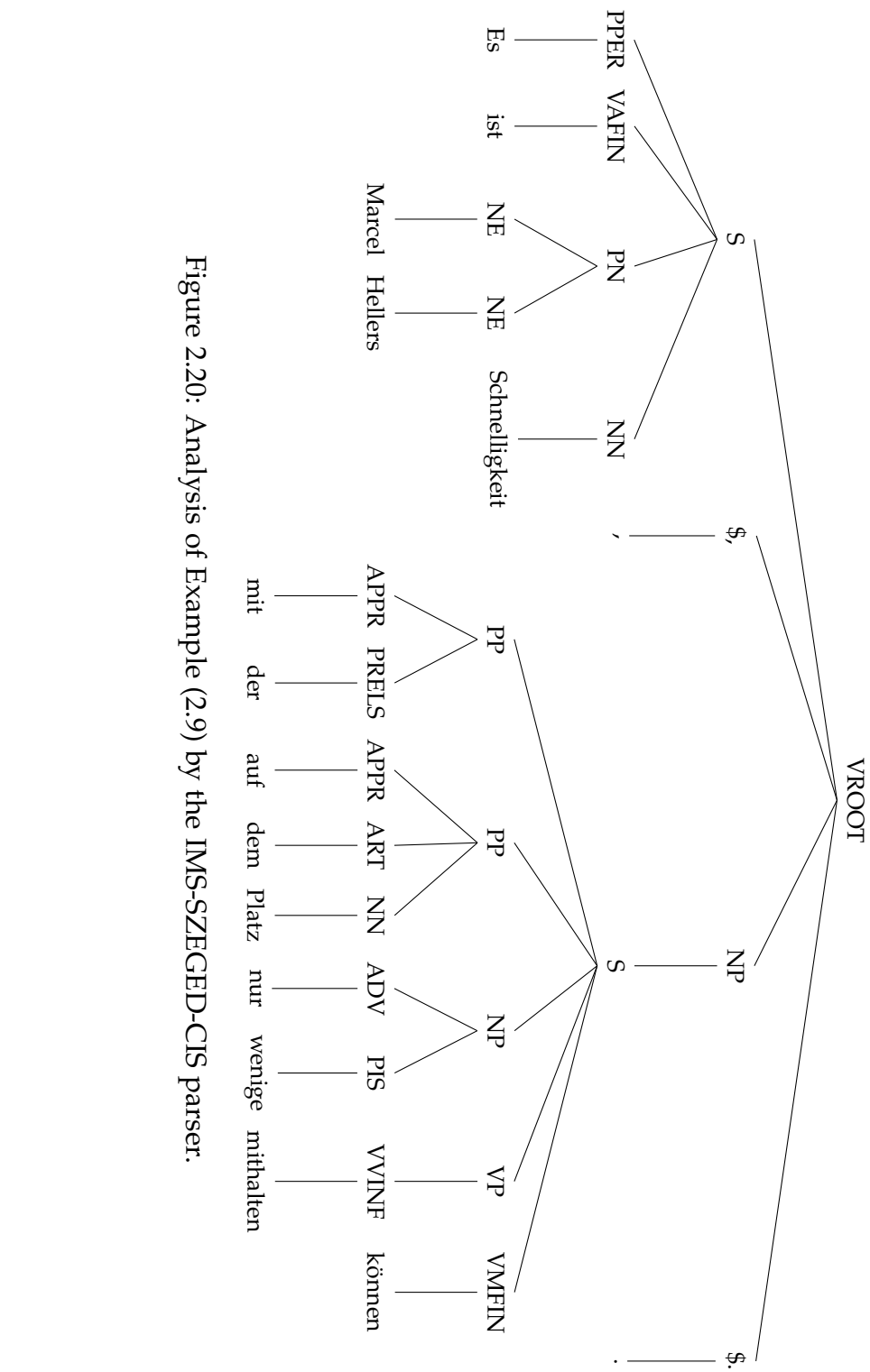


Figure 2.20: Analysis of Example (2.9) by the IMS-SZEGED-CIS parser.

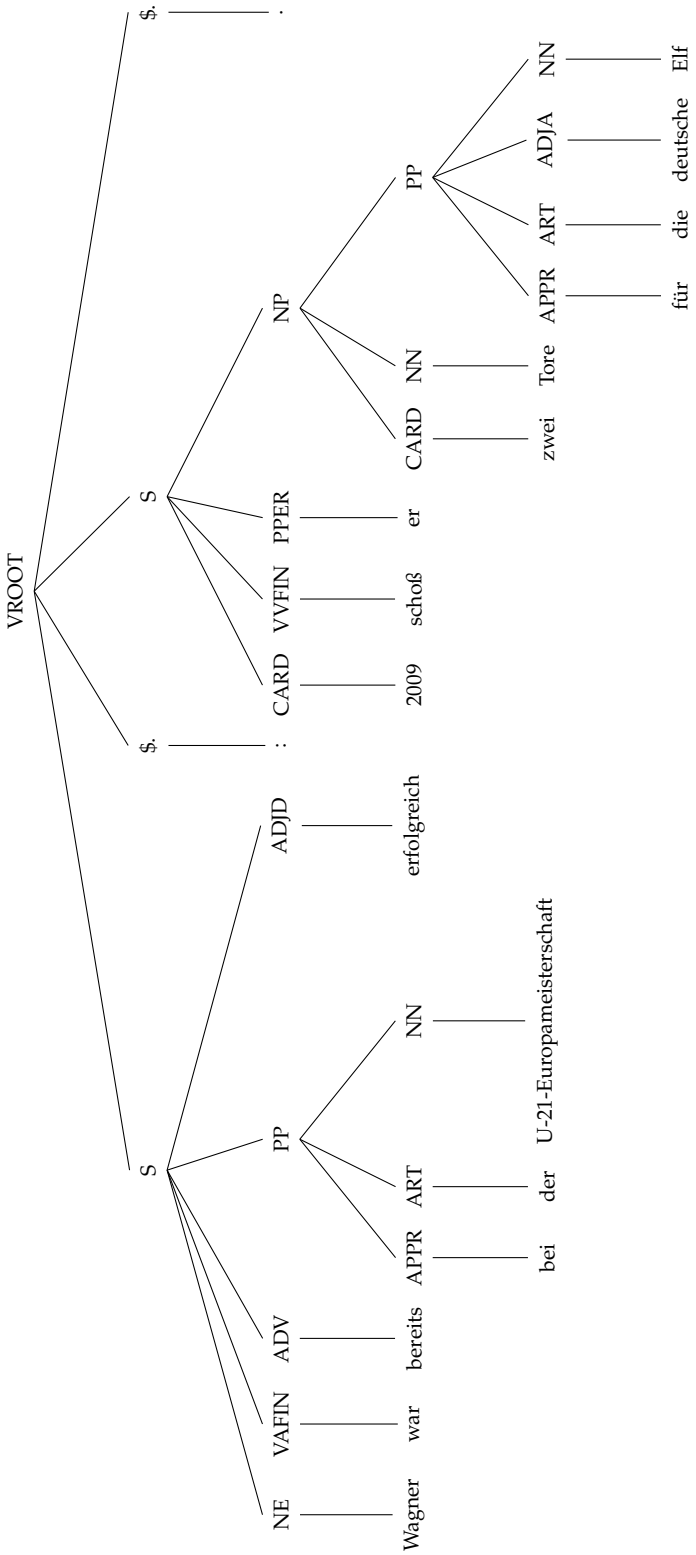


Figure 2.21: Analysis of Example (2.10) by the IMS-SZEGED-CIS parser.

The LFG parser. What is called LFG parser here, actually refers to a combination of a Lexical Functional Grammar (Kaplan and Bresnan 1982) and a parser. The Xerox Linguistic Environment (XLE) (Crouch et al. 2011) serves as parser and development platform (Langer 2004). With an LFG, there are two types of output: the c-structure (constituent structure) and the f-structure (functional structure). An f-structure example can be found in Figure 1.1c (Page 22). Since only the c-structure plays a role in the case studies of this work, Figures 2.22 to 2.25 only display c-structures and we categorize the LFG parser in our context as rule-based constituency parser. The utilized grammar has been tested and enhanced based on the TIGER treebank (Rohrer and Forst 2006), however the tagsets differ from the output of the other parsers discussed so far.[24] As with FSPar, the output of the LFG parser often contains several possible readings, which are represented in one (underspecified) f-structure and several c-structures. Figure 2.22 shows two readings from the set of c-structures: Figure 2.22a applies a common noun for *Stroh-Engel*, which literally translates into 'straw angel', a sort of Christmas tree decoration. In this context however it is a name, so the reading from Figure 2.22b should be preferred. However both examples show, that the LFG parser makes extensive use of unary branching, while the coordination is only marked by the tag CONJCO of the conjunction *und*.

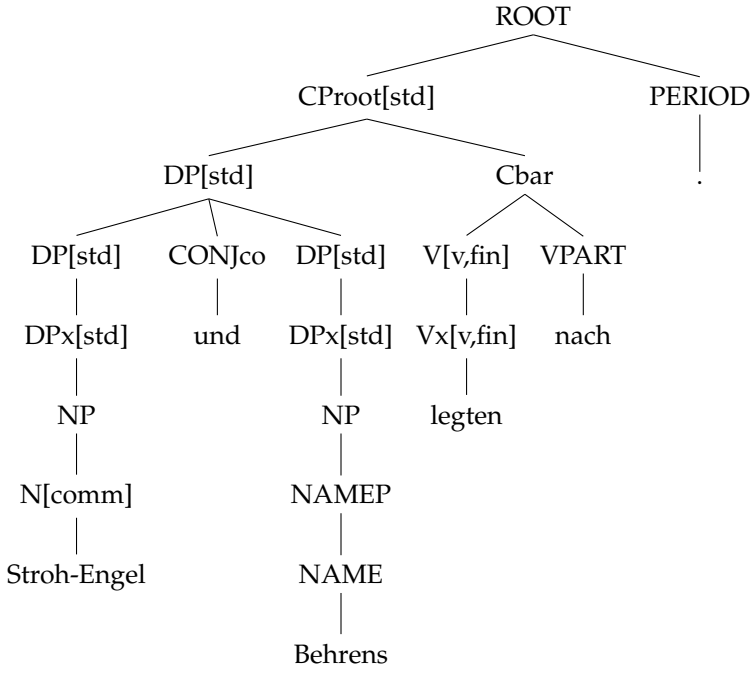
Disambiguation of multiple readings can be done by the user, by selecting a reading; however XLE can also provide a ranking of the output and even suppress dispreferred constructions.

While sentence segmentation is expected, the LFG parser provides its own tokenization and combines *St. Pauli* into one token (Figure 2.23), like FSPar, but keeps *Marcel Heller* as two tokens, combining them in a specific phrase (NAMEP, Figure 2.24). Regarding punctuation, the colon (Figure 2.25) and the sentence final punctuations are attached to the root nodes. However the comma is further embedded and Figure 2.24 shows a specificity of the LFG parser output: it includes additional tokens, in cases where actually two punctuation symbols would have followed each other but one is omitted (haplology, Forst

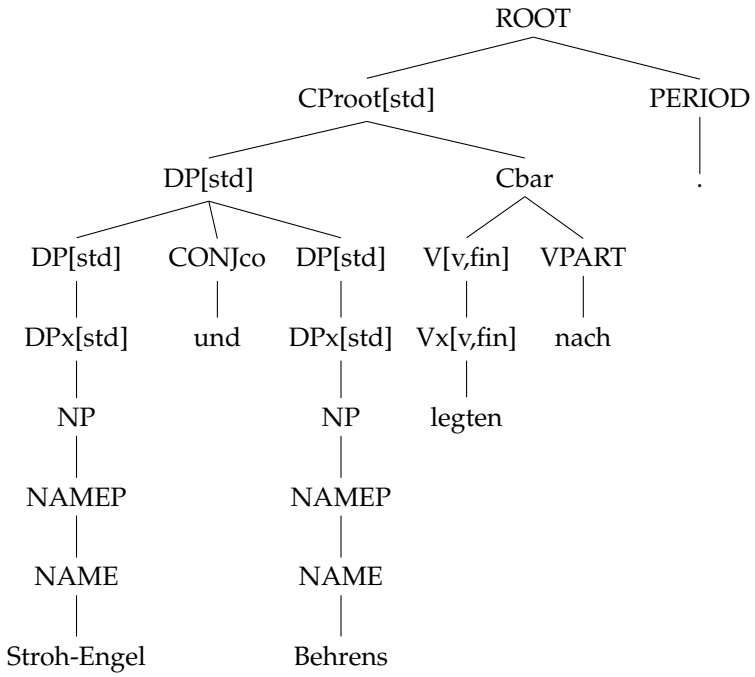
[24] Cf. Zinsmeister et al. (2002) for a discussion of similarities in LFG and TIGER treebank annotations.

and Kaplan 2006). Like in Figure 2.24 the additional operational commas are marked by the part-of-speech tag HAP-COMMA.

Figure 2.24 exemplifies the granularity of the tagset for the constituents by means of the prepositional phrases. A prepositional phrase with a relative pronoun *mit der* is denoted as PP[rel], differing from the PP[std] *auf dem Platz*. This distinction is also found in the BitPar output, however only in the functions (Figure 2.16), but not at all in the output of the IMS-SZEGED-CIS parser (Figure 2.20). To compare the depth of embedding, the phrase *für die deutsche Elf* can be applied. The IMS-SZEGED-CIS parser (Figure 2.21) applies a completely flat structure, in which all tokens directly constitute the prepositional phrase. BitPar (Figure 2.17) embeds an additional proper noun constituent, while the LFG parser (Figure 2.25) embeds a noun phrase (NP), a determiner phrase (DP) and a unary adjective phrase (AP) below the prepositional phrase (PP).



(a) Common noun



(b) Proper noun

Figure 2.22: Two readings from the analysis of Example (2.7) by the LFG parser.

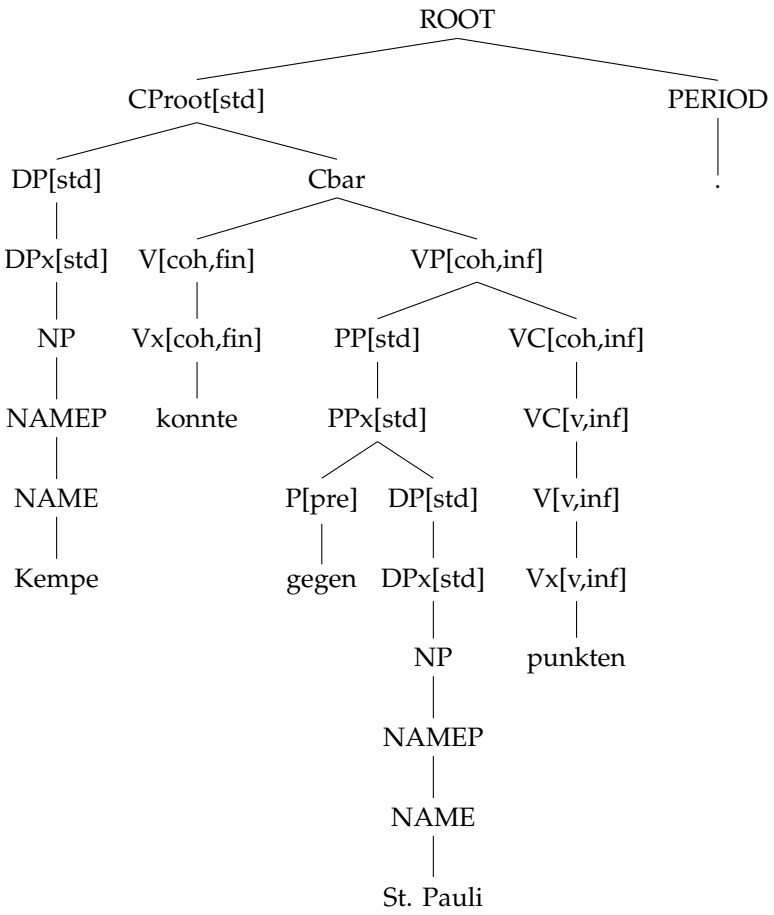


Figure 2.23: Analysis of Example (2.8) by the LFG parser.

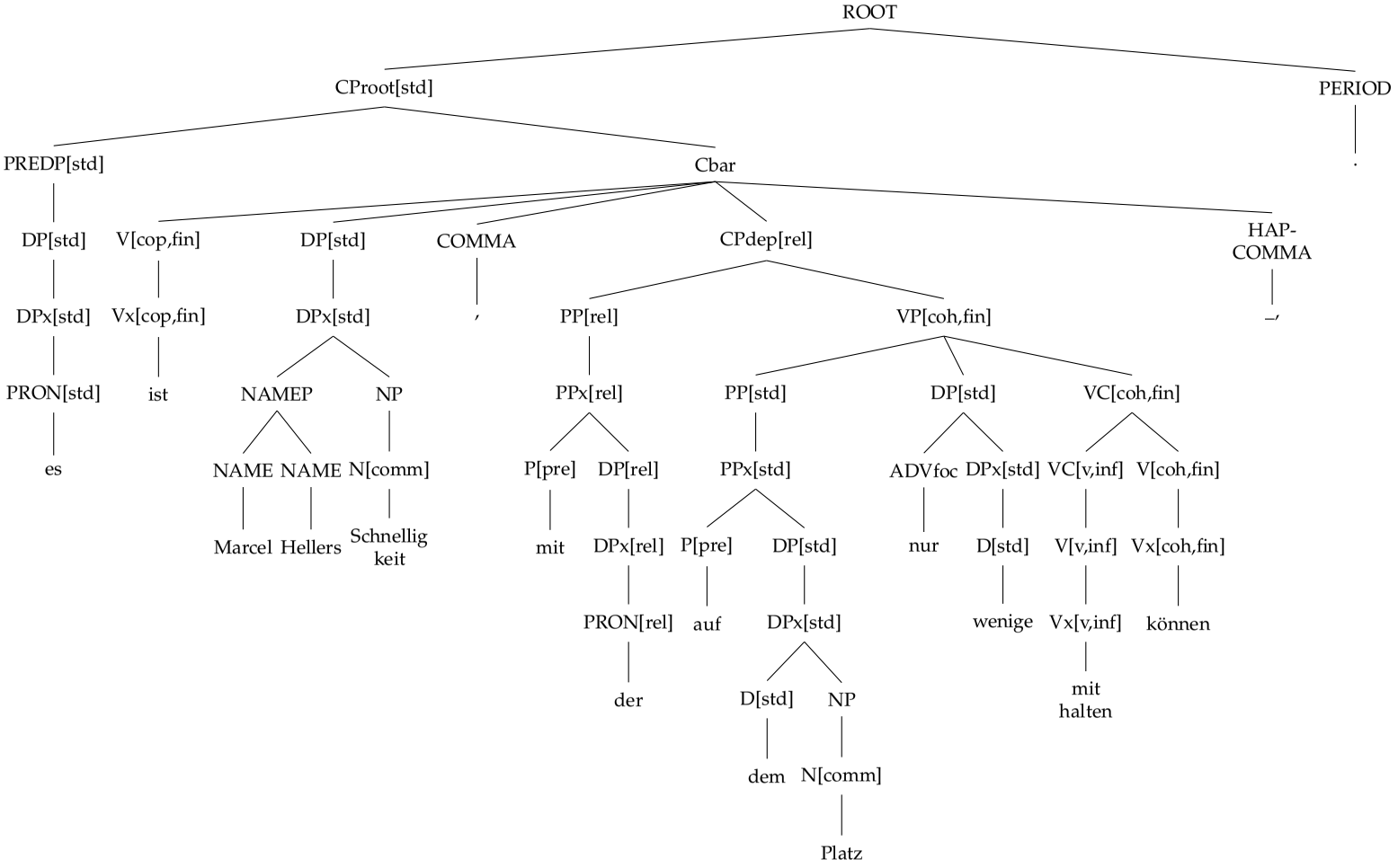


Figure 2.24: Analysis of Example (2.9) by the LFG parser.

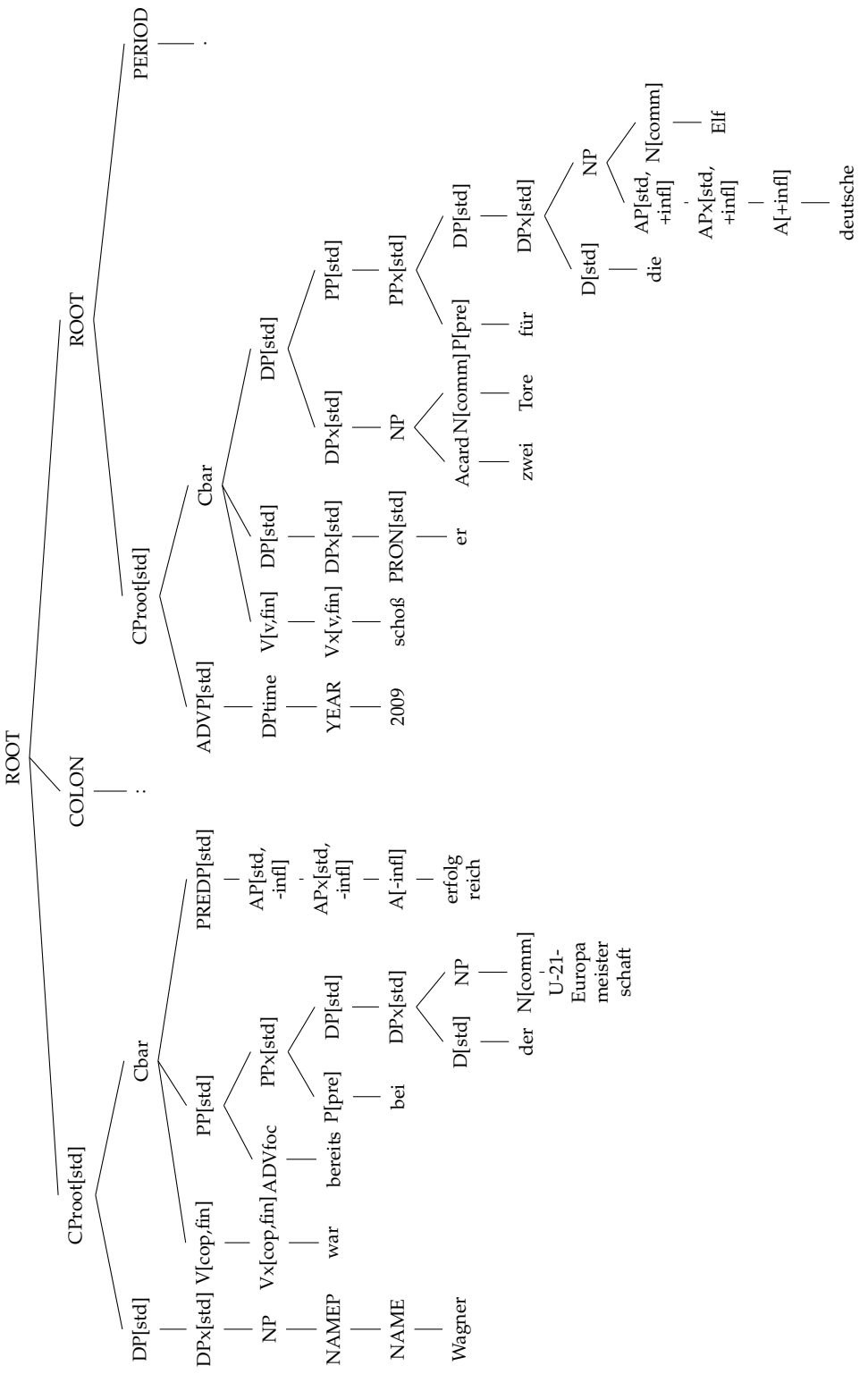


Figure 2.25: Analysis of Example (2.10) by the LFG parser.

2.2 Combination approaches

Section 2.1 included a short introduction to some of the main approaches to parsing: regarding theory and representation we focussed on dependency parsing and constituency parsing; regarding implementation we presented data-driven as well as rule-based parsing. Additionally five parsing systems were introduced which can be categorized along the lines of these approaches.

As discussed in Chapter 1, the parsers, however, cannot be perfect in the sense that they would provide for each input exactly the single, fully specified analysis which was intended by the author. This holds for all approaches to parsing. Thus, the output from every parser might include errors which decrease the reliability of the syntactic information derived from this output.

Combining the capabilities of different systems to increase the reliability of the result is neither a new procedure nor is it restricted to parsing. Section 2.2.1 is thus devoted to earlier work on the combination of the output of tools in natural language processing. Sections 2.2.2 to 2.2.4 focus on several existing approaches to parser combination and while the approach of this work is distinct from machine learning (cf. Section 1.2.7), yet Sections 2.2.2 and 2.2.4 will also include classifier combination as in machine learning, which is traditionally intrinsically evaluated, aiming for the best replication of the annotations from a specific gold standard. In contrast to that, Section 2.3 will present some contributions which do take the task into account, but still tend to rely on a single system. For combination issues in resource interoperability see Section 3.4.

2.2.1 Areas of tool combination in natural language processing

The idea of improving components in natural language processing by combination has been around for a while and is not restricted to parsing, as the following examples show.

Fiscus (1997) introduced ROVER (Recognizer Output Voting Error Reduction), a post-processing combination approach for automatic speech recognition (ASR). Even if two systems show a similar performance (in the case of Fiscus (1997) regarding word error rates), this does not mean that the output

of the systems includes exactly the same errors. This observation introduces the importance of the error distribution of different systems to a combination approach.

The experiments from Fiscus (1997) were set up to exploit the performance of different systems, thereby decreasing the overall word error rate. In this experiments, firstly the output from several ASR systems is combined into a single word transition network (including empty transitions), then a simple voting is applied, i.e. the voting is done for each node and its outgoing edges; no other context is taken into account. The three applied voting schemes are:

1. frequency of occurrence – the relative frequency of each word type appearing on the outgoing edges decides on the result. This scheme does not require any kind of learning, but produces tie votes where multiple word types appear with the same frequency.[25]
2. frequency of occurrence and average word confidence – in addition to the relative frequency, confidence values for the transitions are taken into account. The average confidence values for each word type found at the outgoing edges go into the overall score. This scheme requires a confidence value for each output word produced by the ASR systems and the training of two parameters, (i) for the confidence value of empty transitions and (ii) for the weight trade-off between frequency and confidence values.
3. frequency of occurrence and maximum word confidence – like the second scheme, but with the maximum confidence value for each word type. Thus this scheme also requires confidence values and the training of two parameters.

The three schemes were evaluated on settings from two benchmark tests. In the first setting, scheme 1 reduces the error rate by 1%, scheme 2 reduces the error rate by 3.2% and scheme 3 reduces the error rate by 4.1%. Interestingly the parameter for the trade-off between confidence and frequency behaves differently in scheme 2 and 3. When average confidence values are applied, these are much more important than the frequency values (0.8 to 0.2), while when the maximum confidence values are applied, the weight of the confidence values is lower than that of the frequency values (0.3 to 0.7).

[25] Tie votes enforce an arbitrary decision between the options.

In the second evaluation setting, for all three schemes the error rate decreases by 5.3-5.6%. While the explicit numbers rank scheme 3 the highest and scheme 1 the lowest, the improvement of scheme 2 over scheme 1 was multiply tested as non significant by the author. The trained parameters did apparently not generalize over different application sets. However, the two application sets seem to differ in terms of the included input systems, which makes the actual finding the necessity to adapt the parameters to the set of input systems. This is not so much unexpected, and the procedure proposed in this thesis also relies, for the development of its combination schemes, on the involved systems.

Lastly, the contribution by Fiscus (1997) shows that the overall results of the combination show a lower error rate than the best involved single system: the major motivation for all combination approaches. In a detailed analysis they show that this is not true for every single data point (i.e. segment hypothesis), but for a majority of these, which results in the overall better performance of the combination.

Fiscus (1997) was not the first to exploit combination in natural language processing. Similar approaches appeared in machine translation (Frederking and Nirenburg 1994) and part-of-speech tagging (Tapanainen and Voutilainen 1994).

Tapanainen and Voutilainen (1994) combine a rule-based and a data-driven system to decide on correct part-of-speech tags. The input is preprocessed with a lexicon-based morphological analyser which assigns the possible part-of-speech tags to the words. Words which are not recognized, and have thus no entry in the lexicon, are processed by rule-based heuristics based on pre- and suffixes, and again several possible part-of-speech tags can be assigned. The combination happens for the disambiguation of the part-of-speech analyses. The rule-based system, which is the original follow up of the above mentioned morphological analyser, applies a large set of grammar-based constraints and optionally a small set of heuristic constraints. The trained data-driven system applies a different tagset which is mapped onto the (richer) tags of the rule-based system based on a decision list. For each tag from the data-driven system its decision list ranks the possibly fitting tags from the tagset of the rule-based system. The

output of the data-driven system is taken into account in the cases where the output of the morphological analyser was ambiguous. The disambiguation of the set of alternatives can happen in two modes: either the full decision list is taken into account and only tags not appearing in the decision list are removed from the set or the first, i.e. most probable, entry of the decision list is applied (which according to the authors not always deletes alls ambiguity[26]).

This leaves several combination configurations to test, including or not including the heuristic constraints of the rule-based system and applying one of two modes of the disambiguation of the data-driven system. Out of these configurations Tapanainen and Voutilainen (1994) compare three single system configurations for disambiguation and three combination approaches.

These configurations are evaluated along two dimensions: remaining ambiguity and error rate. While the single rule-based configurations have the lowest error rate, they have the highest rate of remaining ambiguity. The single data-driven configuration has the highest error rate with low remaining ambiguity, however two combination configurations reach full disambiguation with an error rate clearly below that of the single data-driven configuration. All tested combination approaches show a lower error rate than the single data-driven approach and a lower rate of remaining ambiguous words than the rule-based approaches, two of them even lower than the single data-driven approach.

Also for complex systems such as in machine translation a combination approach is feasible as could already be seen in the early approach of Frederking and Nirenburg (1994). They applied three different machine translation systems: a knowledge-based system, an example-based system and a lexical transfer system, and relied heavily on estimated quality measures of single chunks to compute the best combination of translation components. The knowledge-based system and the example-based system provided quality scores themselves, while for the lexical transfer system an estimated reliability of the databases was applied. The quality scores had to be normalized to be comparable and the length of the chunk was taken into account. In a recursive divide and

[26]While not stated explicitly, this could probably be the case when the decision list has no overlap with the proposed set of tags or when two tags in the decision list share the first rank, being equally likely.

conquer approach, the input was split into parts and the exhaustive set of all possible combinations of components was utilized to compute the overall translation. They applied a keystroke evaluation, counting the number of activities of a human translator utilizing the result of the single systems and the combined system respectively in a translator's workstation and found the output of the combined system to be better than the available single systems.

In all these approaches the combined systems are preferred over the participating single ones and especially the approach by Tapanainen and Voutilainen (1994) is important for this work, since they (i) combine rule-based with data-driven systems, (ii) apply systems with different tagsets and different tokenizations and (iii) utilize a very basic rule-based approach in output combination. Similarly, this work applies rule-based and data driven systems in case studies for two example tasks (Sections 6.1 and 6.2) some of which make use of different tokenization aspects and tagsets, but while Tapanainen and Voutilainen (1994) go for a broad mapping approach, this work applies a task-based approach, focussing the need for interoperability on the concepts relevant for the task (cf. Chapters 3 and 5). Finally, this work also proposes rule-based combination as a promising method and goes beyond the approach of Tapanainen and Voutilainen (1994) in taking more details of the output of the single systems into account (cf. Section 6.1.3).

Although his topic is completely different from parsing, the approach of this work is based on Fiscus (1997)'s ROVER ideas. In fact, it adopts the idea of the error distribution and like ROVER, it focusses on combinations of single parts of different analyses instead of reranking complete analyses. The output of the systems is the input for the combination rather than combining the systems as such. Furthermore this work focusses on simple voting approaches rather than exhaustive parameter training to provide an abstract workflow which can be instantiated also by non-parsing experts (cf. Chapter 5).

After discussing related combination approaches for a subset of different areas, of course the main focus of this work is on parsing. The following sections will present related work regarding combination approaches to parsing and will compare them with respect to three main aspects: (i) which kinds

of single systems are chosen or created to take part in the combinations (cf. Section 2.2.2), (ii) which basic structures are taken from the single systems and their outputs to form the eventual outcome (cf. Section 2.2.3), and (iii) which combination methods are applied (cf. Section 2.2.4). Thereby aspects (ii) and (iii), taken together, are subsumed by the concept of the **combination scheme** in the task-based parser output combination workflow presented in Chapter 5 and in the related case studies in Chapter 6.

Most of the related work mentioned here proposes and compares several combination approaches itself. The following sections include main approaches from Henderson and Brill (1999); Zeman and Žabokrtský (2005); Sagae and Lavie (2006); Surdeanu and Manning (2010); McDonald and Nivre (2011) and Ballesteros (2012).

2.2.2 Methods for the provision of parsing systems to be used in combinations

This section focusses on the single systems which take part in the combination approaches. As stated before (Section 1.3) and seen in the combination approaches for other areas above (Section 2.2.1), we expect that the more different the systems are, the more different will be their error distributions and the higher the gain of a combination approach.

The objective here is not to compare the performance of specific single systems, but to inspect the outcome of different combination approaches and to see how they perform in relation to the single systems involved in the combination. Thus, this section will classify the respective systems along the categories of tools and parsing techniques established above, i.e. constituency vs. dependency and data-driven vs. rule-based parsers, cf. Section 2.1.3. It will not actually introduce the single parsers applied in the related work. For information on the single systems applied there, e.g. the names, algorithms and training features of the parsers, the readers are referred to the original papers. All single systems which are applied in the case studies of this work, cf. Chapter 6, are introduced in Section 2.1.4.

What will be mentioned here are the data sets which have been used to train, tune or test the single parsers. On the one hand, because treebanks[27] are comparatively rare and thus many approaches are based on the same data, and on the other hand to document the validity of the respective evaluations.

Henderson and Brill (1999) combine three data-driven constituency parsers, while Sagae and Lavie (2006) combine five such tools. Henderson and Brill (1999) report that the parsers they use have been trained on several (and maybe different) sections of the Wall Street Journal part of the Penn Treebank (Marcus et al. 1993). However since sections 22 and 23 were not used for any of the parsers, for the combination they make section 23 their development set and section 22 their test set. Sagae and Lavie (2006) also use the Penn Treebank, but use sections 22 and 00 as development sets and section 23 as the test set. They use sections 02 to 21 for training.

Sagae and Lavie (2006) also present combinations of four data-driven dependency parsers, making use of unlabelled dependencies and the same data split of the Penn Treebank as for their experiments with constituency parsers.[28] Data-driven dependency parsers are also combined by Surdeanu and Manning (2010): seven parsers, McDonald and Nivre (2011): two parsers, and Ballesteros (2012): n parsers. Zeman and Žabokrtský (2005) combine up to seven dependency parsers, but include also rule-based parsers in the setting.

Zeman and Žabokrtský (2005) use the Prague Dependency Treebank. The single data-driven systems have been trained on the training section of the *analytical level* of the treebank[29], which is the annotation layer for surface syntax. A rule-based system is reported to apply lexical lists based on the training set as well. For the combination, they use the *d-test* from the analytical level, which is classified as development set in the treebank, and they split it into a combination training set of 77 files and a combination test set of 76 files.

Surdeanu and Manning (2010) use seven dependency parsers, where six of them are variants of one system, which comes with different parsing algorithms

[27] Treebank usually refers to a corpus with syntactic annotations of gold-standard quality.

[28] For section 22 they only report its exact use as development set for the combination of constituency parsers, and for section 00 they do it only for the combination of dependency parsers, so maybe not both development sets have really been applied in both cases.

[29] `http://ufal.mff.cuni.cz/czech-parsing`

and parsing directions (left to right vs. right to left). They use the data from the CoNLL 2008 shared task (Surdeanu et al. 2008), i.e. the syntactic dependencies[30].

Ballesteros (2012) reports on an *N-Version* dependency parser, which is a combination of several versions of the same system: one general parser trained on the training set, and several specific parsers, trained only on extracted subsets of the training set, to obtain a parser for a specific pattern. Ballesteros (2012) reports on using the training and test splits for Spanish included in the CoNLL-X shared task for a first experiment. Since it is not stated otherwise, this discussion assumes that the same corpus is used in the rest of the reported combination experiments as well.

McDonald and Nivre (2011) combine two different data-driven dependency parsers.[31] The training sets of the CoNLL-X shared task (Buchholz and Marsi 2006) for all 13 languages are applied.

Most of the approaches are applied to English data (Henderson and Brill 1999; Sagae and Lavie 2006; Surdeanu and Manning 2010), but some approaches are also exemplified on other languages such as Czech (Zeman and Žabokrtský 2005), Spanish (Ballesteros 2012), or on all 13 languages from the CoNLL-X shared task (Arabic, Bulgarian, Chinese, Czech, Danish, Dutch, German, Japanese, Portuguese, Slovene, Spanish, Swedish, Turkish), cf. McDonald and Nivre (2011).

Regarding the sets of single systems which are applied in combination approaches, the main focus is on data-driven systems. This is not particularly surprising, since the approach of combining systems in natural language processing has been transferred from classifier combination in machine learning, and shared tasks such as conducted by CoNLL (Buchholz and Marsi 2006; Nivre et al. 2007; Surdeanu et al. 2008; Hajič et al. 2009), SANCL (Petrov and McDon-

[30]The CoNLL 2008 shared task also included semantic dependencies.

[31]The important difference, here, is that one of the parsers is a graph-based (MSTParser McDonald et al. 2006) and one is a transition-based parser (MaltParser Nivre et al. 2006), which provides the basis for the different error distributions, and thus for a successful integration/-combination. However, in this discussion we do not want to add another dimension to the classification of the single parsers beyond the data-driven/rule-based, constituency/dependency distinction discussed above; thus they are referred to as different data-driven dependency parsers here. For a detailed error analysis of the two parsers see McDonald and Nivre (2011).

ald 2012) and SPMRL (Seddah et al. 2013, 2014) have fostered the creation of a variety of data-driven parsers. However, a few approaches also take rule-based systems into account, such as Zeman and Žabokrtský (2005).[32]

Furthermore, the sets of single parsers which take part in the combination approaches can be roughly sorted into three groups:

1. Systems which are immediately applied as is. This includes pre-trained off-the-shelf parsers and mostly rule-based parsers[33]. In short, systems which are applied without additional training, modification or configuration.
2. Sets of different systems which are trained on the same training set.
3. Single systems which are trained on different training sets (or different parts of a training set) for the purpose of maximizing the gain of their combination.

The parsers from Henderson and Brill (1999) belong to group 1, because the authors state that they use the parsers as trained by their creators. Sagae and Lavie (2006) describe the training sets they used, thus at least part of the parsers seem to be trained by them (group 2).

Actually it is not always completely clear from the papers, if a data-driven parser has been trained with respect to the specific combination approach, or if it has been trained before and is thus applied as is with an available model. Therefore the distinction between groups 1 and 2 may seem a bit artificial for data-driven systems.

This work opts for an approach to "get the best out of what is already there". Thus the workflow presented in Chapter 5 discusses the tuning of the combination, but does not take a tuning of the single systems into account. Accordingly the case studies presented in Chapter 6 only apply parsers of group 1. Additionally, the case studies combine data-driven with rule-based systems and dependency parsers with a constituency parser, to broaden the range of different systems. Section 3.3 introduces a classification of combinations based

[32]Nevertheless, for a parser labelled as rule-based, they add lexical lists based on the training data.

[33]Rule-based systems can of course also be tuned to some extent taking training data into account, cf. Zeman and Žabokrtský (2005).

on structural and concept-related differences of the output from participating systems.

2.2.3 Basic structures for combination in parsing

In combination approaches, there is usually a basic unit of the original output or model, which will appear in a possible combination result. On a high level, this can be the full structure, e.g. when several outputs or some additional information is utilized to decide which full parse is to be chosen. This kind of ranking procedure is called *parser switching* by Henderson and Brill (1999), due to the fact, that the decision to take the output from a specific parser varies per sentence.

Alternatively, substructures can be combined, such as single constituents (Henderson and Brill 1999; Sagae and Lavie 2006) or subtrees from a dependency analysis (Ballesteros 2012). As opposed to the parser switching above, Henderson and Brill (1999) call this *parse hybridization*. Ultimately, and especially for dependency analyses, the substructures to be combined can be broken down to single edges, e.g. each candidate dependency or suggested head (Sagae and Lavie 2006; Surdeanu and Manning 2010; Zeman and Žabokrtský 2005).

A third approach uses features for combination. Either by describing the parser's own context (Henderson and Brill 1999), or by including additional information from the output of another parser (Surdeanu and Manning 2010; McDonald and Nivre 2011).

A major aspect in combination is the well-formedness of the resulting structure. When a choice between full parses takes place, as with the parser switching by Henderson and Brill (1999), the result is naturally well-formed within the respective framework of the chosen parser. The same is true, when features from one parser output become part of the actual parsing process of another.

When substructures are combined, well-formedness can only be guaranteed under specific conditions. Henderson and Brill (1999) show with their Lemma *No Crossing Brackets* that in a majority voting approach, where the number of votes for each included constituent is greater than half of the participat-

ing parsers, the constituents cannot cover crossing spans of the input. They argue similarly for a classifier approach where the probabilities for included constituents are greater than 0.5. In the approach of Ballesteros (2012) specific analyses of function words are included in a general dependency parse. The author argues that structure is not broken since function words are often heads of subtrees, but does not discuss cases in detail where a subnode of the specific tree has been incorrectly attached.

Another approach is to apply reparsing algorithms, e.g. to find the maximum spanning tree in a set of unlabelled candidate dependencies from several parser outputs (Sagae and Lavie 2006). Surdeanu and Manning (2010) compare a simple voting scheme which might produce ill-formed output with two reparsing algorithms on in domain and out of domain data, and they find that out of domain, the performance of both reparsing algorithms drops with respect to the voting approach, and even for in domain data, only one reparsing algorithm achieves a similar performance. Thus, one option is to take into account ill-formed structures, e.g. containing unattached nodes, multiple heads or cycles (Zeman and Žabokrtský 2005).

Since the present work has a task-based focus, in tasks which need well-formed structures other combination schemes are applied than in tasks which only take specific parts of the outputs at all into account. In the case studies presented in Chapter 6, one task only focuses on specific parts (Section 6.1) while the other task needs a well-formed structure for further analyses (Section 6.2); thus the combination schemes applied for the second task are based on the lemma by Henderson and Brill (1999). This work does not provide a case study to exemplify parser switching, which is in line with Henderson and Brill (1999)'s results, where even the best parser switching results for precision (90.78) and F-score (90.74) are lower than the combination of constituents (precision: 92.42, F-score: 91.25). Only recall suffers a bit from combining substructures (90.81 vs. 90.10).

2.2.4 Combination methods in parsing

After having seen which parts can be utilized in a combination approach, now the focus is on the possibilities of how to combine them. First several methods of voting will be discussed, then methods which involve a (possibly additional) classifier.

Voting

Voting comprises all methods where the participating systems can cast a vote for or against a structure being (part of) the result. The basic method is a **majority voting**, where a structure needs a majority of votes to be accepted. When combining substructures, a binary vote (yes/no) can be cast for each substructure which has been proposed by any of the participating systems, e.g. *constituent voting* by Henderson and Brill (1999).

There can also be more than two options, e.g. all proposed heads for a specific token (Zeman and Žabokrtský 2005; Surdeanu and Manning 2010). The decision which substructure to include can then be based on several thresholds (Zeman and Žabokrtský 2005): the absolute majority of votes, at least half of the votes or simply the most votes. See Table 2.1 for a simple example. There are six participating systems (A-F); for a specific token, each of them votes for a head (by its position in the sentence). System A votes for head 0, system B votes for head 1, etc. In case a) only head 4 gets two votes, all others get one vote. If the most votes are sufficient, the chosen head is 4, although it has neither an absolute majority nor half of the votes. In case b) head 4 gets half of the votes, but since there is an even number of participating systems, this is not the absolute majority, which is only the case in c), where four systems vote for the same head.

	a)						b)						c)					
head	0	1	2	3	4	4	0	1	2	4	4	4	0	1	4	4	4	4
system	A	B	C	D	E	F	A	B	C	D	E	F	A	B	C	D	E	F

Table 2.1: Three voting scenarios for possible heads of a token. Systems are marked with letters (A-F), heads are marked by their position in the sentence.

A specific handling is needed when a threshold is not reached or in case of ties. For a binary vote, choosing an odd number of systems prevents ties (Henderson and Brill 1999). For other approaches, there is often a fall-back option, e.g. applying the decision of the best single system (Zeman and Žabokrtský 2005). Alternatively some sort of weighting can be introduced. For a setting with an absolute majority Zeman and Žabokrtský (2005) try the following: if there is no absolute majority, the best system gets two additional votes; if there is still no absolute majority the two additional votes are switched to the second best system; if this still does not lead to a decision, the fall-back sets in and the decision of the best single system is applied.

Depending on the threshold and the number of participating systems, the chosen components might not be combined into one well-formed structure, e.g. some tokens are not attached or get more than one head. The recall of such methods suffers when these structures are excluded, however the methods are more precision-oriented. Zeman and Žabokrtský (2005) use at least half of the votes as threshold in a respective setting.

An alternative to the basic voting approaches is **weighted voting**. Each vote gets a weight based on the system which cast it. The weight can be for example the accuracy of the single system (Zeman and Žabokrtský 2005; Sagae and Lavie 2006), or its accuracy value with respect to a specific condition (Surdeanu and Manning 2010; Sagae and Lavie 2006), e.g. the part-of-speech tag of the dependent, the dependency label, the length of the dependency or the sentence length.

Sagae and Lavie (2006) use the weights together with their reparsing algorithms which guarantee a well-formed tree structure as output. For the unlabelled dependency setting they build a graph of the tokens and weighted directed edges. The edges reflect the decisions of the parsers and a related weight. When several parsers propose the same edge, their weight is added. The maximum spanning tree covers the optimal dependency structure based on the set of weighted dependencies, by maximising the votes for the dependencies while the result is still a tree. They apply the accuracy of the parser and the accuracy based on the part-of-speech tag as weights.

In the labelled constituent setting they add weights of constituents with identical span and label and use them in a weighted parse chart. An exhaustive search maximizes the weights of the constituents which can be combined in a well-formed structure. Alternatively they employ a threshold which omits all constituents with a lower weight, and they tune the threshold on held out data e.g. for high F-score. As weights, they apply the precision of the parsers for the constituent labels.

The last method discussed in this respect can be seen as **scoring**. Henderson and Brill (1999) propose to decide on a full parse based on the number of constituents it has in common with the other parses. Thus a system votes for a parse with the number of constituents its result and the parse have in common. That way each parse receives a score and the parse with the highest score is the single result. Within this method they decide to break ties arbitrarily, which is another option to deal with them.

Classifiers

Combination approaches can also involve classifier methods. When the parsers themselves are seen as classifiers, these methods are often called (classifier) **stacking** (Zeman and Žabokrtský 2005; McDonald and Nivre 2011) or **ensemble models** (Surdeanu and Manning 2010), and the topmost classifier is sometimes called a **meta-classifier** (Surdeanu and Manning 2010).

The actual combination can happen by means of an additional classifier deciding, based on the output of all parsers, which parser to trust in which situation (Henderson and Brill 1999; Zeman and Žabokrtský 2005; Surdeanu and Manning 2010). These methods leave the single systems untouched and allow in principle the inclusion of any kind of parser, i.e. also rule-based ones (Zeman and Žabokrtský 2005).

Alternatively, one of the parsers can be turned into the combination classifier, by taking features or the output from another parser into account when generating its own analyses (McDonald and Nivre 2011; Surdeanu and Manning 2010). These methods of course require detailed familiarity with the internal setup of at least one parser and access to its training routines.

Henderson and Brill (1999) apply a Naïve Bayes classifier for the combination of constituents as well as for the decision between full parses, Zeman and Žabokrtský (2005) take two approaches to classifiers for the combination of substructures from three dependency parses: a memory-based approach with only the part-of-speech tags of head and dependent as features, and a decision tree approach trained on a specific data set and a large set of features. The specific data set consists of the cases where at least one parser was right AND at least one parser was wrong, and the large feature set comprises 12 morphological features (e.g. part-of-speech, gender, number, case, tense, etc.) and 4 semantic features (e.g. the labels proper-name and geography) for head and dependent, features on the relation of head and dependent (mutual position), and a binary feature on the relation of each parser pair, i.e. agreement or disagreement on a specific dependency relation. Surdeanu and Manning (2010) combine candidate dependencies based on a medium size set of features (and their combinations): the part-of-speech tags of head and dependent, the dependency label, the lengths of the dependency and the sentence, as well as an identifier for the system which provided the respective features.

Voting or classifiers?

Henderson and Brill (1999) report that when combing constituents, their voting approach performs equal to the classifier (precision: 92.42, recall: 90.10, F-score: 91.25). Only for the decision between full parses, the classifier improves over the voting in precision (90.78 vs. 90.04) and F-score (90.74 vs. 90.43), with a similar recall (90.70 vs. 90.81). Zeman and Žabokrtský (2005) do not report the outcome of all possible combinations of the participating dependency parsers, but they state that the classifier approach, which they used for three systems, had only a very small accuracy improvement over the respective voting combination. They report the accuracy for two cases: 86.3 vs. 86.2 for the memory-based classifier on the best, second best and fourth best single system vs. accuracy-weighted voting of the same systems; and 86.9 vs. 86.7 for the decision tree classifier on the best, second best and third best single system vs. accuracy-weighted voting of the same systems. Additionally, their conclusion states that the combination

of the four best parsers in an accuracy-weighted voting resulted in an accuracy of 87.00.

Zeman and Žabokrtský (2005) set their findings in context with the outcome of Henderson and Brill (1999), especially because they use different parsers and treebanks (regarding language: Czech vs. English, and structure: dependency vs. constituency); both find that simple voting is already a powerful approach in combination. This finding is also supported by the comparisons from Surdeanu and Manning (2010).

The case studies in Chapter 6 will show that voting is also easy to adapt to task-based methods, and thus straightforwardly applicable for the users of the task-based parser output combination pipeline set up in this work. Furthermore the case studies of this work will introduce rule-based combination as an additional combination method.

2.3 Task-based approaches

This last section switches the focus from combination to another important aspect of this work: taking the task into account.

The task of Katz-Brown et al. (2011) is machine translation, especially translations including the need for major changes in word order. They thus experiment with translations from English into Subject-Object-Verb languages such as Japanese. Thereby the input sentences are first reordered and then translated in a second step. Since reordering requires syntactic analysis, a parser is applied, and they train the parser in what they call a **targeted self-training** approach. The (baseline) parser is trained on newspaper data, and the training data is then enhanced by analyses of the baseline parser on a set of web data. Regular self-training includes the best-ranked output into the training set, however in their approach of targeted self-training they select a parse from the n-best list, which is best for reordering. When testing on a web data test set, they find that in comparison to the news-trained baseline parser and the parser applying regular self-training, the parser with targeted self-training improves the reordering step, as well as the quality of the overall translation (regarding

BLEU scores[34] and human evaluation), while the results for the parsing as such decrease as shown in an intrinsic evaluation of attachment scores.

This result supports the outline of this work, based on the fact, that the best overall parse is not necessarily the best parse for a specific task (cf. Section 1.3). Similarly, they also create small task-based sets with manual annotations to be able to rerank the n-best list of the parser according to the task, stating that a respective task-based data set requires less effort than a new extensive treebank.

Another relevant aspect is that they experiment with a dependency and a constituency parser. However, although they hint at the possibility of combination, for their experiments, they only take them separately into account.

Xu et al. (2011) apply a parser within a pipeline for relation extraction, i.e. being able to extract parts of a text which are connected by a specific relation. The data they use contains English news documents on Nobel Prize awards and several arguments are extracted for a relation, e.g. name of the winner, name of the prize, area of the prize and the respective year in which the prize was awarded.

They apply a manually created (HPSG) grammar and a ranker, the latter being trained on an HPSG treebank. Their parsing system thus includes a rule-based and data-driven part. The system they apply for relation extraction learns extraction rules in a bootstrapping approach: starting from a 'seed', i.e. some examples for the relation to be extracted, rules are learned and utilized to process further text and extract relation candidates and new rules. Newly found examples of the relation are added to the seed for the next iteration until neither new rules nor new candidates are found. Additionally, they compute confidence values for the rules, based among others on the candidates extracted by the respective rule. Since the rules are based on information from the parser output, these confidence values are also applied in the ranker of the parsing step: based on the quality of the rules related to a specific reading, its score is increased or decreased.

Based on a comparison of experiments including and not including reranking, they conclude that recall was increased by means of the reranking and also F-scores were increased, even if there was some drop in precision values.

[34] Papineni et al. (2002)

However, they also find that the highest ranked parsers, i.e. the parses best suited for their task of relation extraction, are not necessarily the best parses as such.

Similar to this work they state that with the task-specific processing, a full disambiguation is not necessary in all cases (cf. Chapter 5), and they also only need few additional data for the setup of their task-specific processing, in their case the seeds, which are also needed in the bootstrapping of their relation extraction system. However, like Katz-Brown et al. (2011), they focus on reranking. In the case studies for Task II of this work (Section 6.2), an n-best list of BitPar is applied, and one case study also selects specific BitPar readings which are not ranked highest; however, the case studies do not focus on reranking as such.

In the following we will combine the benefits of the approaches of combination and task-based processing discussed in this chapter, to the concept of task-based parser output combination. Chapter 3 paves the way to the combination by discussing interoperability aspects.

Chapter 3

Interoperability of resources

As stated in Section 1.3, combining two different syntactic analyses means that they have to show a certain degree of interoperability. One main point in the output combination approach therefore is finding a balance between needing different analyses to benefit from different error distributions and having to be able to combine them. So the actual question is: How different are our initial analyses allowed to be in order to still be interoperable?

With respect to task-based parser output combination, the interoperability of different syntactic annotations in the horizontal dimension (cf. Section 1.2.5: horizontal analysis relations) and the interoperability of these annotations with subsequent processing steps in the vertical dimension (cf. Section 1.2.5: vertical analysis relations) are of main concern.

In this chapter we discuss the notion of interoperability and split it into different aspects applicable to most linguistic resources. This division into **representational interoperability**, **syntactic interoperability** and **semantic interoperability**, which we introduce, provides us with a clearer picture as to where the actual differences between two analyses lie, and will therefore allow us to apply different strategies to increase the interoperability of the analyses. As a consequence, we will be able to combine very different analyses in the task-based approach.

The conceptual work on interoperability discussed here is however not restricted to the interoperability of syntactic annotations. Thus in this chapter we

will take resources in general into account, while the focus and many of the examples are on the interoperability of different parser output.

Section 1.2.1 describes the broad notion of linguistic resources applied throughout this work. Creating a new resource is usually costly with respect to work expended, time, and amount of data to be processed. It might involve finding participants for studies, paying licence fees for primary data or software, and utilizing special equipment. Thus sustainability has become an important aspect when creating new resources or curating existing ones. The possibility to effectively reuse existing resources heavily depends on two factors: the documentation of a resource and its interoperability.[1] A thorough documentation helps potential users to decide if an available resource fits their needs and it describes how the resource can be applied[2]. The interoperability of a resource depicts its ability to be used together with other resources and thus may restrict the set of possible applications or may allow for various application procedures.

Many computational linguistic approaches deal with interoperability. Some explicitly focus on interoperability as a concept (e.g. Witt et al. 2009; Ide and Pustejovsky 2010; Stede and Huang 2012), some intend to increase the interoperability of different (kinds of) resources (e.g. ISO 24612:2012; ISO 12620:2009; Zipser and Romary 2010; Tsarfaty et al. 2012; Nivre et al. 2016), and some have to deal with interoperability of resources as a part of their actual approach (e.g. de la Clergerie et al. 2008; Hinrichs et al. 2010; Buchholz and Marsi 2006; Chiarcos et al. 2012).

Section 3.1 inspects how the notion of interoperability has been discussed by different authors in theoretical approaches. Sections 3.2 and 3.3 present refined categories of interoperability and a respective classification of combination types as a contribution of this work. Section 3.4 discusses other practical approaches which have to deal with interoperability aspects and Section 3.5 discusses the

[1] Of course there are further aspects involved, e.g. the possibility to find out that a resource exists, potential licence restrictions, etc.

[2] See also the introduction to metadata and process metadata in Sections 1.2.3 and 1.2.4.

handling of interoperability with respect to the combination approach of this work.

The approach to interoperability of resources presented in this chapter has been published in Eckart and Heid (2014) in joint work with Ulrich Heid.

3.1 Notions of interoperability

Interoperability of language resources has been discussed in various approaches which focus on different aspects of interoperability and define the concept of interoperability in slightly different ways. Witt et al. (2009) apply a general definition, stating that the most general notion of interoperability of language resources conveys the idea that these resources are able to interact with each other. Ide and Pustejovsky (2010) define interoperability as a measure for the degree to which resources are able to work together and thus aim at an operational definition of interoperability. Stede and Huang (2012) make use of a more methodological definition taking especially the interoperability of linguistic annotations and the process of creating annotation guidelines into account.

Consequently, based on the different definitions, these approaches classify scenarios of interoperability in a different fashion. Witt et al. (2009) classify such scenarios by the types of resources to be combined, examples are (i) applying tools to a corpus vs. (ii) combining corpora to create a common subset. Ide and Pustejovsky (2010) describe conditions for interoperability classified by thematic areas: metadata, data categories, publication of resources and software sharing. Additionally they distinguish between syntactic interoperability and semantic interoperability, adopting these notions from the study of interoperability of software systems and adapting them to the field of computational linguistics. According to them, syntactic interoperability is characterized by properties which ensure that different systems are able to exchange data and to process them either without any conversion or including only a trivial conversion step;

while semantic interoperability is the capability to interpret the data in an informed and consistent way.[3]

Witt et al. (2009) also highlight the difference between (i) a transfer philosophy of interoperability, where a mapping of the information of one resource to the format and, if necessary, the framework of the other resource is applied, and (ii) an interlingua philosophy of interoperability, where data from both resources are mapped to a new representation which generalizes over both.[4] Accordingly, Stede and Huang (2012) discuss the role of standard formats for interoperability in an interlingua approach.

In this work, we adopt the general definition of Witt et al. (2009), which defines interoperability of resources as the ability for these resources to interact, work together or be combined. The approach also distinguishes between representational and content-related aspects, as Ide and Pustejovsky (2010) do, but we will introduce an additional classification on the content side. Thus our definition of syntactic and semantic interoperability is slightly different from theirs. Like Stede and Huang (2012) we will in particular take the aspect of the combination of linguistic annotations into account.

3.2 Refined categories of interoperability

We propose a refined concept of interoperability. Like Ide and Pustejovsky (2010) we make a distinction between representation-related and content-related interoperability. However, for content-related interoperability we make an additional distinction between syntactic and semantic interoperability.

Representational interoperability focuses on the different possibilities of representation, i.e. encodings of information. For example, syntactic information is usually structured as a tree, but this tree can be represented by the introduction

[3] Ide and Pustejovsky's categories are referred to as structural and conceptual interoperability by Chiarcos (2012), focussing conceptual interoperability even more on vocabularies.

[4] In this context, Witt et al. (2009) include formatting aspects as well as aspects of linguistic classification in the discussion of representations. Section 3.2 applies different terms for these aspects, e.g. focussing representational interoperability on aspects which are independent of linguistic decisions.

of brackets to the original input, or it can be encoded in an XML representation, embedded in a graph visualization or arranged in a tabular format. Figure 3.1 shows three different representations of exactly the same content produced as output by BitPar for the sentence in Example (3.1).

(3.1) *Er liest ein Buch.*
He reads a book.

Note the difference between this example and the example from Figure 1.1 on Page 22 in Chapter 1. In Figure 1.1 three different parsers were applied to the sentence, thus the analyses differ with respect to their linguistic content, in Figure 3.1 the same parser was applied in all cases, but three different representations of the same output are shown.

With respect to linguistic information, i.e. data categories and their structured combination, the three analyses in Figure 3.1 are identical – each of them encodes the same phrase structure tree based on the same grammar and tagsets. Yet, at first sight, it is hard to even see if they are similar. Figure 3.1a is an inline representation of the annotation, where linguistic information is introduced into the original sentence by means of brackets (structure) and tags (part-of-speech, syntactic and morphological information), similar to a well-known representation format of the Penn Treebank (Marcus et al. 1993). Here the opening bracket followed by NP-OA denotes the start of the noun phrase *ein Buch*, which is the direct object of the sentence. Exactly the same linguistic information is represented differently in Figure 3.1c. There, we see a graphical representation of the annotated linguistic structure of the sentence.[5] No brackets are applied, but two edges connect the node labelled NP-OA to its children, the parts of the noun phrase. Figure 3.1b is an XML stand-off representation of the annotation as an excerpt of the TCF format (Heid et al. 2010). Here the output of BitPar is represented in its own layer (<parse/>), i.e. separated from the actual tokens (<tokens/>).

While the examples in Figure 3.1 show how difficult a manual comparison will be, also an automatic comparison of the output would involve either

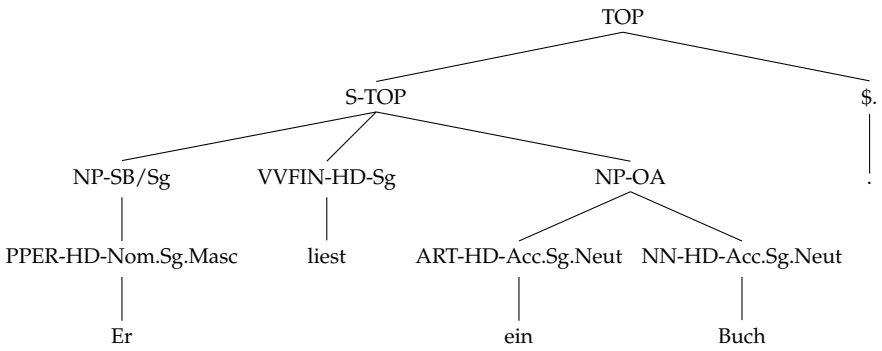
[5] A graphical representation for BitPar output can for example be created by VPF: `http://www.ims.uni-stuttgart.de/forschung/ressourcen/werkzeuge/vpf.html`

```
(TOP (S-TOP (NP-SB/Sg (PPER-HD-
Nom.Sg.Masc Er))(VVFIN-HD-Sg liest
)(NP-OA (ART-HD-Acc.Sg.Neut ein)(
NN-HD-Acc.Sg.Neut Buch)))(\$. .))
```

(a) Inline bracketing

```
[...]
<tokens>
  <token ID="t1">Er</token>
  <token ID="t2">liest</token>
  <token ID="t3">ein</token>
  <token ID="t4">Buch</token>
  <token ID="t5">.</token>
</tokens>
[...]
  <parse>
    <constituent cat="TOP">
      <constituent cat="S-TOP">
        <constituent cat="NP-SB/Sg">
          <constituent cat="PPER-HD-Nom.Sg.Masc"
                tokenIDs="t1"></constituent>
        </constituent>
        <constituent cat="VVFIN-HD-Sg"
              tokenIDs="t2"></constituent>
        <constituent cat="NP-OA">
          <constituent cat="ART-HD-Acc.Sg.Neut"
                tokenIDs="t3"></constituent>
          <constituent cat="NN-HD-Acc.Sg.Neut"
                tokenIDs="t4"></constituent>
        </constituent>
      </constituent>
      <constituent cat="\$." tokenIDs="t5"></constituent>
    </constituent>
  </parse>
[...]
```

(b) XML: excerpt of the TCF format



(c) Tree visualization

Figure 3.1: Three representations of the same linguistic content.

thorough investigation or several conversion procedures. Thus we claim that representational interoperability is often the first step towards a general goal of resource interoperability and that it should not be confused with the linguistically motivated structural decisions reflected in the content (e.g. 'deep' vs. 'flat' analyses, see the discussion on Figure 3.2 below). Especially these content-related structural decisions should not get mingled with representational aspects in the process of comparison or conversion.

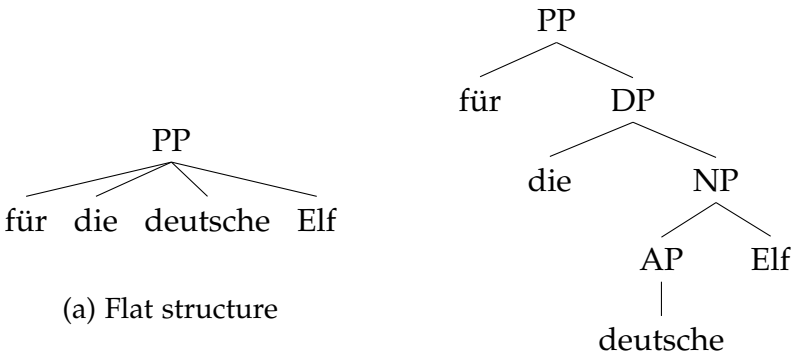
Content-related interoperability comprises all linguistically motivated decisions. Here we introduce an additional distinction between syntactically and semantically motivated differences.[6]

Syntactic interoperability takes structural decisions into account and evaluates the similarity of the underlying models: Is the information based on a tree model, i.e. including hierarchical categories and a single father constraint[7], or can the intended correlations only be covered by a directed acyclic graph? Is a node in the tree allowed to have more or less than two children? Are the correlations labelled? To highlight the difference from representational interoperability: In the latter case, the question of where the labels are attached, i.e. to nodes or to edges, would be a representational question; the question important for syntactic interoperability is if correlations are at all intended to include additional information.

On a high level, differences with regard to structural interoperability for example also include the differences between phrase structure and dependency trees. While in dependency trees a token is directly connected to its head, phrase structure trees introduce additional nodes for each phrase. Another important distinction can be made between 'flat' and 'deep' structures. In Section 2.1.4 several analyses for the same example sentences are given. Figure 3.2 shows two phrase structure analyses for the phrase *für die deutsche*

[6] 'Syntactic' and 'semantic' does not refer to the linguistic description levels here but to the distinction between structure and categories within any layer of annotation.

[7] Each node in the structure but the root node has exactly one father node. See also Section 2.1.2 for a definition.



(a) Flat structure

(b) Deep structure

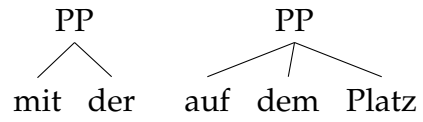
Figure 3.2: Aspects of syntactic interoperability: flat vs. deep structure.

Elf ('for the German football team') from the examples in Section 2.1.4.[8] Figure 3.2a applies a flat structure consisting only of a prepositional phrase (PP). In Figure 3.2b three more phrases are embedded: a determiner phrase (DP) *die deutsche Elf*, a noun phrase (NP) *deutsche Elf* and an adjective phrase (AP) *deutsche*.

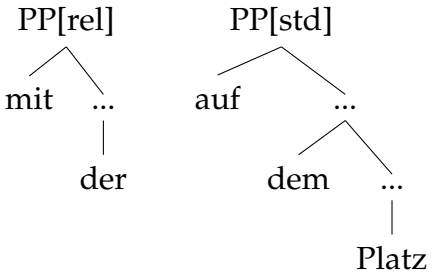
Semantic interoperability focuses on the concepts which are applied within the resources. These are often modelled by means of a tagset, where every tag stands for a concept with which parts of the resource can be labelled. A typical example is part-of-speech tagging, where categories such as NOUN, VERB or PRONOUN are attached to words or word combinations[9]. Distinctions regarding semantic aspects can be found in the annotation guidelines and in the coverage of the single concepts. In the simplest case two different names are applied for the same concept, e.g. NN or N[comm] for common nouns. More difficulties arise when the same name is applied for different concepts, e.g. when different approaches to dependency syntax use the term HEAD either to refer to a lexical or to a functional head. A further issue is granularity, i.e. cases where a specific concept is applied in one resource, while it is split into several concepts in

[8] Parts-of-speech, category details and unary chains are omitted for readability here, for the full analyses see Figures 2.21 and 2.25 on Pages 91 and 97, respectively.

[9] Cf. Page 54 for the use of "word" in this work.



(a) Coarse-grained tagset



(b) Fine-grained tagset

Figure 3.3: Aspects of semantic interoperability: tagset granularity.

another one, cf. Figure 3.3a, where the IMS-SZEGED-CIS parser applies the tag PP to both prepositional phrases in the sentence from Example (3.2)[10], while in Figure 3.3b the LFG parser distinguishes the prepositional phrase with the relative pronoun (PP[rel]) from a standard case (PP[std]).[11] . The hardest case is one where two concepts only cover part of each other, and no mapping scheme can be applied.

(3.2) *Es ist Marcel Hellers Schnelligkeit, mit der auf dem Platz*
it is Marcel Heller's speed with which on the field
nur wenige mithalten können.
only few keep up can
'It is Marcel Heller's speed, which only few can keep up with on the field.'

Another aspect which can be applied to the syntactic and the semantic view alike is the ability to include underspecification. For an example of how to deal

[10]Repeated from Example (2.9), Page 70.

[11]Parts-of-speech, some category names and unary chains are omitted for readability here, for the full analyses see Figures 2.20 and 2.24 on Pages 90 and 96, respectively.

with two resources where one applies underspecification and the other does not see Section 6.1, for a short discussion on the relation of underspecification with the combination workflow see Section 5.3.

Figure 3.4 summarizes the refined categories of interoperability.

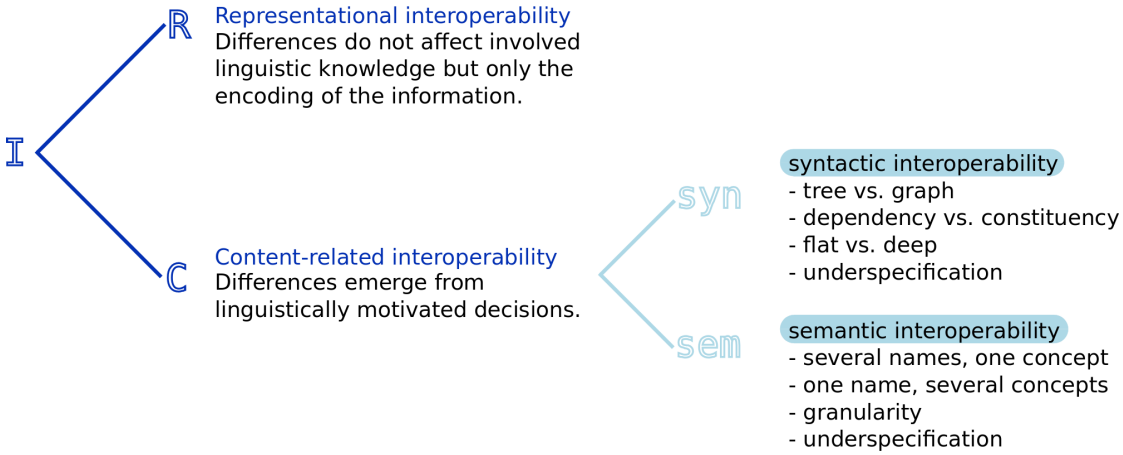


Figure 3.4: Overview of refined categories of interoperability.

When aiming at interoperability of resources, we need to evaluate their representational closeness, their syntactic closeness and their semantic closeness. Even if these aspects are often interrelated, two resources might show discrepancies to a different degree with respect to each of these categories.

Representational interoperability can mostly be achieved by some sort of conversion step which either adapts the representation of one resource to the requirements of another one (transfer philosophy) or converts both resources into a common and, if necessary, more generic representation (interlingua). While with the transfer philosophy information loss can be accepted with regard to an explicit target resource, an interlingua approach usually opts for lossless conversion of both resource representations. In any case, content-related interoperability is usually much harder to achieve, and is more often than not outside of the scope of a mapping.

Taking the separation of representational and content-related aspects into account, it is easier to assess what is the most beneficial way for researchers or projects to invest work into achieving resource interoperability.

3.3 Classification of combination types

Since content-related interoperability is the much harder part in handling resources, we additionally introduce a classification for content-related interoperability from the resource-side rather than from the theoretical viewpoint.

An important use case for the assessment of interoperability arises in situations where different resources are to be combined (in a horizontal or a vertical way). In the following we introduce a classification of combination types for such resources, regarding content-related interoperability. Combination type 1 is the case with the highest degree of interoperability, while combination type 3 is a case where neither syntactic nor semantic interoperability are given.

Combination type 1 applies, when two resources are based on the same concepts and the same structural decisions. In this case the resources are fully interoperable with respect to content-related aspects. Examples are different development versions of the same resource or a set of systems taking part in a shared task, where all systems are trained on the same training data.

Combination type 2 applies, when two resources are similar with respect to their structure, but differ with respect to their concepts. Thus semantic interoperability has to be provided while the resources are already syntactically interoperable. Examples are different part-of-speech taggers which can be applied to the same tokenization; or two lexical resources with a word-based structure but different annotations; or a dependency parser trained on different training sets providing for a similar structure with respect to aspects such as projectivity[12], head-type and coordination.

[12]Cf. Section 2.1.3.

Combination type 3 applies, when resources differ in structural as well as in semantic criteria. Examples are the combination of several independent annotation layers, e.g. prosodic and semantic annotations[13] based on different segmentations of the primary data; or a query tool for dependency treebanks and a corpus annotated with constituency trees; or a labelled word net and a classical lexicon.

In this chapter we utilize our refined classification to localize theoretical as well as practical approaches. In Chapter 6 we will make use of the combination types to classify our experiments.

3.4 Existing approaches

After already locating some theoretical approaches to interoperability within our refined categories of interoperability (cf. Sections 3.1 and 3.2), we will now look at some applicatory realizations. We will start with some approaches which have to deal with interoperability of resources as a part of their actual project. Afterwards we will discuss approaches which explicitly intend to increase the interoperability of different (kinds of) resources and where our approach is in accordance with some of them, and where we draw different conclusions for our work.

Shared tasks. Shared tasks are usually set up to foster the creation and to enhance and evaluate the quality of language processing systems for a specific task such as named entity recognition, dependency parsing or machine translation. Thereby, they happen to be also a platform for the creation of interoperable resources with regard to horizontal relations. In a typical shared task, a certain amount of data is made available and shows the targeted input/output combination. This material can be used to statistically train or otherwise build a system to produce high-value output with respect to the theory or setting the material is based on. At a specific point in time, test data is released, which

[13] Here, 'semantic annotations' refers to the linguistic description level, whereas 'semantic criteria' in the preceding sentence refers to the conceptual aspects of interoperability.

is processed by the participating systems, and their output is evaluated and ranked by specific metrics. Thus a set of systems emerges, where each system is able to handle the same input data and is aiming to produce the same output information, including the same structure and tagset. These systems are thus possible candidates for an easy combination on the horizontal level.

chunks		**dependencies**	
GN	noun phrase	SUJ-V	subject
GP	prepositional phrase	AUX-V	auxiliary
NV	verbal nucleus	ATB-SO	attribute-subject/object
GA	adjective phrase	COD-V	direct object
GR	adverb phrase	CPL-V	verb complement
PV	verb phrase with preposition	MOD-V	verb modifier
		COMP	complementizer
		MOD-N	noun modifier
		MOD-A	adjective modifier
		MOD-R	adverb modifier
		MOD-P	preposition modifier
		COORD	coordination
		APP	apposition
		JUXT	juxtaposition

Table 3.1: Categories for syntactic annotation applied in the PASSAGE project (Vilnat et al. 2010; PASSAGE-L1 2009).

The project PASSAGE (de la Clergerie et al. 2008), invited parsing systems for French to take part in a collaborative annotation approach of textual data from various sources, including oral transcriptions. The goal was to create a valuable and comprehensive corpus resource for French, by combining the output of different parsing systems in a bootstrapping approach. To be able to combine and merge the annotations, a rather abstract set of categories was defined on which all participating systems could agree. This category set comprised six categories of chunks and fourteen categories of dependencies (cf. Table 3.1 and Vilnat et al. 2010). On the one hand, this setting brought up an actual use case, where interoperable systems on the same horizontal level were combined to create a new resource. On the other hand, this interoperability was achieved at the cost of abstracting over the content-related differences of the systems, which

precisely include the most valuable information in combination approaches. Thus this relevant information gets lost in the abstraction.

A similar argumentation applies for the shared tasks regularly conducted in conjunction with the *Conference on Natural Language Learning* (CoNLL). In 2006 and 2009 the task was on dependency parsing for different languages (Buchholz and Marsi 2006; Hajič et al. 2009). There, the content-related specifications of the system output were not based on the least common denominator like in PASSAGE, but predetermined by the chosen data set for each language. While this allows for more detailed analyses, it still excludes the need for a combination of different content-related aspects.

However, the CoNLL shared tasks address content-related interoperability in some other respects. Firstly, since the expected output does not only comprise dependency information but also part-of-speech tagging, lemmatization and the identification of morpho-syntactic features, the approach thus also fosters interoperability for vertical analysis relations. And secondly, the setup leads to systems which are applicable to many languages. Thereby a language-independent and thus interoperable workflow of training and testing procedures has emerged. Additionally the CoNLL shared tasks gave rise to tabular annotation representations, which have become a de-facto-standard in the field. They thus provide for increasing interoperability on the representational level in horizontal as well as vertical approaches.

Processing chains. Processing chains usually implement one path of vertical analysis relations, e.g., starting from the tokenization of primary data and leading up to syntactic and semantic annotations and probably data extraction procedures. Frameworks which implement processing chains are for example UIMA[14] and GATE[15].

A platform for processing chains set up in the context of the CLARIN project[16] is WebLicht[17] (Hinrichs et al. 2010). WebLicht lists a set of web services

[14] `http://uima.apache.org/`

[15] `http://gate.ac.uk/`

[16] `http://www.clarin.eu/`

[17] Web-based Linguistic Chaining Tool,
`http://weblicht.sfs.uni-tuebingen.de/weblichtwiki/index.php/Main_Page`

from which users can build a chain to process some input data. Each web service encodes a natural language processing tool in a wrapper. The output of one web service constitutes the input for another one, until the required annotation level is reached. Thus the processing chain has to deal with three levels of formats: the original input and output format of the underlying tool, the processing format applied to exchange information between the web services, and, if applicable, an additional output format at the end of the processing chain (cf. Heid et al. 2010). In this setting the wrapper ensures content-related interoperability to a certain extent by the way the original tool formats are mapped to the exchange format. Among the different wrappers, representational interoperability is ensured by means of the common processing format which needs to strike a balance between the need for a detailed set of linguistic annotations, and the processing efficiency typically required in a web-based approach.

Linked data. Linked data refers to a concept which provides connections between resources and is set up related to the Semantic Web. Since linked data is thus usually accessible via the web, the topic is closely related to the discussion on open data. While linked data does not only refer to linguistic resources, the concept allows to connect for example corpora with lexicons, word nets or related resources and thus the Open Linguistics Working Group[18] (Chiarcos et al. 2012) has been established within the Open Knowledge Foundation[19]. The connected resources build a network, called a (Linked Open Data) cloud, where the links between the resources can be exploited in joint queries.[20]

To allow for these joint queries, interoperability aspects have to be handled. Representational interoperability is usually achieved by applying the Resource Description Framework (RDF)[21], which is common within the Semantic Web. By means of RDF parts of information and relations between these parts are represented as triples of subject, predicate and object, and information can be

[18] `https://linguistics.okfn.org/`
[19] `https://okfn.org/`
[20] For the Linguistic Linked Open Data cloud see: `http://linguistic-lod.org/llod-cloud`
[21] `https://www.w3.org/TR/rdf11-concepts/`

expressed by means of resource identifiers which link to a representation of the respective data point. Overall the triples build graphs of linked information. Semantic interoperability is handled e.g. by means of OWL[22] ontologies. Queries can be formulated by means of SPARQL[23].

Standardization and converter frameworks. Stede and Huang (2012) observe that standard formats play an important role in interoperability and tend to be applied as a pivot representation in an interlingua approach, to exchange data between more resource-specific formats without loosing information in the process of mapping. One of these generic exchange formats is GrAF (Ide and Suderman 2007, 2014), the serialization of the Linguistic Annotation Framework LAF (ISO 24612:2012). LAF introduces a layered graph structure, where graphs consist of nodes, edges and annotations. The annotations implement the full power of feature structures and can be applied to nodes and edges alike. All standard annotation layers for linguistic corpora can be mapped onto this model, and since references to the primary data are implemented based on the encoding of their minimal addressable unit, such as characters for a textual representation, or frames for video data, several modalities are covered. LAF/GrAF does thus provide for representational interoperability when several analyses are encoded based on the LAF data model.

Figure 3.5 visualizes a possibility for minimal addressable units of primary data for the example in Figure 3.1. The vertical bars between the characters symbolize what LAF calls virtual anchors and represent a position in the primary data.[24] By means of referring to these anchors, regions in the primary data can be identified.

Representing each of the three analyses in Figure 3.1 in GrAF produces an identical result for each of the original representations, see Figure 3.6. While the generic format is more verbose, it reduces the comparison cost for the analyses to a minimum.

[22] Web Ontology Language: OWL `https://www.w3.org/TR/owl-ref/`
OWL 2 `https://www.w3.org/TR/owl2-overview/`

[23] `https://www.w3.org/TR/sparql11-query/`

[24] The real position numbers would however depend on the actual document in which the sentence appears.

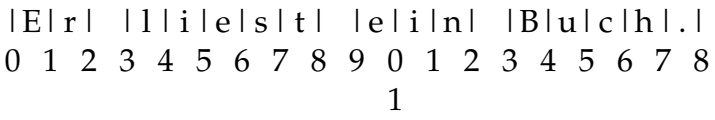


Figure 3.5: Visualisation of LAF's virtual anchors.

Of course, in a typical setting where resources should be combined, the resource annotations are not identical. However, mapping them onto a common representation which is guaranteed to still reflect all resource-specific annotation decisions, helps to bring out the actual content-related differences. To some extent LAF/GrAF can also be used to abstract over aspects which we relate to syntactic interoperability, e.g. like condensing annotations from a non-branching path[25] into a combined edge label, or to minor aspects which we relate to semantic interoperability, e.g. splitting annotations into more detailed feature structures as in Figure 3.7[26], which requires some interpretation of the original node label.

However, by design, LAF itself does not handle semantic interoperability but provides a mechanism for annotation items to be linked to external concept definitions. Such concept definitions could be set up and can be referred to in ISOcat,[27] a Data Category Registry, based on ISO 12620:2009[28]. There, concept definitions were entered in a grass roots approach by the community: if the concept which is needed for a specific resource is not available, it can be entered to the registry. To take care of uncontrolled growth, which might result from the grass roots approach, thematic domain groups were supposed to select and recommend specific concepts relevant to thematic domains such as metadata, lexicography, morpho-syntax or sign language. Data Category

[25]Such a non-branching path is e.g. called a unary chain in parsing results.

[26]Actually, both annotations can be available at the same time: Nodes, edges and annotation are on the same level of the XML structure, i.e. not embedded within each other and reference is made by explicit identifiers. Thus, several annotations can refer to the same node or edge.

[27]`http://www.isocat.org/`
In August 2017 the status of ISOcat is static, i.e. no new entries can be added. The data categories are however still accessible by means of their persistent identifier.

[28]This last published version of ISO 12620 is currently withdrawn but an update is expected.

```
[...]
  <!-- regions in primary data document -->
  <region xml:id="r1" anchors="0␣2"/>
  <region xml:id="r2" anchors="3␣8"/>
  <region xml:id="r3" anchors="9␣12"/>
  <region xml:id="r4" anchors="13␣17"/>
  <region xml:id="r5" anchors="17␣18"/>
[...]
  <!-- leaf nodes referring to regions -->
  <node xml:id="n1">
    <link targets="r1"/>
  </node>
  <node xml:id="n2">
    <link targets="r2"/>
  </node>
  <node xml:id="n3">
    <link targets="r3"/>
  </node>
  <node xml:id="n4">
    <link targets="r4"/>
  </node>
  <node xml:id="n5">
    <link targets="r5"/>
  </node>
  <!-- inner nodes -->
  <node xml:id="n6"/>
  <node xml:id="n7"/>
  <node xml:id="n8"/>
  <node xml:id="n9"/>
  <node xml:id="n10"/>
  <node xml:id="n11"/>
  <node xml:id="n12"/>
  <node xml:id="n13"/>
  <!-- root node -->
  <node xml:id="n14"/>
  <!-- edges -->
  <edge xml:id="e1" from="n14" to="n13"/>
  <edge xml:id="e2" from="n14" to="n10"/>
  <edge xml:id="e3" from="n13" to="n11"/>
  <edge xml:id="e4" from="n13" to="n7"/>
  <edge xml:id="e5" from="n13" to="n12"/>
  <edge xml:id="e6" from="n10" to="n5"/>
  <edge xml:id="e7" from="n11" to="n6"/>
  <edge xml:id="e8" from="n7" to="n2"/>
  <edge xml:id="e9" from="n12" to="n8"/>
  <edge xml:id="e10" from="n12" to="n9"/>
  <edge xml:id="e11" from="n6" to="n1"/>
  <edge xml:id="e12" from="n8" to="n3"/>
  <edge xml:id="e13" from="n9" to="n4"/>
  <!-- annotations -->
  <a label="syn" ref="n6">
    <fs>
      <f name="cat" value="PPER-HD-Nom.Sg.Masc"/>
    <fs>
  </a>
  <a label="syn" ref="n7">
    <fs>
      <f name="cat" value="VVFIN-HD-Sg"/>
    <fs>
  </a>
  <a label="syn" ref="n8">
    <fs>
      <f name="cat" value="ART-HD-Acc.Sg.Neut"/>
    <fs>
  </a>
  <a label="syn" ref="n9">
    <fs>
      <f name="cat" value="NN-HD-Acc.Sg.Neut"/>
    <fs>
  </a>
  <a label="syn" ref="n10">
    <fs>
      <f name="cat" value="\$."/>
    <fs>
  </a>
  <a label="syn" ref="n11">
    <fs>
      <f name="cat" value="NP-SB/Sg"/>
    <fs>
  </a>
  <a label="syn" ref="n12">
    <fs>
      <f name="cat" value="NP-OA"/>
    <fs>
  </a>
  <a label="syn" ref="n13">
    <fs>
      <f name="cat" value="S-TOP"/>
    <fs>
  </a>
  <a label="syn" ref="n14">
    <fs>
      <f name="cat" value="TOP"/>
    <fs>
  </a>
[...]
```

Figure 3.6: GrAF representation excerpt of the examples in Figure 3.1; region anchors referring to virtual anchors in Figure 3.5.

```
<a label="morphosyn" ref="n12">
  <fs>
    <f name="cat" value="NP"/>
    <f name="func" value="OA"/>
  <fs>
</a>
<a label="morphosyn" ref="n8">
  <fs>
    <f name="pos" value="ART"/>
    <f name="func" value="HD"/>
    <f name="case" value="Acc"/>
    <f name="num" value="Sg"/>
    <f name="gen" value="Neut"/>
  <fs>
</a>
```

```
<a label="morphosyn" ref="n9">
  <fs>
    <f name="pos" value="NN"/>
    <f name="func" value="HD"/>
    <f name="case" value="Acc"/>
    <f name="num" value="Sg"/>
    <f name="gen" value="Neut"/>
  <fs>
</a>
```

Figure 3.7: Examples for alternative annotations of the accusative object from Figure 3.6 in GrAF representation.

Registries or Concept Registries as such provide most valuable support for semantic interoperability: if two different labels from different resources link to the same concept entry, they can easily be mapped; if two labels with the same name, but links to different concepts exist in the resources, extra care needs to be taken when the respective resources are to be combined.

In addition, frameworks such as SaltNPepper (Zipser and Romary 2010) support conversion from one annotation format into another. Salt, the internal meta model of the Pepper converter framework, handles representational differences, and the system also allows to introduce semantic information by external references to ISOcat data categories.

Evaluation projects. An approach to increase content-related, and specifically syntactic interoperability of parser output is embedded in the evaluation methods described by Tsarfaty et al. (2011) and Tsarfaty et al. (2012). In their approach (multi-)function trees are introduced, to which different parse trees can be mapped. In the actual evaluation, tree edit distance is utilized but does not take edits into account which adhere to theory-specific aspects. In multi-function trees, e.g., unary chains over grammatical functions can be condensed into a single edge with a respective label set, thus increasing the syntactic interoperability of the analyses.

General ontologies and universal tagsets. Another approach to interoperability of resources focuses on finding a common ontology or tagset which the different resources are supposed to use or can be mapped onto. The creation of data categories in ISOcat can be seen as a grass roots approach, which might eventually converge to a set of recommended concepts. GOLD (Farrar and Langendoen 2003), the General Ontology for Linguistic Description and OLiA, the Ontologies of Linguistic Annotation (Chiarcos and Sukhareva 2015) both propose their application as lingua franca and OLiA also implements links to ISOcat and GOLD amongst others. However, while both are related to the Semantic Web, GOLD focuses on linguistic data with one target being the cooperation among linguists rather than applications in natural language processing.

Such ontologies are helpful to handle semantic interoperability on a more general basis, but require effort for new tagsets to be mapped onto the respective models, which still may result in information loss at least on a certain representation level.[29]

In any case it is difficult to come to a common solution against the background of many different linguistic theories and frameworks as well as different language groups. Thus many approaches to common annotation semantics take a more practical viewpoint, with the goal to make different resources work together.

OLiA partially includes such a practical focus by providing Linking Models from theoretical recommendations as well as existing tools and tagsets. The EAGLES guidelines[30] provide recommendations for applicable categories in several annotation layers, e.g. the "Recommendations for the Morphosyntactic Annotation of Corpora" provide obligatory and recommended attributes and values. However for syntactic annotation they do not impose obligatory annotations but only recommended and optional ones. Approaches towards universal part-of-speech tagsets, e.g., the approach by Petrov et al. (2012), or approaches

[29] OLiA also keeps a model of the original annotation scheme to track lost information or mismatches.

[30] `http://www.ilc.cnr.it/EAGLES/browse.html`

NOUN	nouns
VERB	verbs
ADJ	adjectives
ADV	adverbs
PRON	pronouns
DET	determiners, articles
ADP	prepositions, postpositions
NUM	numerals
CONJ	conjunctions
PRT	particles
.	punctuation marks
X	others, e.g. abbreviations, foreign words, etc.

Table 3.2: Universal part-of-speech tagset proposed by Petrov et al. (2012).

towards a universal treebank such as the Universal Dependencies (McDonald et al. 2013), take an even more pragmatic point of view.

Petrov et al. (2012) based their work on the assumption that across different languages a basic set of coarse-grained part-of-speech tags can be identified. Thus, they present a part-of-speech tagset consisting of twelve tags, cf. Table 3.2, and mappings from 25 different treebank tagsets in 22 languages to their part-of-speech tagset. They focus on the usefulness of their tagset in research, e.g. experiments on grammar induction, and downstream applications, i.e. applications which go beyond the annotation step under consideration.

As in the PASSAGE project mentioned above, the coarse-grained setting smooths out differences, in this case also language-specific differences, and provides thus less information than the original tagsets. It is therefore not surprising that in their evaluation experiment the setting where a part-of-speech tagging model is trained on the language-specific tagset and afterwards mapped to and evaluated on the universal tagset yields higher or equal results to the settings where training and evaluation are either both on the fine-grained or both on the universal tagset. The mixed setting allows for the language-specific

differences to be recognized by the tagger and the mapping to the coarse-grained tagset is likely to conceal confusions in classes which are hard to distinguish.

One other aspect however which is shown in the work of Petrov et al. (2012), but also by Zeman (2008), is that a mapping to common tags does not necessarily mean a common application of annotation guidelines. Petrov et al. (2012) as well as Zeman (2008) state that for phenomena which have not been annotated in a treebank (even if they might exist in the respective language), no mapping to the respective tag can be introduced. Petrov et al. (2012) describe that in the Bulgarian treebank they use, articles and determiners do not have their own category, while their universal tagset does provide such a category. The mapping process does not create new information and thus might produce inconsistencies when resources which have been mapped to a common tagset are combined.

The Universal Dependencies project[31] aims at providing a set of common dependency relations and annotation guidelines for treebanks in different languages. It is based on the Universal Stanford Dependencies (de Marneffe et al. 2014), the universal part-of-speech tagset by Petrov et al. (2012) and the morpho-syntactic interset interlingua by Zeman (2008). Among others, it stands in the history of multi-lingual dependency parsing, such as done in the CoNLL shared tasks (part-of-speech tags and representation format) and the adaptation of the Stanford Dependencies to other languages (dependency relations). Up to now the project has released two major versions of the guidelines and several treebank creators provided datasets in 50 languages (Nivre et al. 2017). While they also take a practical viewpoint and see their approach as beneficial in terms of development of multilingual systems in natural language processing and cross-lingual learning (Nivre et al. 2016), they do not go for plain parser performance, but take the linguistic quality regarding the dependency relations and their benefit for the downstream applications into account (de Marneffe et al. 2014).[32] Thus their universal dependency relations consist of a rather

[31] `http://universaldependencies.org/`

[32] According to de Marneffe et al. (2014) the Universal Stanford Dependencies choose for example content words as heads to support further semantic processing also in cases where functional heads like auxiliary verbs or prepositions seem to be beneficial for parser processing.

fine-grained tagset[33] (37 tags in the Universal Dependencies v2 vs. 14 tags for dependencies in PASSAGE) and they allow for additional language-specific relations as subtypes of existing universal relations.[34] By basing the Universal Dependencies on the Stanford Dependency system, linguistic traditions are followed, due to the Stanford scheme being partly inspired by the LFG framework (McDonald et al. 2013) and originating from grammatical relation-based syntactic traditions (de Marneffe et al. 2014). This makes it clear that the term *Universal* in Universal Dependencies is rather used with respect to different languages than with respect to an abstraction over different linguistic approaches, since basing the approach on the Stanford Dependency system naturally introduces some bias. Utilizing the lexicalist approach in syntax, and going for dependency relations rather than constituency analyses however also accommodates to the practical requirements in computational processing (de Marneffe et al. 2014).

So far, the described universal approaches feature mainly aspects of what we consider the category of semantic interoperability and some syntactic interoperability (orientation of dependency relations). However the Universal Dependencies project also proposes a new version of the tabular CoNLL representation format, called CoNLL-U[35], which among others allows for multiple token spans and multiword tokens.

Just as de Marneffe et al. (2014) see it as "wrong-headed" to make decisions for an annotation scheme based on parser performance instead of linguistic quality and usefulness for further applications, we consider the possibility of combining heterogeneous but task-based information more desirable than to immediately map each annotation to universal categories.

However, in our approaches we only take information from one language into account. Doubtlessly, the Universal Dependencies project is an elaborate approach for any cross-linguistic perspective, starting from strong theoretical

The same goes for representations of long conjunctions with a single head vs. as a chain, where in the latter case short dependencies would rather support parser performance.

[33] `http://universaldependencies.org/u/dep/index.html`

[34] They also slightly extended the universal part-of-speech tagset by Petrov et al. (2012), and allow for language-specific adaptations of morpho-syntactic features.

[35] `http://universaldependencies.org/format.html`

assumptions and grown through many applicatory realizations in tools and treebanks.

3.5 Discussion of handling interoperability for a combination approach

In this chapter we discussed the interoperability of resources. Since the overall approach of this work includes the combination different resources, i.e. different parser output, this discussion is a central point for the development of our general workflow.

Related work on parser combination has been given attention to in Section 2.2. In the current chapter, the focus was broadened to the combination and compatibility of various kinds of resources, as well as at the notion of interoperability in general to motivate and define the scope of our own approach. Related work from the theoretical as well as the application perspective was taken into account.

For the combination approach set up in this work it is important to be able to work with diverse syntactic analyses, in order to maximally profit from the available information when combining resources. As a consequence, some work has to be invested to increase the interoperability of the involved resources without loosing any of the required information. To assess how much work has actually to be done, we defined several subtypes of interoperability as introduced in Section 3.2 and discussed these in detail. On a high level we distinguish between representational interoperability and content-related interoperability, similar to Ide and Pustejovsky (2010). However, we propose an additional refinement on the content side, introducing a distinction between what we call syntactic interoperability and semantic interoperability, not referring to linguistic description levels, but to the formal structure introduced in an annotation scheme and the semantics of the utilized tagset.

Regarding the high level distinction, it is possible and we consider it appropriate to establish full representational interoperability by mapping different

representation formats onto one exchange format. We instantiate this by mapping all parser output to representations within a relational database management system whose data structures are based on LAF/GrAF (ISO 24612:2012), cf. Chapter 4. For content-related interoperability, a general and comprehensive solution cannot be expected to be realizable, due the fact that the resources are based on different linguistic theories and approaches. We claim that however in many cases such a general or comprehensive solution is not needed to reach a sufficient degree of interoperability for the task at hand. Accordingly, we propose a task-based approach to content-related interoperability, which reduces complexity to the task-related aspects and even allows for different combination approaches, depending on the type of task at hand. We introduced a classification of combination types with respect to content-related interoperability in Section 3.3, taking features of the resources involved in a specific setting into account. While for combination type 1 interoperability is easy to achieve, combination type 3 describes the most difficult setting with respect to syntactic and semantic interoperability.

One aspect which is of main importance in our overall workflow is to keep all of the information, and especially the diversity of the information from the output of the different parsers, for our combination approach. Thus, on the representational side we opt for a verbose, generic exchange format, such as GrAF (Ide and Suderman 2007) rather than for an exchange format for efficient processing such as appropriate in processing chains (cf. Heid et al. 2010). On the content-related side we will neither go for a unification of the annotation categories, as in PASSAGE (de la Clergerie et al. 2008) and approaches to universal data categories in general ontologies or universal tagsets, nor for a specific set of output systems as in shared tasks. These approaches would restrict the differences between the information, e.g. when all tools are trained on the same data set and thus also on the same annotation scheme, as it happens in shared tasks; alternatively, it would mean information loss, e.g. by mapping different kinds of output to a specific set of categories or a specific annotation scheme. We will rather take a task-based approach, i.e. focus on the parts of the output which are relevant for the task, and evaluate it on a precision-oriented task and on a recall-oriented task (Chapter 6).

In the following chapters we instantiate the conclusions of this chapter in our combination approach. Starting with the comparatively easy to solve representational aspect, Chapter 4 introduces the relational database system we apply and discusses the design decisions which lead to its generic applicability for different types of resources. Chapter 5 describes the full workflow in a general way, from the first parser inspection to the application of combined syntactic information for a task. In Chapter 6 the workflow is instantiated in two different tasks. Several case studies show the effect the output combination workflow has on the overall result of the respective task. Since these case studies are classified according to the combination types introduced in this chapter, we are then able to assess the effect of the degree to which the combined outputs differ.

Chapter 4

Supporting infrastructure: the B3 database

This chapter describes an infrastructural approach, implemented in a relational database. The approach supports task-based parser output combination in two respects. First, it provides the data structures to store different types of parser output and to conduct queries on this output; second, it allows to keep track of the processing steps by representing each step in the database as well.

While the former allows us to also implement combination schemes within the database, the latter represents which parses are available for a specific sentence, and how they have been generated (i.e. with which pipeline, in which version, etc.).

The infrastructural approach uses a relational database management system. The approach was developed based on the demands of a specific project, i.e. the project B3 of SFB 732 (cf. Section 4.1), but with a focus on extensibility and data structures generic enough to serve as supporting infrastructure for different kinds of computational linguistic projects. While the database is still denominated as the B3 database[1] or B3DB, it has been successfully applied in further project work (e.g. Riester and Piontek 2015; Schweitzer et al. 2012).

[1] PID: `http://hdl.handle.net/11022/1007-0000-0007-BFEA-B`

As it was used in different projects, the B3 database has evolved over several years. It is thus joint work with many people who were involved in this evolution, be it for the initial schema design or the addition of frequently used stored procedures[2]. A rough overview identifies three major stages in the development: the initial implementation, the first extension and the most recent schema as described at the end of this chapter (Section 4.5). The first implementation of the B3 database was realized according to a schema by Andreas Madsack and Kurt Eberle. For the second major development step, this author provided an extension based on the Linguistic Annotation Framework (LAF, ISO 24612:2012) and work by Kountz et al. (2008), cf. Eckart (2009). Next to this author's work for the development from the second stage of the schema to the version presented here, work by Ivanova (2010) and Lu (2013) with respect to data integrity was integrated.

While this chapter will focus on the internal design of the database, there has also been work on query support and visualization by Raubal (2011), Ulusoy (2014) and by Moritz Stiefel.

With regard to this thesis, this chapter introduces an appropriate infrastructural background for the task-based parser output combination approach. However the chapter is also intended to serve as a stand alone introduction to the motivation for and design of the B3 database and as a documentation of the current schema. This chapter thus aims at being mostly readable independently of the rest of this work, while still fitting seamlessly into the overall approach.

4.1 Framework and objectives

The B3 database was developed within project B3 of the German collaborative research centre SFB 732 – *Incremental Specification in Context*[3]. The context project B3 focused on was the – mostly sentential – context of German *-ung* nominalizations. Those *-ung* nominalizations, i.e. nominalizations derived from verbs by adding the suffix *-ung*, can be ambiguous between different readings, cf. Ehrich and Rapp (2000). In project B3 – *Disambiguation of German -ung*

[2]Functions implemented within the database management system.
[3]Cf. Section 1.1 for more information on SFB 732.

nominalizations in corpus data extraction[4] – indicators from the context were identified which help to specify and disambiguate the readings of the *-ung* nominalizations, cf. Eberle et al. (2009).

More generally, the objective of the project was to develop linguistic hypotheses about the possible readings and the disambiguating context partners and to test these hypotheses on corpus data. This leads to the following approach for the refinement of linguistic hypotheses, visualized on an abstract level:

$$hypothesis_1 \rightarrow test\ on\ data \rightarrow inspection\ of\ results \rightarrow hypothesis_2$$

Within project B3 this approach took mainly the syntactic and the semantic level of linguistic description into account, but its principle is quite common to projects working with linguistic hypotheses and corpus data and is also applicable to other levels of linguistic information. With respect to the workflow types discussed in Section 1.2.4, this approach is best reflected by a bootstrapping workflow. While the hypotheses are conceived by the researchers, several analysis processes which are applied in tests of the hypotheses on corpus data include results of automatic tools, e.g. syntactic parsers. Therefore the creation and enhancement of the analysis processes, as well as the management of the resulting analyses produced by this processes also became an objective in the project work and increased the need for an infrastructural function supporting linguistic research. Other approaches coping with different aspects of an infrastructural function include e.g. the ANNIS/PAULA framework (Chiarcos et al. 2008; Krause and Zeldes 2016) employing a relational database as a backend, LAUDATIO[5] (Krause et al. 2014) comprising also process metadata, or format-related ISO standards, such as LAF (ISO 24612:2012). Taking other approaches into account, the B3DB incorporates data structures based on the LAF standard (Eckart 2009) and a collaborative pilot study was conducted to support a full mapping of parser output from BitPar and the B3-Tool[6] (Eberle et al. 2008) by vi-

[4]Original title in project phase I: *Disambiguierung von Nominalisierungen bei der Extraktion linguistischer Daten aus Korpustext*; phase II: *Disambiguierung von Nominalisierungen bei der Datenextraktion aus Korpora: Morphologisch verwandte Wörter*.

[5]LAUDATIO PID: `http://hdl.handle.net/11022/1007-0000-0000-8E65-F`

[6]A processing tool from project SFB 732 B3.

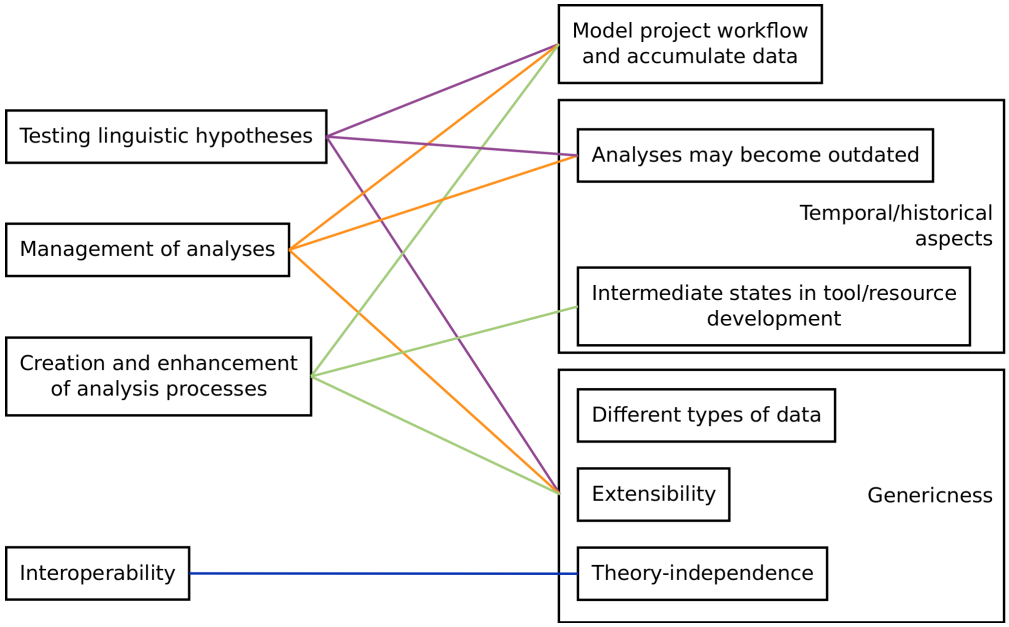


Figure 4.1: Objectives and requirements for the technical infrastructure.

sualizing parsed sentences in parallel applying the ANNIS/PAULA framework (Chiarcos et al. 2010).

Following a software development process (e.g. Ludewig and Lichter 2013: pp. 155f.), the requirements for the infrastructure, which were derived from the project objectives, are presented in Section 4.2. The technical decisions are discussed in Section 4.3 and the design decisions to implement the requirements are presented and discussed in Section 4.4. Figure 4.1 gives an overview of the relationships between objectives and requirements.

4.2 Requirements

A general requirement derived from the project objectives is the management of accumulating data as well as the ability to model the project workflow. Each step of the workflow and the corresponding data should adopt a representation in the database.

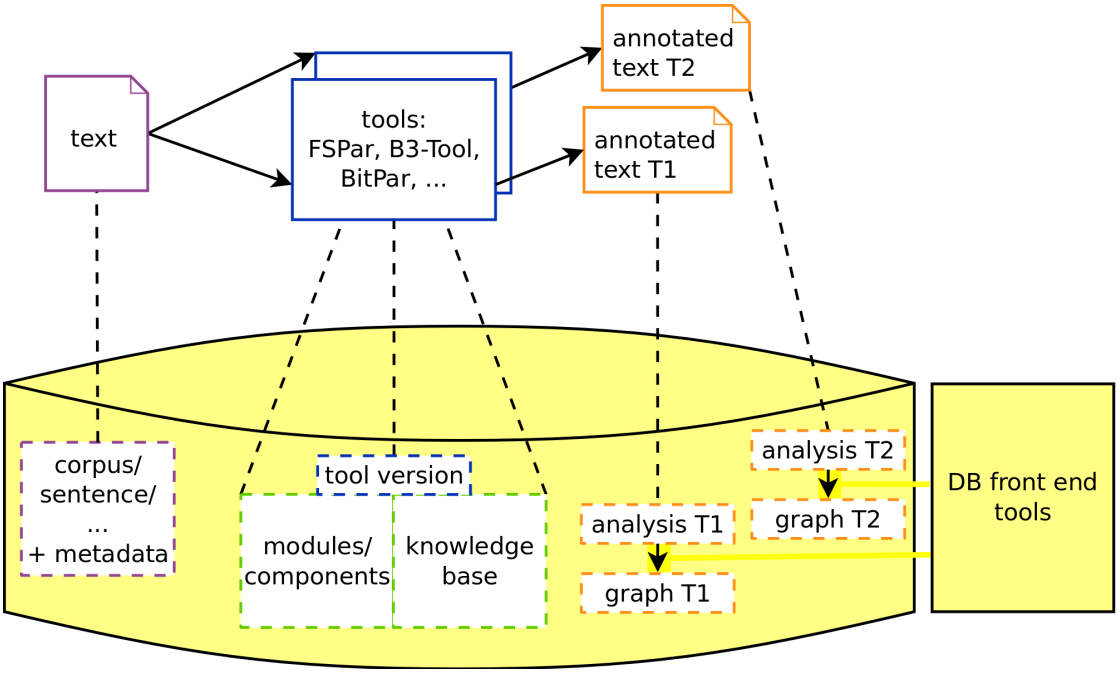


Figure 4.2: Project workflow and data and their mapping to the database.

Figure 4.2 shows an example workflow based on the processes and accumulated data utilized in project B3. In the leftmost position of the depicted workflow is the primary data (cf. Section 1.2.1), i.e. a corpus text, which is represented also in the database, sometimes enhanced by some metadata information, such as the source of the primary data or the original encoding. In the next step, this set of primary data is processed by analysis tools such as the TreeTagger (Schmid 1994), FSPar (Schiehlen 2003), BitPar (Schmid 2004) or the B3-Tool (Eberle et al. 2008). To keep track of the evolving tools, at least the information which version of the tool was utilized to process the primary data has to be stored in the database. If available, also some modules of the tool or a changing knowledge base, i.e. a lexicon or a set of rules, can be stored as strings in the database to keep track of the changes.

Most natural language processing tools produce a string-based output representing either stand-off or inline annotations to the input data. These strings are represented in the database as the tool output, i.e. the analyses. Since many such

analyses include a graph-based structure (e.g. parse trees) individual analyses can additionally be stored as graph-based representations.

The general requirement, the modelling of the project workflow along with the management of the accumulating data, can be split again into two parts, namely the requirement for **generic** structures and the ability to model **temporal** aspects. Generic structures include the possibility to model very different types of data objects by means of the same data structures, e.g. modelling primary data, information about tools, and the tool outputs, as seen in the example workflow in Figure 4.2. Moreover it has to be possible to introduce completely new types of data objects which might become necessary during an evolving project but have not been taken into account in the first implementation of the infrastructure. So the generic infrastructure also has to be easily **extensible** to new types of data, i.e. without changes to the database schema. Being generic also involves that the data structures do not reflect a particular linguistic theory but are modelled **theory-independently**. That way it is possible to take into account different analysis types, based on different theoretical approaches – a very high level example would be constituency vs. dependency annotations.

The modelling of temporal aspects is necessary to reflect the workflow processes and the constant evolution of tools. On the one hand analyses may become outdated if a newer tool version exists, on the other hand it can be helpful in tool or resource development as such to represent intermediate states one could reproduce or go back to, e.g. in order to discard a no longer functional development branch.

4.3 Technical decisions

The technical decisions made with respect to the required infrastructure were to utilize a relational database management system (RDBMS), and to implement the database based on PostgreSQL[7].

PostgreSQL databases are accessed via the SQL language, either directly by a command line tool (psql) or an administrative graphical user interface

[7] `http://www.postgresql.org/`

SELECT <columns>
FROM <tables>
WHERE <conditions on (combinations of) tables>;

Figure 4.3: Basic clauses of an SQL query

(pgAdmin3), or indirectly, i.e. by utilizing interfaces to programming languages such as Java, Perl, etc. An administrative GUI is transparent with regard to the content of the database, i.e. it does not support users in producing queries for a linguistic corpus but in writing queries for individual use cases. PostgreSQL also provides internal procedural languages to build database-specific functions.

Figure 4.3 shows an abstract example of a simple SQL query. Choosing to stick with direct SQL queries for access to the B3DB reflects the generic approach of the database, as SQL is independent of the actual kind of data queried. This is often seen as a disadvantage, which gave rise to different query languages tailored to the underlying data (e.g. TIGERSearch for syntactic structures, cf. Lezius 2002). This specialization of query languages is supposed to make structured data more easily accessible to users. In the same sense, by avoiding that users have to learn different query languages, the use of SQL provides a genericity advantage. SQL serves as one query language for several use cases, while the users have to become acquainted with the database schema and the mapping of the data for each use case. To facilitate the application of queries it is helpful to provide documented query templates, or deduce views which help to visualize and refer to data in a common way, e.g. as a table in the format of the CoNLL shared tasks (Hajič et al. 2009) for dependency trees.

SQL is a set based language and each query condition restricts the result set. There are several different approaches to traversing deep structures like trees and graphs, e.g. by means of nested sets (Celko 2004) or transitive closures.

While the decision of PostgreSQL over other types of SQL databases is mainly based on availability, there are other approaches to data structures, such

as graph databases[8] or XML-based structures. However, there is a high degree of standardization regarding SQL, which leads to the query advantages mentioned above. Graph databases find their advantages in even more unstructured data than the analyses and workflows we want to track. Although the workflows are graph based, there is still a time-based direction in their processing, which restricts the flexibility of a graph database and does thus not natively fit with the approach. Next to their underlying graph structure, further structuring can be included in graph databases by means of types. For the B3DB, we make use of the relational structures as well as additional typing and an object-relation setting by means of these structures. Thus an SQL database provides support for the several levels of data and supports the idea of structuring objects and relations, while leaving their type and content open such that the structures are generic enough to capture all relevant data. For further discussion on the advantages of relational databases which were taken into account for the technical decisions in the setup of the B3DB see Davies (2005) and Eckart (2009).

4.4 Design decisions

There were several decisions made with respect to the design of the database to support the objectives presented in Section 4.1 and to fulfil the requirements from Section 4.2. These decisions regarding the design are presented in the following paragraphs.

Separating macro and micro layer. The database is conceptually divided into two parts: the **macroscopic** layer and the **microscopic** layer. This partitioning is reflected by the data structures, as each table is either part of the micro layer or part of the macro layer. Conceptually, all objects which are viewed as atomic objects are represented on the macro layer, and all objects which might be queried with a structural view, can appear also on the micro layer. Thus, process metadata as defined in Section 1.2.4 is reflected on the macro

[8] E.g. Neo4j: `https://neo4j.com/`

layer, while generated annotations are reflected on the macro layer as workflow objects and also on the micro layer as structured and searchable information.

The example workflow in Figure 4.2 shows the text corpus along with its metadata and the information about the tool (versions), which are represented on the macro layer. The analyses however are represented on the macro and the micro layer. On the macro layer they are atomic objects, i.e. the output strings of the analysis tools; on the micro layer they are split up into one or more graph-based representations for each string.

Generic object relation structures on both levels. Another major decision was to implement mainly generic object relation structures on both layers (macro layer as well as micro layer). Structures of typed objects and relations appear on the macro layer with the tables `obj_definition`, `obj_relation` and `type_definition` and on the micro layer with the tables `node`, `edge` and `graph_type_definition`. On the macro layer every atomic object has an entry in table `obj_definition`, independently of being a primary data object, a tool (version), an analysis, etc. Table 4.1 shows an excerpt of the columns of table `obj_definition` displaying example entries for three different objects on the macro layer.

obj_id	obj_descr	obj_type	grp
12	B3Tool	5	tool
1234	ung-collection	349	corpus
1237	ung-collection.out	351	analysis

Table 4.1: Example entries for the first columns of table `obj_definition`.

Let the object with `obj_id` 12 be a version of the B3-Tool, the object with `obj_id` 1234 a loose collection of sentences, each containing one or more *-ung* nominalizations and the object with `obj_id` 1237 the output string generated by the B3-Tool on input of the content of object 1234. The attribute `obj_id` is a unique identifier for each object in the table `obj_definition` (primary key) and the attribute `obj_descr` comprises a name or a short description of the object.

Due to the fact that all objects have entries in the same table, a type system is needed to classify the objects. This is instantiated by (type,group) pairs

represented by the attributes `obj_type` and `grp` and in table `obj_definition`. The `grp` attribute introduces a classification into major groups such as *corpus*, *text*, *sentence*, *tool*, *analysis*, etc. whereas the `obj_type` attribute denotes a fine-grained typing within a group.

A type can either be atomic (operator: `id`), negated (operator: `neg`) or built in a complex way from several subtypes by means of a conjunction or disjunction (operators: `and`, `or` respectively). In this way many different features can be combined into a detailed description of the object by just assigning a type to it. While a type can naturally be part of more than one other type, each complete type is unique within its group. In this context, *unique* refers to the Boolean combination of the subtypes, two types may still be equivalent as regards their information content.

In fact the values for the (type,group) pairs are all user-defined, which allows on the one hand for enormous flexibility for the included types of data, but on the other hand, it also creates the opportunity for uncontrolled growth of the value set, increasing the risk of multiple types sharing the same semantics. We thus provide an (extensible) vocabulary for the group values, but leave the type values to the user. The group values which have been established in our database instances so far are listed alphabetically in Table 4.2. An additional approach could be to have the types, or at least the atomic types, refer to some sort of concept registry as proposed in ISO 12620:2009.[9]

Together, the attribute pair (`obj_type`,`grp`) refers to table `type_definition` (foreign key) where each type is listed along with its group and the subtypes it consists of. For each type, two subtypes can be given, such that a type consisting of more subtypes has to be built recursively. Table 4.3 shows an excerpt of the `type_definition` table containing example types referenced in Table 4.1.

The *B3Tool* object is of group *tool* and the type name is *ling_ana*, denoting a tool conducting linguistic analyses. The *ung-collection* object is a *corpus* and its type name denotes that it contains a *collection* of *news* text. The information contained in the type is useful for example for deciding upon the tools to process a corpus and to interpret the results. It informs the users that the corpus does consist of single sentences and not of continuous text, and thus,

[9]This last published version of ISO 12620 is currently withdrawn but an update is expected.

value	description
analysis	output of a tool or result of a manual annotation step, which refers to the complete set of input data
annotator	operator of a manual processing step
component	part of a tool
corpus	collection of primary data
graph	linking point for the micro layer, objects of this type are available as structured objects
inspection	output of a tool or result of a manual extraction step which only refers to parts of the input (e.g. extraction of only those sentences which contain a specific lemma)
lexicalentry	encoded entries of a lexical resource, discourse representation structures, etc.
lexicon	lexical knowledge base, e.g. as result of a workflow (might contain overlap with the component group, cf. Section 1.2.4 on the bootstrapping workflow)
sentence	sentence-like part of primary data
sequence	continuous sequence of language data which can be of arbitrary size but is neither a **text** nor a **sentence** (e.g. utterances, nominal phrases, etc.)
text	primary data of text context comprising e.g. a document layer
tool	processing tool in a specific configuration which can be applied on input data

Table 4.2: Group values for objects on the macro layer.

e.g. intersentential coreference resolution would lead to corrupted results. The other corpus feature *news* could be helpful in cases where a statistical parser which has been trained on news text is expected to be appropriate when applied to news text again. Nevertheless these are two really different features of the corpus, which is reflected by the type being a Boolean combination of two single features.

The last object in our example, *ung-collection.out*, is an *analysis* object, typed by *dep & nlist*. The type denotes that the processing result is a dependency

type_def_id	name	grp	ref_id1	ref_id2	operator
5	ling_ana	tool			id
93	dep	analysis			id
347	collection	corpus			id
348	news	corpus			id
349	collection & news	corpus	347	348	and
350	nlist	analysis			id
351	dep & nlist	analysis	93	350	and

Table 4.3: Example entries for the first columns of table `type_definition`.

analysis in an nlist representation format, a proprietary output format of the B3-Tool.

This typing mechanism is applied in the same way to relations between objects, stored in the table `obj_relation` and to the `node` and `edge` tables on the micro layer. To keep macro and micro layer separate however there is a separate table for the types on the micro layer, i.e. `graph_type_definition`, which the (type,group) pairs of the tables `node` and `edge` refer to.

For object relations on the macro layer, groups consist of the group of the source object of the relation and the target object of the relation separated by an underscore, e.g. *sentence_analysis*. Object relations can for example denote a processing step, e.g. if a relation is introduced between the *ung-collection* corpus object and the analysis object *ung-collection.out*, it is typed with the tool version and the call parameters utilized to generate the analysis.

While at first it may seem confusing and producing unnecessary overhead to include entries for each object in the same table and then classify them again into their groups with the help of another table, it is exactly this setting which allows us to fulfil the requirements of genericness and extensibility. Introducing new kinds of objects, even those which have not been thought of in the first place while setting up the project workflow simply means to insert new (type,group) pairs into the `type_definition` table. No changes to the database schema are necessary whatsoever.

Separating objects from contents. In the previous paragraph objects like corpora, tools and annotations have been introduced to the database by inserting an entry to the `obj_definition` table. While sometimes it suffices to declare the existence of an object in the database (**abstract object**), the objects actually often denote a content, i.e. a character string which forms the corpus, analysis output, etc. which can be stored in the database as well (**content object**). Storing the content along with the object entry in the `obj_definition` table would encounter some disadvantages. At first, the table would be populated very unbalancedly, as some entries would include a whole corpus whilst others would not have a content entry at all. On the other hand, contents or parts of contents might be stored several times as they occur in more than one object. Therefore a separate table `obj_content` is introduced on the macro layer. Content strings are inserted to this table and referenced by respective entries in the `obj_definition` table via the attribute `content_id` (foreign key). Figure 4.4 shows the abstract corpus object *ung-collection* which is related to a set of sentence objects,[10] referencing entries in the `obj_content` table. The

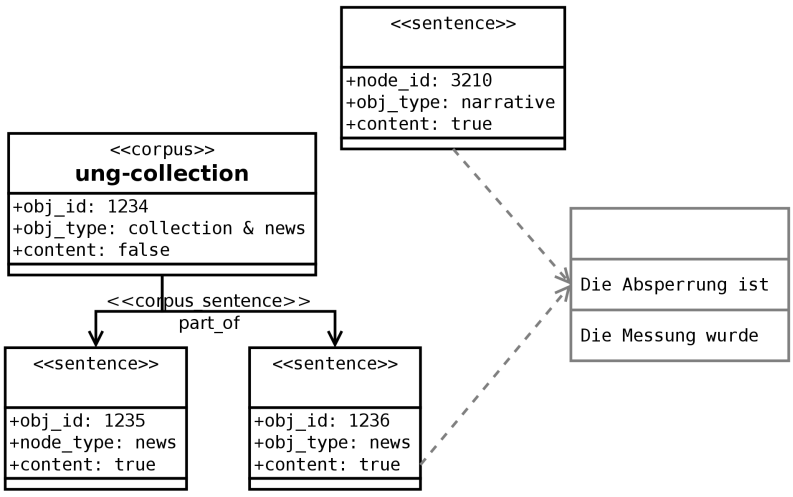


Figure 4.4: Separating objects from their contents on the macro layer.

benefit of being more efficient in disk space and handling comes at the cost of having to prevent dangling references. Dangling references occur if an

[10] In case of continuous text the ordering of the sentences is preserved by adding a position to each sentence object via the `attribute` table.

entry in the `obj_content` table is deleted, while there is still an object in the `obj_definition` table referencing this content. Therefore a delete command to the `obj_content` table has to be rejected, as long at there is still a corresponding object entry. Nevertheless this is already handled internally by the database due to the foreign key constraint (ON DELETE NO ACTION[11]). On top of that, the concept of the database will usually include delete commands only in exceptional cases, cf. the following paragraph on timestamps.

Modelling temporal aspects by timestamps. Modelling the project workflow also includes modelling temporal aspects. Tools may be changed due to errors or enhanced due to newly acquired information. Parts of the workflow may be processed more than once with different data sets. And existing analysis results may become invalid or lead to new insights, which again lead to changes in the knowledge bases of the tools. To model this complex and sometimes circular procedure timestamp attributes are applied to the tables of the macro layer.[12]

Table 4.4 shows again an excerpt of the `obj_definition` table now also including the attributes `created` and `invalidated`.

obj_id	obj_descr	obj_type	grp	created	invalidated
12	B3Tool	5	tool	2010-06-27	2011-11-29
1234	ung-collection	349	corpus	2011-11-28	
1237	ung-collection.out	351	analysis	2011-11-28	2011-11-29

Table 4.4: Excerpt of columns with example entries for table `obj_definition`, including temporal information.

In this example[13], the *B3Tool* object has been inserted into the database on the 27th of June in 2010 and the *ung-collection* corpus object on the 28th of

[11]ON DELETE NO ACTION is the default behaviour and will raise an error if a row should be deleted which is still referenced. NO ACTION lets the check be deferred until later in the transaction while RESTRICT will not.

[12]On the micro layer only the tables defining the types and the `value_type` table contain timestamps, see the paragraph on static graph representations for details.

[13]The timestamps used in the example would not be detailed enough to be actually employed. The data type for the timestamps used in the B3DB is PostgreSQL's TIMESTAMP-WITHTIMEZONE, containing date, time and time zone information, example: "2009-07-28 18:36:31.545443+02".

November in 2011. The corpus has then been processed the same day its object was inserted, so the result analysis generated by the B3-Tool has also been inserted on the 28th of November. Due to changes to the lexicon of the B3-Tool a new version might have been inserted on the 29th of November in 2011, thereby invalidating the old version. This does not affect the ung-collection corpus but does affect the analyses generated from it with the old version of the B3-Tool. There are two possible methods how to deal with this. Either all analyses created by the old version of the B3-Tool become invalid at the time this version does or invalidation is postponed as long as no new version of the actual analysis has been created, i.e. by processing the same primary data with the new tool version. Irrespective of the decision which method to apply, the timestamp mechanism allows for administration of data interpretations. On the one hand quality changes of evolving tools can be followed and changes causing a decrease in the quality of the analysis can just be withdrawn by restarting the workflow with the old tool version. On the other hand reliability details can be pointed out by comparing different objects or states of objects. These are also the reasons, why entries are not deleted but invalidated.

A third method is to ignore the timestamps completely and only establish relative temporal relations by directed relations between two objects. This is also applicable, when the original dates of the procession steps are no longer known. In this interpretation, the timestamps only mark the date when an entry was added to the database.

While the timestamps allow for some sort of versioning, it is not in the scope of the B3DB architecture to provide the functional range of a version control system. It would for example not be relevant for the workflow to monitor a state of a tool in which it has never been processing any of the relevant workflow data. Series of smaller tool changes and processing of test cases or a cascade of character encoding conversions of primary data therefore do not have to be represented in the B3DB unless their results become again relevant for the project workflow. The data or tool states included in the database can be thought of as releases which are actually interacting in the workflow or as states to which one would like to go back to or possibly be able to start again from.

All annotations are graphs. The probably most relevant design decision for the micro layer is to regard all annotations as graphs. The content string denoting an analysis result usually includes some sort of internal structure, representing how the annotations are attached to the primary data, or how they relate to each other.

To represent this structure on the micro level, each structure is mapped to the concept of nodes and edges. This concept is evident for parse trees, however, an actual tree structure would not suffice to represent different kinds of annotations, for example the concept of secondary edges as applied in the TIGER annotation scheme already exceeds the tree model (Albert et al. 2003).[14]

Full graph capabilities allow for the representation of trees as used in constituency-based syntax analyses, directed acyclic graphs as sometimes needed in dependency parsing, nets for synonym linking and also basic sequences as in speech alignment or part-of-speech tagging, where annotations are directly added to the sequence of parts of the primary data. Such an annotated sequence can be represented by a trivial graph without edges, i.e. consisting only of nodes containing the annotation.

The nodes and edges are stored in respectively named tables on the micro layer, their entries being typed to distinguish different groups of nodes and edges. Tables 4.5 to 4.7 show two examples for node-edge combinations. These examples belong to two different graphs, one being a representation of a dependency parse created by the B3-Tool the other being a representation of a dependency parse created by the mate parser (Bohnet 2010). Figure 4.5 shows excerpts of the string representations of those analyses. In Figure 4.5a the first three lines denote the token *Land* with its annotations by a node in nlist representation (n-node), and the last line denotes its determiner *des*. The first entry of an n-node 6-tuple states the token position and the last entry the dependents of this token along with the label of the dependency relation. Therefore `n(11,s(land,..)  ...[..,ndet:10])` states that there is a dependency relation of type *ndet* between tokens 11 and 10 of which token 11 is the head. In the database tables this is represented by two nodes of

[14]The TIGER annotation scheme applies secondary edges for omitted constituents in coordination of verbal phrases and sentences.

```
n(11,s(land,422061),[slot(nid,op,_11100441,st(countryname'|'unknown)),
slot(nobj,op,_11100453,_11100454)],noun(cn,gen,pers3-sg-nt,[]):[[ctry,inst,land]],
ngen,[head,ndet:10]),
n(10,s(des,d),[],det(gen,pers3-sg-nt,[def]):[[d,des]],ndet,[head]),
```

(a) B3-Tool output in nlist representation for the tokens *des Landes* in the context *in den 18 Provinzen des Landes* ('in the 18 provinces **of the country**')

636_8	der	_	der	_	ART	_	dat|sg|fem	_	9	_	NK	_	_
636_9	Abgabe	_	Abgabe	_	NN	_	dat|sg|fem	_	7	_	NK	_	_

(b) Excerpt of output from the mate parser in the CoNLL format from 2009's shared task (Hajič et al. 2009) for the tokens *der Abgabe* in the context *bei der Abgabe der Finanzierungsbestätigung* ('on **the delivery** of the confirmation of financing')

Figure 4.5: Examples of different tool output.

group *b3_tool_nlist* and an edge of of group *b3_tool_tree_edge* and type *ndet*. In Figure 4.5b the two lines denote one token each, and the position of the head token is listed along with the label of the dependency relation in each row (10th and 12th column in the example). The respective database entries in the `node` table are of group *conll_node* and in the `edge` table of group *conll_edge*.

node_id	graph_id	node_descr	node_type	grp
35027	751	n	10	b3_tool_nlist
34817	751	n	10	b3_tool_nlist
272252	5142814	636_8	161	conll_node
272253	5142814	636_9	161	conll_node

Table 4.5: Excerpt with example entries for table `node`.

source_id	source_graph	target_id	target_graph	edge_type	grp	edge_id
35027	751	34817	751	85	b3_tool_tree_edge	56723
272253	5142814	272252	5142814	163	conll_edge	9756471

Table 4.6: Excerpt with example entries for table `edge`.

Taking these two examples into account, the flexibility of mapping to the data structures shows along with the resulting complexity in querying. The label of the dependency relation can be represented by the edge type as in the

graph_type_def_id	name	grp
10	nlist_element	b3_tool_nlist
85	ndet	b3_tool_tree_edge
161	2009	conll_node
163	2009	conll_edge

Table 4.7: Excerpt with example entries for the first columns of table graph_type_definition.

representation of the B3-Tool example, but can also be added to the edge as a distinct annotation. The node_descr attribute may contain the token position as in the CoNLL example or state any other information related to the node. This leaves some of the complexity of the data representation to the querying as the users need to have detailed insight into how the analysis data has been mapped to the data structures to conduct reasonable queries. On the other hand the flexibility of this approach allows for canonical representation of different structures and with respect to different tasks. And if different formats need to be represented in a most similar way, more than one graph representation can be inserted for the same analysis object.

Graph representations are static. In contrast to the macro layer, where all tables contain temporal information, the tables containing the specific graph representations on the micro layer do not include any temporal markers. This is due to the fact that each of the graph representations has been created by specific rules applied to an analysis object at a specific point in time. With respect to this graph segmentation algorithm, the graph representation is always valid and no partial modifications, deletions or updates should be applied to the graph representation on the micro layer.

The temporal information of the graph representation as a whole is attached to an object of group *graph* on the macro layer, which each contained node refers to by the attribute graph_id (foreign key). If the segmentation algorithm changes or another kind of graph representation is needed for the analysis, the analysis string can be processed again and is stored in addition to the existing

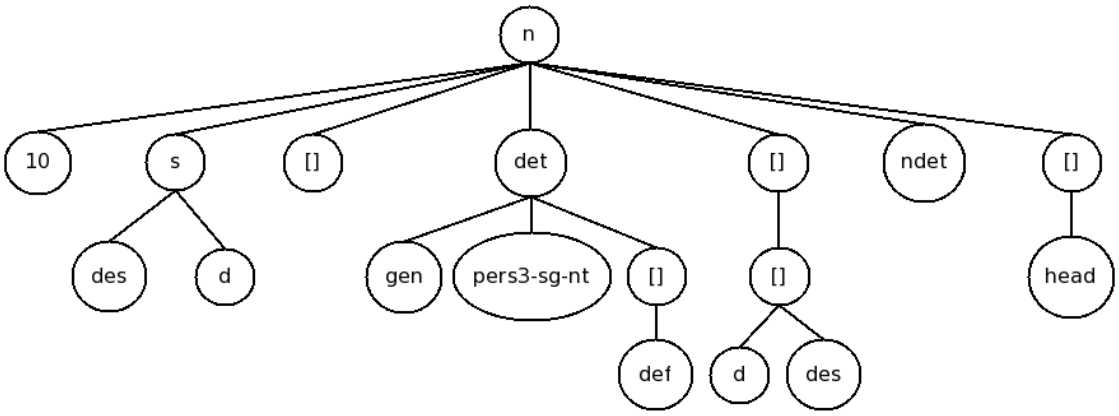
representations. If a representation is not needed anymore it should also be deleted as a whole.

An example where more than one graph representation could be needed for an analysis is the direction of the dependency edges in the above examples of Figure 4.5 (see also the discussion in Section 2.1.3). In the analyses of the B3-Tool the canonic way to represent the dependency relations would be from head to dependent. In the analyses of the mate parser the information of the head is encoded in the annotations of the dependent, so the canonic way here would be to insert edges from the dependent to the head. If queries only include analyses from either the B3-Tool or the mate tools parser, this can easily be accounted for with different query templates, but if different parses should be queried at once, it is helpful to provide the edges in the same direction.

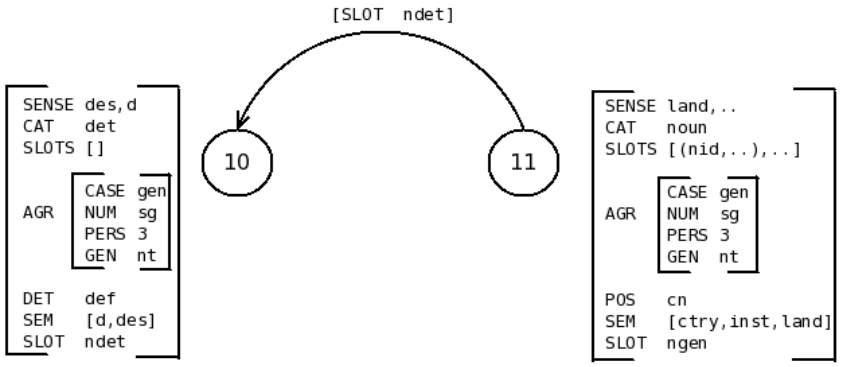
Another example for the usefulness of concurrent graph representations is the nlist representation of the B3-Tool. It can either be segmented into a graph according to its bracketed structure or according to the linguistic dependency tree encoded in this structure, cf. Figure 4.6 for examples of both representations for the example in Figure 4.5a.

Another advantage which comes with the concept of static graph representations is that it facilitates the availability of a transitive closure, which allows for fast identification of subgraphs and graph-traversal. Since no modifications are allowed within the graph representations in the database the transitive closure has to be build once per graph only instead of keeping track of modifications (inserts, deletions) of nodes and edges.

GrAF-based data structures for complex annotations The data structures of the micro layer are based on the structures proposed by LAF ISO 24612:2012, an ISO standard on the Linguistic Annotation Framework; its XML-serialization GrAF, the Graph Annotation Format and an extension by Kountz et al. (2008). For a detailed discussion of LAF and GrAF see Ide and Romary (2006); Ide and Suderman (2007); Eckart (2012); Ide and Suderman (2014). LAF is designed to constitute an exchange format to be mapped into and out of, from and to many different representation formats.



(a) Graph representation for bracketed structure of node related to token *des*



(b) Graph representation for dependency relation between tokens *des* and *Landes*

Figure 4.6: Differing segmentations of an analysis on the micro level.

Many of the concepts proposed in LAF adapt well to those considered for the B3DB (cf. Eckart (2009) for a detailed discussion). LAF proposes the distinction of annotation structure and annotation content as discussed in Sections 3.1 and 3.2, and the sanctity of the original primary data object. The annotations are attached to the original primary data objects in a stand-off approach and the semantics of the annotations have to be encoded outside of the representation in LAF, e.g. by utilizing persistent identifiers (PIDs) from data categories stored in a Data Category Registry as proposed by ISO 12620:2009. The annotation structures are mapped to a graph-based structure consisting of nodes, edges and feature structures, which can be attached to nodes as well as edges to represent complex annotations. To refer to parts of the primary data object, regions of the primary data are introduced. Next to theory-independent and graph-based data structures, the property of being an exchange format also supports interoperability on data export for the data structures of the B3DB being based on those proposed in LAF.

All information is annotation The last design decision to be discussed here is that on the micro layer all information contained in an analysis string can be treated as annotation within the graph representation. On the one hand this allows for coping soundly with time-aligned annotations, e.g. timestamps occurring in the label files can be attached as annotations just like any other tag. On the other hand if process-internal or resource-internal metadata is part of an analysis string, it can also be included in the graph representation, therefore allowing for reproduction of the original analysis string. Examples for such metadata are the identifiers of terminal and non-terminal nodes in TIGER-XML. In Figure 4.7 the values of attribute `id` would be treated as annotations.[15]

4.5 Schema

This section describes the most recent version of the B3DB schema. The case studies described in Chapter 6 were conducted on different development

[15]This excerpt is part of an example from the DIRNDL corpus (Eckart et al. 2012).

```
<terminals>
   <t id="s10_1" word="Frau" pos="N[addr]"/>
   <t id="s10_2" word="Merkel" pos="NAME"/>
   [...]
</terminals>
<nonterminals>
   <nt id="s10_506" cat="NAMEP">
     <edge label="--" idref="s10_1"/>
     <edge label="--" idref="s10_2"/>
   </nt>
   [...]
</nonterminals>
```

Figure 4.7: Terminal and non-terminal nodes in TIGER-XML.

branches and B3DB instances from the second major stage of the schema to the current one. However, all tables needed for the data in the case studies are still part of the current schema.

Figures 4.8 and 4.9 show the current release of the B3DB schema.[16] Due to the size of the figure, only the names of the tables and attributes as well as the data types of the attributes are given. Further specifications such as default values and `NOT NULL` constraints are omitted. Main parts of the micro layer are based on the pivot format specification in ISO 24612:2012 (LAF), on (Ide and Suderman 2014) and on the suggestions by Kountz et al. (2008).

Sections 4.5.1 to 4.5.8 describe the parts of the schema according to the purpose of the included tables. Section 4.5.9 adds some notes on indexes.

4.5.1 Describing objects on the macro layer

As described in Section 4.4, the objects represented on the macro layer can be any kind of data object related to the project workflow, i.e. also any kind of resource as defined in Section 1.2.1. Examples of such objects are sets of primary data, analyses, tools, and tool components such as trained models, grammars or lexicons (cf. also the list of groups in Table 4.2).

[16] Denoted as version 6.0.

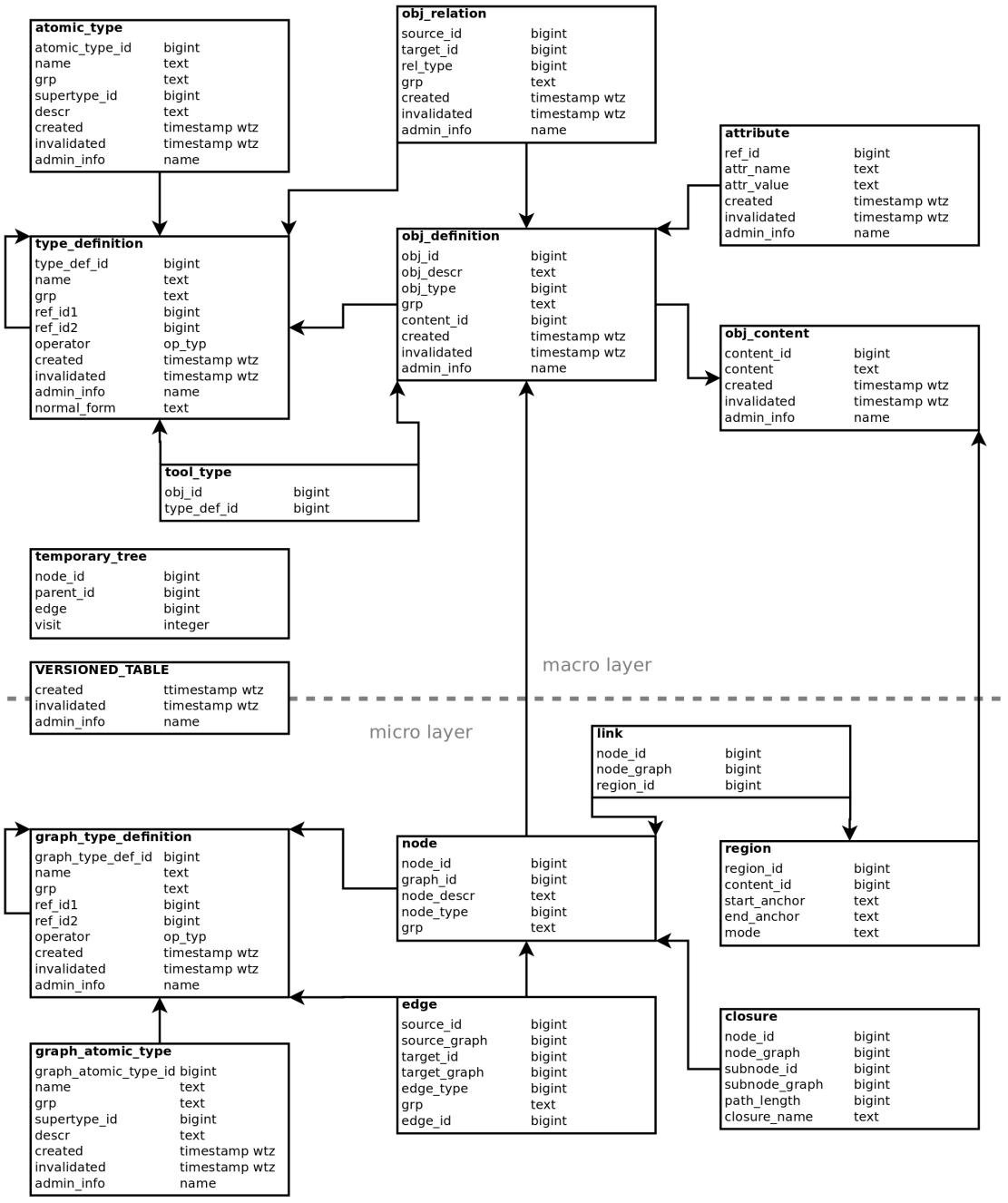


Figure 4.8: Part 1 of the B3DB database schema: macro layer and some parallel tables from the micro layer.

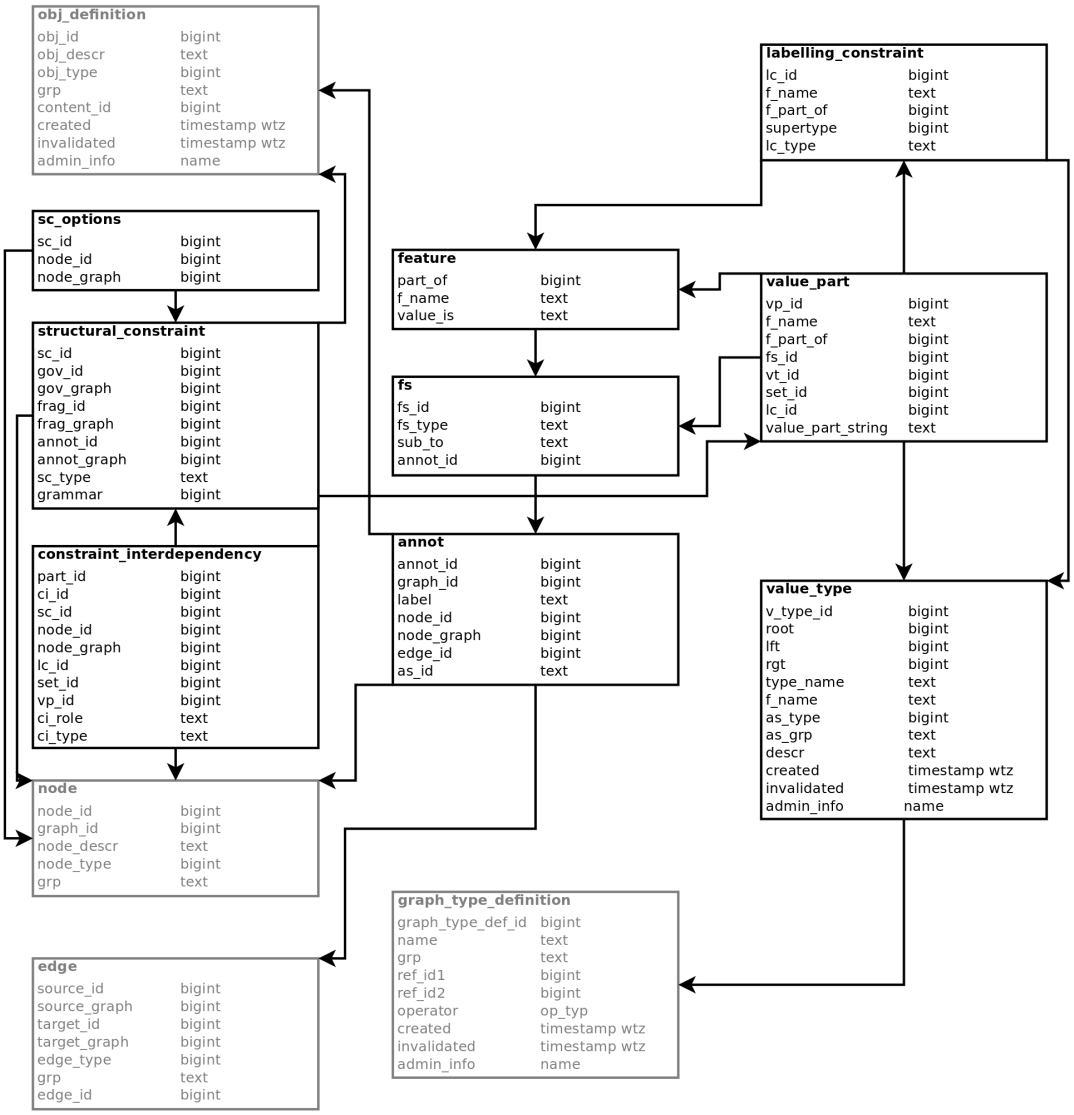


Figure 4.9: Part 2 of the B3DB database schema: encoding values and constraints on the micro layer (referenced tables repeated from Figure 4.8 are marked in grey).

The three tables related to the description of these objects on the macro layer of the database are:

obj_definition. Every object from the macro layer of the database is registered in this table. The object is uniquely identifiable by a numeric identifier (`obj_id`) based on a sequence, and, if a name is given in the `obj_descr` attribute, via the (`obj_descr`, `grp`) pair. This means that the name of the object (`obj_descr`) has to be unique within its group (`grp`), and thus no two corpora with the same value for `obj_descr` are allowed to coexist in the database. If the object has a content which is stored in the database (content object), the foreign key attribute `content` referencing table `obj_content` is applied. Every object is typed (cf. Section 4.5.5) and temporally marked (cf. Section 4.5.6).

obj_content. The table stores object content as strings (`content`). Cf. Section 4.4 for a discussion on an efficient representation of the object contents. While the approach to efficient representation is recommended, the database does not enforce this behaviour by means of the current schema, for it has so far not been tested for scalability to numerous large contents. Each content string is uniquely identifiable by a numeric identifier (`content_id`) and temporally marked (cf. Section 4.5.6).

attribute. Each object can have any number of entries in the table called `attribute`, where an entry is a feature value pair (`attr_name`, `attr_value`) consisting of two strings. There is no restriction on the features which can be attributed to an object. Examples are its author, its original character encoding or the position of a text or sentence object in a corpus. The entries of the table `attribute` refer to a specific object by means of the foreign key attribute `ref_id`, referencing table `obj_definition`. An entry from the `attribute` table is uniquely identifiable by a tuple consisting of the identifier of the referenced object, its name and its value (`ref_id`, `attr_name`, `attr_value`). The entries of the table are temporally marked (cf. Section 4.5.6) and whenever an object is deleted, all entries referring to this specific object are automatically deleted as well.

4.5.2 Describing relations on the macro layer

Relations on the macro layer exist between a source object and a target object, both also on the macro layer. Thereby the relation represents how two objects are associated with each other, for example by means of a workflow step (cf. Section 1.2.4) where the source object is the input and the target object is the output of the respective step, or in a part-of relation where a specific version of a library is part of an abstract tool object.

All relations on the macro layer are kept in one table:

obj_relation. A relation is uniquely identifiable by its source and target objects with the foreign key attributes (`source_id`, `target_id`). That is, there can exist at most one relation between each pair of source/target objects, however a target object in one relation can of course be also the source object of another relation. If there is more than one kind of relation between two objects, this has to be encoded by a combined type for the relation (cf. Section 4.5.5). The group of a relation (`grp`) is a combination of the group of the source object and the group of the target object, a trigger is utilized to reject entries which do not conform to this pattern. Relations are also temporally marked (cf. Section 4.5.6) and are automatically deleted when their start or target objects are deleted form the `obj_definition` table.

4.5.3 Describing nodes on the micro layer

Similar to the objects on the macro layer, there are also nodes on the micro layer. However they do not represent full objects, but are parts of a graph representation of an object content. In case that a node represents a specific part of an analysis, i.e. an annotation of a region of the primary data, the node can be linked to this region of the primary data by means of virtual anchors as defined in LAF. Thereby the defined regions can be of different granularity, and overlap is allowed, since different annotations are related to different parts and segmentations of the primary data. Depending on the modality of the primary data, the anchors to specify the referenced part of the primary data can be e.g. character offsets (for textual data) or timestamps (for audio data).

node. Nodes are uniquely identified by a `node_id` and the reference to the graph object they belong to (`graph_id`). Nodes can carry a `node_descr`, e.g. an important part of the annotation or information on the order of the nodes, and they are typed (cf. Section 4.5.5). The group of a node (`grp`) can for example denote the annotation layer a node belongs to. In the case that a graph object is deleted, all nodes belonging to this graph are automatically deleted as well.

region. A region is a part of a primary data object. It is uniquely identified by a `region_id` and references the primary data by means of anchors (`start_anchor`, `end_anchor`). The attribute `mode` specifies the type of the anchor and has to be given, such that external applications can determine how to resolve the anchors. The default mode is the character offset for textual resources. Although LAF allows to specify any number of anchors needed to reference a region in the primary data, the setting in the B3DB with a dedicated start and end anchor is oriented towards textual resources or audio and video data when referred to by means of timestamps. When the content of the object is also stored in the database, the `content_id` attribute can be utilized to refer to the table `content`. In case of abstract objects with content outside of the database, the attribute `content_id` is NULL.

link. A link relates a node to a specific region. Its primary key is the tuple of all its attributes (`node_id`, `node_graph`, `region_id`). The `link` table defines an $n : m$ relation between the node and the region table. That is, a node can be linked to several regions and a region can be referred to by several nodes. When the node or the region are deleted, the respective links are automatically deleted as well.

4.5.4 Describing edges on the micro layer

Edges belong to the micro layer and are similar to relations on the macro layer. There are two major types of edges: (i) edges which are part of a graph representation of an object, and (ii) edges which link several graphs on the micro layer, e.g. when mapping two different tokenizations (cf. the case of DIRNDL in

Section 6.2.1). Since the graph representation is static, it is helpful to compute a transitive closure for edges which are part of a graph representation and store the closure in the database to circumvent recursive graph traversal.

edge. An edge is defined between a source node and a target node. However it differs from relations on the macro layer insofar as there can be more than one edge between a pair of source/target nodes. Thus an edge is uniquely identifiable by the combination of its source node (`source_id`, `source_graph`), its target node (`target_id`, `target_graph`), its type (`edge_type`, cf. Section 4.5.5) and the group it belongs to (`grp`). In contrast to relations, there are however no restrictions on the group value of an edge. The group value of an edge can for example denote the annotation layer the edge belongs to or mark it as a an edge linking different graph representations.

If edges are annotated, they are referenced by the attribute `edge_id`. The `edge_id` is unique and a numeric identifier is assigned to a new edge by default upon insertion. However for an edge which is not annotated, the `edge_id` is not strictly necessary due to the unique identifier described above. This identifier is set up analogously to the identifier for relations on the macro layer, while the `edge_id` follows the lines of the specification in LAF.

An edge is automatically deleted as soon as its source or target node is deleted from the `node` table.

closure. In a separate table, the transitive closure of a graph representation can be stored. Thereby the existence of a path in the graph from a start node (`node_id`, `node_graph`) to a target node (`subnode_id`, `subnode_graph`) is stored in its own entry. Optionally the length of the path can be given via the attribute `path_length`. It is obligatory to define a name for the closure (`closure_name`). The name allows the users to insert several closures into the database which refer to the same nodes, e.g. when a closure should only take edges of a specific type into account. By means of this device, there could exist e.g. one closure for a syntactic analysis based on the TIGER annotation scheme and taking secondary edges into account, and at the same time another closure which excludes secondary edges. An entry to the table `closure` is uniquely identified

by the combination of its start node, its target node and the name of the closure. The table carries no restrictions if the closure is reflexive or not. In the first case, a graph edge which is part of the closure is stored twice: once in the `edge` table and once in the `closure` table with `path_length` zero. This leads to redundant information, but queries do not have to take two tables into account to find all connections. In the second case, the storage is more efficient, while the queries need to take an additional table into account.

Start and target node of the closure have to belong to the same graph object, and as soon as one of the nodes is deleted the path entry in the `closure` table is deleted as well.

4.5.5 Typing

Objects and relations on the macro layer as well as nodes and edges on the micro layer are typed. The tables related to typing are the following.

atomic_type. Atomic types are types which are not supposed to be split any further. The atomic type is uniquely identified by the `atomic_type_id`, carries a name (`name`), belongs to objects or relations of a specific group (`grp`) and can be liked to `type_definition` by means of a dedicated supertype (`supertype_id`). In addition to the description possibilities of table `type_definition`, `atomic_type` can include a description, e.g. also a link to an external definition. Each entry in `atomic_type` is temporally marked (cf. Section 4.5.6).

type_definition. The table `type_definition` contains the types for objects and relations on the macro layer. As opposed to `atomic_type`, a type can be combined by means of subtypes (`ref_id1`,`ref_id2`) and a specific operator (`operator`) taking one of the values from {`id`,`neg`,`or`,`and`}. Types with operator `id` can be atomic types, but further splitting can be added later, while entries in table `atomic_type` are predefined atomic types and not supposed to change their state. A type has a name which is unique within the group of objects or relations it can be attached to (`name`,`grp`). It is uniquely identified by `type_def_id`

and temporally marked (cf. Section 4.5.6). Additionally a normal form of its (complex) structure can be kept in `normal_form`.

temporary_tree. The `temporary_tree` table is a purely administrative table. It supports data integrity with respect to the datasets in table `type_definition` and is used in one of its trigger functions.

tool_type. The `tool_type` table supports the typing of workflow steps, by linking types in the `type_definition` table to tool objects in the `obj_definition` table. It is the only table which (i) focusses on only one group of objects and the only non-administrative table on the macro layer which (ii) does not carry a temporal mark.

The first property (i) is due to this table allowing for some sort of workaround: A workflow step is specified by its input object, its output object and the relation between these two objects. The type of the relation usually describes the involved processes which were applied to the input to receive the output. However the type as such has no relation to a specific object, even if a known tool object was involved in these processes. To mark the involvement of a known tool object one option is to include an additional relation between the tool object and the output object. Nevertheless, this separates the treatment of workflow steps including known tool objects from the standard representation of the workflow steps. Thus, the table `tool_type` was added to allow for a uniform treatment: a known tool object can be linked to its type representation and can be combined with further types denoting, e.g. specific parameters or other tools involved in the workflow step.

Due to this specific purpose of the table it is also not included in the temporal marking scheme. It should be mentioned that besides the name of the table it is not determined that the object is actually of group *tool*. This is due to the fact that there might be additional groups included by the user which might require a similar treatment. This should not be restricted.

graph_atomic_type and graph_type_definition. These tables from the micro layer are parallel to the tables `atomic_type` and `type_definition` on the macro

layer. The unique identifiers are called `graph_atomic_type_id` and `graph_type_def_id` respectively. They are also temporally marked (cf. Section 4.5.6) but up to now, no normal form is added to the table `graph_type_definition` since the types are usually related to specific representations.

4.5.6 Temporal marks

Most tables on the macro layer are marked temporally with timestamps, as they reflect workflow aspects, which include an order or a point in time, e.g. when a data object has been created or a workflow step has been processed. Moreover, by means of timestamps, entries can be marked as no longer valid, for example in cases when a newer version of a tool has been created and the old one is not to be used any more. To invalidate an object in this case is preferred over deleting it, since the old version might still be needed for documenting an old workflow which led to specific results.

Additionally, types are also marked temporally, in order to show when they have been included to the set of types or marked as no longer valid.

By default, the temporal marks show when the entry was added to the database. While this might in many cases not reflect the actual creation time of the data object or processing step, it is often sufficient to be able to access the order in which the entries were made to the database. There are no explicit restrictions on the temporal scheme, so the users are free to define their own, or make use of the default approach. A general discussion of the design decision to model temporal aspects can be found in Section 4.4.

Since the graph representation of an object is always created with respect to a set of specific rules, its representation on the micro layer is static and not temporally marked. See Section 4.4 for details on the design decision behind this. Thus most tables on the micro layer are not temporally marked. However the type set on the micro layer has the same temporal aspect as the type set on the macro layer and is thus also marked temporally, as well as the table `value_type`, containing reusable tagsets which are not restricted to be applied to one specific graph only.

VERSIONED_TABLE. This table is part of the schema, but is only used as a pseudo-table which other tables can inherit structure from. It is placed in the schema such that it is created as the first table, and all temporally marked tables are then created with the attributes defined for the `VERSIONED_TABLE`. These attributes are (i) `created`, by default specifying the date and time when an entry was made to the database, (ii) `invalidated`, specifying the date and time of invalidation, if set, and (iii) `admin_info`, additionally specifying the "owner" of the entry, by default the user initiating the insert.[17] Thus the `VERSIONED_TABLE` facilitates writing the schema, because the attributes and defaults only have to be specified once, instead of for each table. However, this is no longer visible as soon as the schema is created. The attributes are fully specified for each table then, and also a schema dump is no longer aware of the role of the `VERSIONED_TABLE`. Changes to the `VERSIONED_TABLE` of an instantiated database have no effect on the other tables. Due to its special status, its name is written in all capital letters, the table is never filled, and it is neither assigned to the macro layer nor to the micro layer.

4.5.7 Describing annotations on the micro layer

Annotations on the micro layer are attached to the nodes and edges of the graph representation. Since it is not obligatory for a node or an edge to carry an annotation, the annotation entries refer to the nodes and edges and not the other way round. Moreover, a node or an edge can be decorated with various annotations.

The structure of an annotation is based on the XML serialization GrAF and extended according to Kountz et al. (2008), cf. Section 4.5.8. Regarding feature structures, GrAF adopts ISO 24610-1:2006. Figure 4.10 shows an example of an annotation encoded in GrAF.

The `<a>` element denotes an annotation for a node or an edge, which is referred to by means of the attribute `ref`. The category of the annotation is specified by the attribute `label`. In cases where the label defines the complete annotation, e.g. by means of a PID for a specific data category, or by means

[17] For the timestamps PostgreSQL's data type TIMESTAMPWITHTIMEZONE is utilized.

```
<a label="NN" ref="node01" as="TIGER">
  <fs>
    <f name="pos" value="NN" />
    <f name="morph">
      <fs>
        <f name="gender" value="Neut" />
        <f name="case" value="Gen" />
        <f name="number" value="Sg" />
      </fs>
    </f>
  </fs>
</a>
```

Figure 4.10: Morphosyntactic annotation of a common noun (part-of-speech NN), based on the annotation scheme of the TIGER corpus, and encoded according to GrAF.

of reference to a feature structure library, the `<a>` element can be empty (cf. Ide and Suderman 2014). Elsewise, `<a>` elements contain feature structures as defined in ISO 24610-1:2006.

In Figure 4.10 the label states the overall category, in this case a common noun (NN) from the *AnnotationSpace* (Ide and Suderman (2014), attribute `as`) TIGER.

The AnnotationSpace supports disambiguation, when annotation labels are subject to overloading, i.e. when several annotations are marked by the same name, but with different semantics. The AnnotationSpace information can also be used to mark annotations belonging to the same annotation layer.

According to the morphological annotation scheme of the TIGER corpus (Crysmann et al. 2005), a common noun is labelled with the morphological features case, number and gender. Figure 4.10 shows the part-of-speech annotation (`pos`) by means of a simple feature value pair and encodes the morphological features as a feature structure, which poses the value of the feature `morph`.

In the following the tables are presented, into which the annotation structure from GrAF was translated. While not the full power of feature structures from

ISO 24610-1:2006 was explicitly transferred to the database structures, many of its annotation possibilities can be emulated.

annot. The `annot` table represents the `a` element from GrAF and links annotations to nodes and edges. An entry refers either to exactly one node (`node_id`, `node_graph`) or exactly one edge (`edge_id`). A constraint is in place to check that this condition is met. Each `annot` entry is uniquely identified by its `annot_id`. A feature structure which is part of this annotation refers to the `annot` entry by means of the `annot_id`. An `annot` entry can be specified as part of a specific graph (`graph_id`), but does not have to, e.g. in cases where the annotation belongs to an edge linking different graphs. The settings do not by default restrict a situation where the `annot` entry is assigned to a specific graph but annotates a node from a different graph. However, if needed, this behaviour can be restricted by a simple `CHECK` constraint, testing that when `node_graph` is not `NULL` and `graph_id` is not `NULL`, the two attributes have to refer to the same graph object. The attribute `as_id` specifies the AnnotationSpace described by Ide and Suderman (2014). As opposed to most of the other attribute names ending in '_id', the AnnotationSpace is not restricted to an integer value, but can be any kind of text. The attribute `label` is mandatory.

When the referenced edge or node is deleted, or the graph object to which the annotation belongs is deleted as a whole, the annotation entry is deleted as well.

fs. The `fs` table represents the `fs` element from GrAF. Each entry is uniquely identified by the attribute `fs_id`. The attribute `sub_to` is mandatory and can by means of a constraint take the values `annot`, `feature` and `global`. In the first case, the feature structure is directly embedded in an annotation from the `annot` table, in the second case, the feature structure is the value of a feature and in the third case the feature structure is globally accessible by means of its `fs_id`, e.g. when it should be applied in multiple places. While stating an `annot_id` is mandatory in the first case and this behaviour is controlled by a `CHECK` constraint, it is optional in the other two cases. In case that an `annot_id` is referenced, the feature structure is deleted on deletion of this `annot` entry.

Since in ISO 24610-1:2006 feature structures can be typed, there is an optional attribute `fs_type` available.

feature. The `feature` table represents the `f` element from GrAF. It is uniquely identified by the combination of the feature structure it belongs to (`part_of`) and its name (`f_name`). On deletion of the respective feature structure the feature is deleted as well. The attribute `value_is` defines the kind of the feature value and can be one from the following list:
vt an element from a pre-defined set of values,
fs a feature structure,
string a simple string,
vt_set a set of pre-defined values,
fs_set a set of feature structures,
string_set a set of strings.
The values can be retrieved via the table `value_part`.

value_part. The table `value_part` relates the different kinds of values to their features. Each entry is uniquely identifiable by means of the attribute `vp_id` and references the feature it belongs to by the combination of the name of the feature (`f_name`) and the feature structure in which the feature appears (`f_part_of`). If the feature is deleted, the `value_part` entry is deleted as well. Values which are simple strings are represented by means of the attribute `value_part_string`. Values which are themselves feature structures are referenced by the attribute `fs_id` and values which can be found in a pre-defined set in table `value_type` are referenced by the attribute `vt_id`. A constraint enforces that exactly one of these three attributes is filled.

In case that the feature value is a set of values of either type, the attribute `set_id` is not `NULL`, and all values belonging to the same set, are marked with the same `set_id`, such that multiple entries to `value_part` constitute the actual feature value.

Reference from `value_part` to an entry in `fs` or `value_type` prevents deletion of these entries, because they are still referenced. It is however checked by means of a constraint, that there is no direct circular relationship within a

feature structure, i.e. the feature structure of which the feature is part cannot be (part of) its value.

If a labelling constraint refers to an entry in `value_part`, this is marked by the attribute `lc_id`, cf. Section 4.5.8.

value_type. The table `value_type` is able to represent predefined tagsets and especially tagsets with a hierarchical structure. Therefore it is based on nested sets (cf. Celko 2004) with the attributes `root`, `lft` and `rgt`.[18] The name of the value and the respective feature are described by `type_name` and `f_name`, respectively. Additionally a description can be given (`descr`). The value type is linked to a specific AnnotationSpace (`as_type`,`as_grp`). Each entry is uniquely identified and temporally marked (cf. Section 4.5.6).

4.5.8 Describing annotation constraints on the micro layer

Kountz et al. (2008) propose an extension for GrAF to efficiently represent ambiguities in syntactic analyses by means of constraints. In Eckart (2009) the underspecified representation from Kountz et al. (2008) is implemented as part of the B3DB. This section describes the resulting tables. For a discussion on the representation from Kountz et al. (2008) with respect to the database and a list of deviating design decisions in the implementation as relational database tables see Eckart (2009). For an example instantiation with dependency and constituency trees, as well as the basic query concept see Kountz et al. (2008).

The tables based on Kountz et al. (2008) have not been used in the case studies for this work and are thus not necessary for the reproduction of the experiments. Additionally, while the constraint representation from Kountz et al. (2008) provides a generic and extensible means to represent ambiguities and especially underspecified analyses, some ambiguities, e.g. regarding label alternatives, can also be represented by feature structures themselves as defined in ISO 24610-1:2006. However, in the database only those structures from ISO 24610-1:2006 can be represented which are a combination of (typed) feature

[18] Cf. Eckart (2009: p. 24) for an example.

structures and features. One could think of using specifically typed feature structures as values to emulate value alternatives.

structural_constraint. A structural constraint encodes structural ambiguity, i.e. ambiguity regarding attachment points. For syntactic analyses, an example would be ambiguities in the attachment of prepositional phrases. A structural constraint relates two nodes, (i) the root node of a graph fragment into which another fragment should be inserted and (ii) the root node of the fragment to be inserted. (i) is defined by the foreign key attribute pair (`gov_id`, `gov_graph`) and (ii) is defined by the foreign key attribute pair (`frag_id`, `frag_graph`). The way in which the two nodes are related is defined by the attribute `sc_type`. By means of a `CHECK` constraint, the attribute `sc_type` can take one of the following values:

BelowAppropriate the fragment with root node (ii) can be attached to any node of the fragment with root node (i) as long as the result is well formed with respect to a specific grammar,

BelowWithOptions the fragment with root node (ii) can be attached to a specified set of nodes,[19] which are part of the fragment with root node (i),

AttachmentOption the fragment with root node (ii) can be attached to the root node (i).

An entry of type *AttachmentOption* is only needed in relation with other constraints, cf. Eckart (2009: pp. 61f.) for an example. In case of *BelowWithOptions* the nodes which are part of the set are represented in table `sc_options`, and in case of *BelowAppropriate* the grammar can be specified as an object on the macro layer, by means of the foreign key attribute `grammar`. It is however not mandatory to define the grammar within the database, e.g. in cases where an appropriate external reference exists. Overall by the use of *BelowAppropriate*, further processing steps need to interpret which grammar has to be applied, and a structural self-containedness of the analysis is ruled out. It is thus recommended to use this option only in cases where it is enforced by the encoded analysis.

[19] In theory, the empty set is excluded, however this is not enforced by the database.

When the constraint is instantiated, this usually introduces an edge for the attachment. Depending on the framework, however, instantiating a structural constraint might mean to include additional nodes and/or several edges, cf. Kountz et al. (2008) for the case of structural constraints in constituency trees. In the simple case, i.e. when exactly one edge is added to mark the attachment, for this edge an annotation can be predefined and referenced by means of the foreign key attribute pair (`annot_id`, `annot_graph`).

Finally, an entry in the table `structural_constraint` is uniquely identified by its attribute `sc_id`.

sc_options. The table `sc_options` specifies the node sets for the structural constraints of type *BelowWithOptions*, which are specified in table `structural_constraint`. The nodes are referenced by the foreign key attribute pair (`node_id`, `node_graph`) and the structural constraint is referenced by the foreign key attribute `sc_id`. An entry of the `sc_options` table is only uniquely identified by the combination of all its attributes, which guarantees that none of the nodes erroneously appears in the same set twice.

labelling_constraint. Labelling constraints represent alternative labelling possibilities. In the database they are attached to a specific feature by means of the foreign key attribute pair (`f_name`, `f_part_of`) referencing table `feature`. The type of the labelling constraint is defined by the attribute `lc_type`. By means of a `CHECK` constraint, this attribute can take one of the following values:

LabelSet the feature value is one of the values from a specific set of values,

LabelSubtype the feature value is one of the atomic subtypes of a defined supertype,

Existence the feature is only instantiated in specific cases.

The *Existence* constraint applies when another decision triggers the existence of the feature, e.g. the existence of a morphosyntactic information which depends on the part-of-speech tag. In case of a *LabelSubtype* constraint, the attribute `supertype` refers to an entry in the `value_type` table, which can represent hierarchical label sets, cf. Section 4.5.7. All atomic labels which are subtypes to the specified supertype constitute the possible feature values. In case of a

LabelSet constraint values from table `value_part` refer to the unique identifier `lc_id` of the `labelling_constraint` entry. These values are the possible values referred to in the labelling constraint. However, the values themselves can be sets in the sense of table `value_part`. Then in the referring entries in table `value_part` the attributes `lc_id` and `set_id` are filled.

constraint_interdependency. Constraint interdependencies describe the cases where the instantiations of different constraints depend on each other. Each entry is a part of a full constraint interdependency and is uniquely identified by the attribute pair (`part_id`, `ci_id`). The specified part can either refer to a labelling constraint with values in table `value_part` (attribute `lc_id`) or to a structural constraint with the attribute `sc_id`. A `CHECK` constraint defines that only one of these attributes can be filled at the same time. The specified part has a role (`ci_role`) which is either `A`, precondition, or `B`, conclusion. A `CHECK` constraint is in place to guarantee that no other value can be set for `ci_role`. A third `CHECK` constraint defines the values *Preclude* or *Enforce* for the `ci_type` of the constraint interdependency: In the first case, the instantiation of all `A` parts of this constraint interdependency with a given value or set of values precludes the instantiation of the `B` parts with a given value or set of values. In the second case the instantiation of all `A` parts with a given value or set of values enforces the instantiation of the `B` parts with a given value or set of values.

In case of reference to a structural constraint, the instantiation is specified by the foreign key attribute pair (`node_id`, `node_graph`) defining the attachment point. In case of reference to a labelling constraint the value instantiation is specified by the foreign key attribute `vp_id` referencing the `value_part` table, and in case of a set-type value by the foreign key attribute `set_id`. The current implementation only takes cases into account where the value of the labelling constraint is in table `value_part`.

4.5.9 Indexes for efficient processing

Creating an index on an attribute or a set of attributes targets more efficient processing of queries which involve these attributes. Each primary key and

unique constraint automatically includes the creation of an index on the respective attributes, since tables are often joined on or queried by values defining a unique dataset. Furthermore primary keys and attributes with a unique constraint can be referenced by foreign key constraints, implying that many checks have to be conducted on their content.

However, since an index is updated with every related data manipulation statement, the decision to include additional indexes depends heavily on the way the tables are employed by the users. In cases where there is a need for highly frequent single statement transactions on a table, the cost of updating an index might exceed the gain for the querying. Consequently, the current schema only provides a small set of additional indexes, which have proven useful in the settings where the B3DB was applied so far. Nevertheless, users shall be encouraged to drop indexes which impede their work with the database.

Chapter 5

Introducing a workflow for task-based parser output combination

Chapter 5 presents the core contribution of this work: an abstract workflow for task-based parser output combination. In fact, the main idea is to define basic steps of a general workflow for this combination, independently of the actual task and of the available parsers. The abstract workflow is thus intended as a template, which can be applied to a concrete scenario including a well-defined task and specific parsing systems. Moreover, it aims at users who are not mainly concerned with parsing, but whose own interest is in some other task, which requires reliable syntactic information as input.

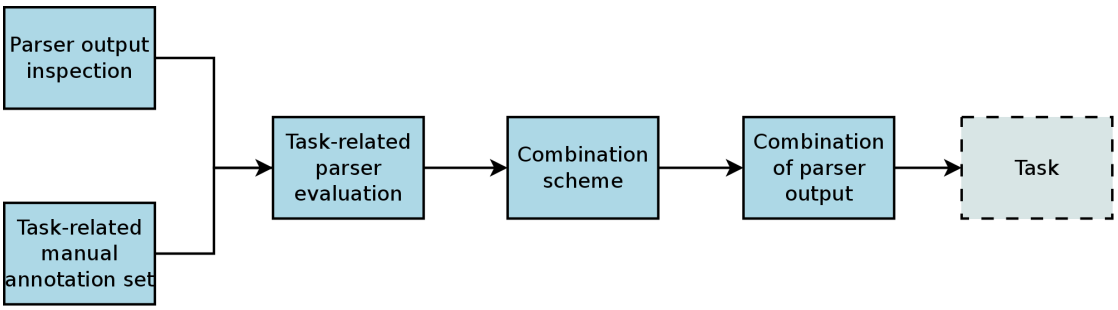


Figure 5.1: Abstract workflow for task-based parser output combination.

Figure 5.1 visualizes this workflow. Each box describes one step which has to be instantiated, and the directed edges introduce a chronological order of the steps. In the step of **parser output inspection** the users investigate how the syntactic phenomena relevant for their task are encoded by different parsers. The creation of a **task-related manual annotation set** is the basis for the evaluation of the involved parsers. Moreover it also formalizes the expectations of the users, which parts of the data capture the syntactic features relevant for the task. The **task-related parser evaluation** shows the performance of the single parsers with respect to the task-relevant features and based on this step, the users have to choose a **combination scheme**, which can apply e.g. (qualified) majority votes or rules. The last step is the actual **combination of the parser output** such that reliable features can be extracted for the task.

Section 5.1 discusses the prerequisites and the conditions under which the workflow can be applied, Section 5.2 explains each step of the workflow in detail and Section 5.3 discusses the applicability of the approach as well as situations not covered by it. Chapter 6 then presents two examples in which the workflow is instantiated for two significantly different tasks.

5.1 Prerequisites

For the workflow to be effectively applied, the following two assumptions need to hold:

(i) there is a clearly defined task which requires syntactic information,

(ii) there is a set of syntactic features relevant for the task which can be defined in advance.

In natural language processing many challenging analysis tasks fulfil these prerequisites and to exemplify them we include an outlook to the different tasks which will be discussed in the case studies in Chapter 6. The first task is a corpus study on German particle verbs with the particle *nach* (e.g. 'after'). It deals with the correlation between specific semantic types of such particle verbs and their realization of dative arguments and accusative arguments.

Precondition (i) is satisfied for this task, because the task can be clearly defined and it requires syntactic information to be able to identify dative arguments and accusative arguments. Furthermore it also needs syntactic information, as opposed to purely lexical and morphological information, to identify some of the particle verbs themselves, because in German verb final sentences, the particle is separated from the inflected form of the verb stem, cf. Example (5.1) where the verb *nachlaufen* ('[to] run after') appears with a separated particle.[1]

(5.1) *Der Hund* ***lief*** *dem Hasen* ***nach.***
the dog ran the hare after
'The dog ran after the hare.'

(Haselbach 2011: (1))

As to precondition (ii), the above description has shown that the task requires clearly defined syntactic features which are known and can be specified in advance: *nach*-particle verbs and their arguments in dative and accusative case. Thus this first task is appropriate to exemplify the instantiation of the combination workflow.

The second task is to annotate and inspect information status of noun phrases, determiner phrases and prepositional phrases: do phrases like *the man with the small dog* in Example (5.2)[2] in a given corpus instance refer back to previously mentioned entities (in the context of the example: *someone from the bar*), do they introduce new referents to the discourse, etc.?

The annotation scheme applied for the annotation of information status is hierarchical, i.e. it takes embedded phrases into account.[3] Example (5.2) shows a case where the embedded phrase *the small dog* is annotated with a label differing from the one of the larger phrase *the man with the small dog*.

[1] Cf. Section 6.1 for more details on *nach*-particle verbs.

[2] Cf. Baumann and Riester (2012) and Examples (6.30) and (6.32) on Page 263 in the description of Task II, Section 6.2.

[3] The annotation scheme applied in Task II is the RefLex scheme by Baumann and Riester (2012). Cf. Section 6.2 for a short introduction and Riester and Baumann (2017) for the annotation guidelines.

(5.2) [While chatting, we suddenly felt watched by someone from the bar.]
(The man with (the small dog)$_{\text{R-UNUSED}}$)$_{\text{R-GIVEN}}$ came up to us.

Here again, the task benefits from syntactic information: in this case from information on phrases and their embedding. The relevant syntactic features are the boundaries of phrases and sub-phrases, and are thus able to be specified in advance. Therefore the second task also complies with the prerequisites (i) and (ii) for the application of the combination workflow.

5.2 Workflow steps

In the following, each step of the abstract workflow shown in Figure 5.1 is explained in detail. These are the steps which users need to instantiate the parser output combination with respect to their specific task at hand. Thereby, the users do not have to develop or modify a parser, but they are guided along these steps to receive more reliable information by utilizing several off-the-shelf systems.

Note that the users will have to parse their data set with all parsers they want to include in the combination.[4] Since this can happen at different points in the workflow, it is not represented as a separate step there. The step of parser output inspection does not necessarily have to be conducted on the task-related data set, however data from at least the same domain might give a better insight on how a parser annotates domain-specific phenomena. The step of task-related parser evaluation will require at least some part of the data set to be processed by all parsers, and a fully parsed version of the whole data set has to be available for the step of combination of parser output at the latest.

The first two steps, parser output inspection and the construction of a task-related manual annotation set, are in no particular chronological order, since they focus on two distinct aspects of the scenario. These distinct aspects are the parsers on the one hand and the data set on the other hand. For the involved parsers, it has to be explored how the syntactic features needed for the task are annotated by each parser (Section 5.2.1). For a small part of the data set to

[4] That is, unless a parsed version of the data set is already available somewhere.

be parsed, the features needed for the task should be manually annotated to provide data on the information types which are expected to be retrieved from the parser output (Section 5.2.2).

In practice, it might often be the case that the data set is available before a decision has been made which parsers will be used. Moreover, building the task-related manual annotation set in advance will guarantee that the guidelines are not influenced by the capability of the parsers but reflect the expectations of the users only. However, inspecting the parser output first leads to a more realistic estimate of what kind of information the parser output can provide, what information it can be mapped to and evaluated against. This is especially so if the users are not familiar with the grammar (formalism) or annotation systems underlying the applied parsers.[5] Early inspection of the technical aspects of the parser output might spare an iteration in cases where the expectations expressed in the manual annotation cannot be met by any of the available parsers.

Thus, Section 5.2.1 starts with a focus on the parser output and the need for a small task-related manual annotation set is then discussed in Section 5.2.2.

5.2.1 Parser output inspection

Since automatic parsing tools originate from different traditions and research paradigms, the annotation of the same or closely related phenomena may take different shapes in different parsing tools. Thus it is important to inspect the way in which the involved parsers annotate the syntactic features relevant for the task; cf. Section 2.1.4 for a description of the parsers applied in the case studies of this work and Section 3.5 for a discussion on interoperability of multiple annotations with respect to a task. The following shows some typical examples where parsers might differ in their annotation.

First of all the applied framework influences the available information. In this work, dependency parsers as well as constituency parsers are applied in the

[5]Of course, the inspection of the parser output must not happen on data which will be part of the manual annotation set, especially if the step of creating the manual annotation set is conducted after the step of parser output inspection. The manual annotation set is applied in the evaluation of the single parsers later and should not be biased by an already encountered analysis of the same sentence.

case studies. A difference on this framework level is that in constituency parses, the head of a phrase is often not explicitly marked, while in dependency parses there is a head in each dependency relation. In dependency parses on the other hand, there is usually no categorisation of phrases, i.e. it is easier, for example, to extract all noun phrases from a constituency tree, than from a dependency analysis.

Second, there might be different theoretical decisions within a given framework, e.g. distinguishing two constituency parsers or two dependency parsers, and thus different representations of the phenomena in the parser output. In the following we repeat some of the phenomena and their different annotations, which were discussed in Sections 2.1.3 and 2.1.4, along with parts of the example sentences from Section 2.1.4, repeated as Examples (5.3) to (5.5) here.[6]

(5.3) *Stroh-Engel und Behrens legten nach.*
Stroh-Engel and Behrens layed after
'Stroh-Engel and Behrens scored also.'

(5.4) *Kempe konnte gegen St. Pauli punkten.*
Kempe could against St. Pauli score
'Kempe could score against St. Pauli.'

(5.5) *[. . .] 2009 schoß er zwei Tore für die deutsche Elf.*
2009 kicked he two goals for the German eleven
'[. . .] in 2009 he scored two goals for the German football team.'

With respect to dependency parses, there are certain constructions for which there is no consensus in the theory of dependency grammar on how they should be annotated, cf. Kübler et al. (2009: p. 5). These constructions often include function words, and the main question is usually which part of the construction should constitute the head. Thus, for these constructions there are several options to annotate them, adopted by the different parsers. Two typical examples, shown in the following, are prepositional phrases and coordinations.

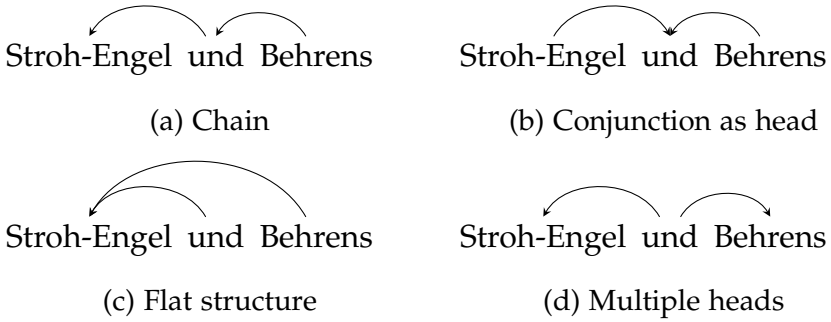
[6]The made-up example sentences from the sports news domain were processed by the same parsing pipelines as stated in Section 2.1.4.

(a) Lexical head (b) Functional head

Figure 5.2: Two dependency analyses for a prepositional phrase.

Figure 5.2 shows the same prepositional phrase from Example (5.4) with a lexical head in Figure 5.2a and a functional head in Figure 5.2b.[7] While there is a relation between the two tokens in both analyses, the instance which is the head differs and thus the direction of the edge.

An example for the range of different annotations for the same phenomenon is the annotation of coordinations. Figure 5.3 shows several ways how a simple coordination of two nouns with a coordinating conjunction, such as in Example (5.3), can be annotated by a dependency parser.



(a) Chain (b) Conjunction as head

(c) Flat structure (d) Multiple heads

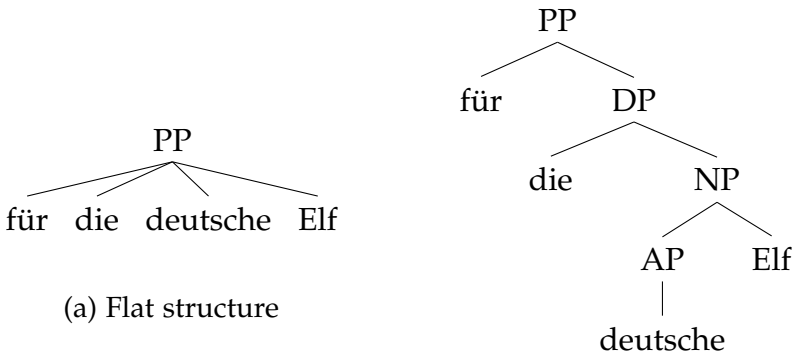
Figure 5.3: Different dependency analyses for a nominal coordination.

Figure 5.3a is a chain, with the first conjunct as the head of the construction, cf. Kübler et al. (2009: Figure 1.3, second example) and Tsarfaty et al. (2011: (2c)). This chain construction is also used in the mate parser analyses, cf. Section 2.1.4. In Figure 5.3b the conjunction is the head of the coordination, cf. Kübler et al. (2009: Figure 1.3, first example) and Tsarfaty et al. (2011: (2a)). The construction in Figure 5.3c, where *Stroh-Engel* is the head and both, the conjunction and the

[7] The named entity *St. Pauli* is treated as one token here; edges are directed from the dependent to the head as described in Section 2.1.3.

second conjunct are direct dependents of the head, is called flat by Tsarfaty et al. (2011), as opposed to the chain structure, which they call the nested structure. Furthermore, there is a structure with two heads in Figure 5.3d, which is applied by FSPar, cf. Section 2.1.4, and which may or may not be accepted as a valid dependency structure, depending if a constraint is imposed which requires all dependents to have only one single head, cf. Section 2.1.3. However one could think of even more possibilities to relate the three words in this construction, e.g. choosing the last conjunct as the main head of the construction instead of the first conjunct in Figures 5.3a and 5.3c.

For constituency parses, a typical difference between individual annotation schemes is the depth of the embedding.



(a) Flat structure

(b) Deep structure

Figure 5.4: Excerpts of two constituency trees for the same prepositional phrase.

Figure 5.4a shows a flat structure, without any substructure in the prepositional phrase, while Figure 5.4b represents the same structure with several embedded phrases.[8] The flat representation is based on the analysis of the IMS-SZEGED-CIS parser, while the deep representation is based on the LFG parser analysis.[9]

[8] Figure 5.4 is a repetition of Figure 3.2 on Page 124. The tags applied in Figure 5.4 are: PP – prepositional phrase, DP – determiner phrase, NP – noun phrase, AP – adjective phrase.

[9] The complete LFG parser analysis for this prepositional phrase even contains some more embeddings by unary branches, cf. Figure 2.25 on Page 97.

So far, the examples only referred to structural variations. However there are also differences in the tagsets, for example with respect to their granularity.

In the analyses of Example (5.3) in Figure 5.5[10], the separated verb particle *nach* is related to the inflected form of its base verb *legen* by both, FSPar and the mate parser. However, the relation is labelled RK by FSPar (Figure 5.5a, token 4), but SVP by the mate parser (Figure 5.5b, token 5).

In our case, the mate parser has been trained on a dependency conversion of the TIGER treebank, thus SVP is the tag for a *separated verb particle*, while RK in the FSPar analysis denotes the *rechte Satzklammer* based on the topological field model for German sentences.

<s>

0	Stroh-Engel	NN	Stroh-Engel	3	NP:1
1	und	KON	und	0|2	KON
2	Behrens	NE	Behrens:H	3	NP:1
3	legten	VVFIN	nach#legen	-1	TOP
4	nach	PTKVZ	nach	3	RK
5	.	$.	.	-1	TOP

</s>

(a) FSPar

1	Stroh-Engel	Stroh-Engel	NN	4	SB
2	und	und	KON	1	CD
3	Behrens	Behren	NN	2	CJ
4	legten	legen	VVFIN	0	–
5	nach	nach	PTKVZ	4	SVP
6	.	–	$.	5	–

(b) mate parser

Figure 5.5: Dependency analyses of Example (5.3).

While a separated verb particle is usually in the *rechte Satzklammer*, not every *rechte Satzklammer* is a separated verb particle. Thus, when extracting dependency relations denoted RK from FSPar, only a subset of the results are verbs with separated particles, and this subset is not identifiable from the dependency label alone.

However, additional information is available from the lemma and part-of-speech annotations in the FSPar pre-processing pipeline. FSPar (as well as the

[10]Morphology and empty columns are omitted for readability reasons.

mate parser) apply versions of the STTS (Schiller et al. 1999) as tagset for part-of-speech annotations, thus a recognized separated verb particle is annotated with PTKVZ. Furthermore FSPar makes use of additional lexicons, such that the lemma of the base verb is already marked as a particle verb (Figure 5.5a, token 3, '#'-character in the lemma column, which separates particle and base verb) and thus the lemma of the full particle verb is given in the annotation of the base verb. For the mate parser, the full lemma has to be reconstructed from the lemma of the base verb (Figure 5.5b, token 4) and the separated particle. In summary, both analyses encode the same information, however by means of different annotation layers.

Another discriminative factor for parser output are the knowledge bases which are accessible to a parser. In the above example of the particle verb, FSPar's tokenizing lexicon supplies information about the full lemma. While this information can also be constructed from the full parsing output of the mate parser, there are cases where FSPar's lexical knowledge base provides additional information, e.g. in the case of named entities which denote locations, companies or associations.

In Figure 5.6a[11] the lemma of the token *St. Pauli* includes the information that it can either denote a city[12] ("Sankt_Pauli:Stadt") or its football club ("1._FC_St._Pauli"), of which the latter is true in the given context of the sentence. The mate parser output encodes the fact that it is a named entity (part-of-speech tag NE), but cannot provide any more information. This applies regardless of the way the name is tokenized: Figure 5.6b shows the analysis of a standard whitespace tokenization, whereas Figure 5.6c enforces a combined token.

Of course, even a parser with a lexical knowledge base can only provide information on entities which are actually found in its lexicon. Thus, in cases where information from a lexical knowledge base which is encoded in the parser output might enhance the extraction of task-related features, it is helpful

[11] In Figure 5.6 morphology and empty columns are also omitted for readability reasons.
[12] *St. Pauli* is a district of the city of Hamburg, to be precise.

```
3  St._Pauli  NE  1._FC_St._Pauli|Sankt_Pauli:Stadt  2  PCMP
```

(a) FSPar

```
4  St.    St.    NE  5  PNC
5  Pauli  Pauli  NE  3  NK
```

(b) mate parser,
whitespace tokenization

```
4  St. Pauli  St. Pauli  NE  3  NK
```

(c) mate parser,
combined token

Figure 5.6: Information on named entity for Example (5.4).

to find out about the coverage of a lexicon, to get an idea which information can be expected to appear in the parser output.[13]

The last aspect to be mentioned here with respect to the differences of how syntactic features are represented in parser output, is the degree of ambiguity resolution. As discussed in Section 2.1.4, the output of FSPar is underspecified with respect to attachment as well as label ambiguities, while the other parsers applied here return one or several disambiguated analyses. While the former leaves the full disambiguation to further steps, the latter might have lost the correct analysis in a step of forced guessing. Sometimes however no full disambiguation is necessary to extract the task-related features, as will be shown in Case study I.i (cf. Section 6.1.3).

In summary, the step of parser output inspection gives insight on how the task-related features are encoded. Furthermore it also shows whether and in which detail they are represented in the analysis, because obviously, information which is not encoded in the analysis cannot be extracted. While this is a trivially reasonable statement, there might be a strong intuition by the users about the information which they consider extractable from a syntactic analysis. A

[13]For example, names of companies which were founded after the last update of the lexical knowledge base, cannot be annotated respectively by the parser.

thorough inspection of the output of a specific parser might in fact contradict this intuition and is thus an important step.

5.2.2 Task-related manual annotation set

The task-related manual annotation set is created for two purposes. On the one hand it will be used in the parser evaluation step to track how well a specific parser works on the features relevant for the task. On the other hand, it helps to define the minimal and maximal expectation regarding the features to be retrieved from the parser output. Together with the parser output inspection described in Section 5.2.1, building a reference data set with manual target annotation will not only provide a basis to evaluate the parsers against, i.e. how good the parser is in recognizing e.g. the respective structure, but also to check if the information needed for the task can at all be produced by a parsing system.

The aspect of minimal and maximal expectations can be exemplified by the span of tokens to be extracted for a specific feature. Let us assume that the syntactic feature we need for a task is to know the subject of every main verb in the input. Then the users have to specify if reliable information on the head of the subject is sufficient, or if the whole span of the subject noun phrase, including embeddings, relative clauses, etc. is needed for the task. In the former case, it is also relevant how the head of a phrase is defined, cf. Section 5.2.1. When subjects are extracted, the lexical head might be more important, while in the extraction of prepositional phrases, the information about the form of the preposition, i.e. the functional head, might be sufficient. In other cases, only the whole token span may contain all relevant information for a given task: an example is the extraction of subjects from a sentence with a coordinated subject, as in Example (5.3). Probably neither the functional head *und* nor one of the possible lexical heads *Stroh-Engel* or *Behrens* will be sufficient, as both are somewhat artificially chosen within the dependency framework.

The manual annotation set is utilized to evaluate the single parsing systems, cf. Section 5.2.3, and to then decide on a fitting combination scheme, cf. Section 5.2.4. Thus the users are obliged to view the data also from the “perspective

of the parsers". The creation of such a manual annotation set includes a step which forces the users to define their required features based on existing data. This can bring up additional insights, e.g. that part of the required features cannot be retrieved by a parsing system as such, but need additional information as well. Staying with the example on extracting subjects from above, it might be necessary for the task to separate animate and inanimate subjects, which on the one hand comes with a preference for the extraction of lexical heads and on the other hand with the requirement of an additional semantic annotation step between the extraction of the syntactic features and the task.

Another objective should be to find out early in the workflow, when the parsing step cannot provide the necessary information. For example, in cases where the needed information is more often than expected found in the discourse context rather than in the sentence context. In such a case the information might not be available from the parser output, since most parsers take only the context of one sentence into account.

Creating the manual annotation set is, as the name conveys, manual effort. Thorough manual annotation is usually very time consuming, because annotation guidelines have to be defined, the annotation has to be carried out by several people, the inter-annotator agreement has to be computed and the annotations have to be merged. Furthermore there tend to be several iterations in the process, during which the annotation guidelines are refined, cf. Lemnitzer and Zinsmeister (2015: pp. 103f.).

For a complete syntactic annotation, this effort might not be outweighed by the gain obtained from the parser combination. However, when this manual effort is restricted to the features needed for the task, the step of creating a task-related manual annotation set is much faster, and, as will be shown in Section 6.1, even a small manual annotation set can be sufficient to increase the overall reliability of the task-specific features. Furthermore, already a small manual annotation reduces a typical bias, since it is easy to confirm an annotation as correct if the expectation has not been formalized in advance. In difficult cases, the small set can serve as a starting point: if the results of the parsers deviate on a grand scale from the expectations, annotating more data can be helpful to identify and classify different cases.

5.2.3 Task-related parser evaluation

In the step of creating a manual annotation set, cf. Section 5.2.2, the question was if parsing as such is appropriate to gain the task-related information. In the task-related parser evaluation step the questions are (i) whether the specific parsers which are available to the users are in principle capable of annotating this information, and (ii) how reliably each of the single systems does this.

An answer to question (i) will help to decide which parsers to include in the combination step, and an answer to question (ii) will help to choose the right combination scheme, cf. Section 5.2.4. This section describes the steps it takes to conduct a task-related parser evaluation which will answer these questions.

In a first step, all parsers which the users would like to include in the combination should be run on the data set chosen for the manual annotation. This might mean that the data set has to be converted into an appropriate representation format for each parser, since different tools might use different input formats. Furthermore the parsers might require a different amount of preprocessing, cf. Section 2.1.1.

Furthermore, it is of major importance to introduce an identification scheme for the input segments, such that the results of the parsers can be related via this identifier. Usually the identification scheme will be a sentence numbering. However, there are different approaches to sentence segmentation within the different parsing systems, e.g. in the case of colons or semicolons. Some parsers take a given set of input symbols as input expression and try to include all symbols into the analysis, other parsers introduce additional segmentations of the input expression on their own and cannot be forced to include a specific set of input symbols into one structure.

Thus, even if the parsers introduce a sentence numbering, it is not guaranteed that output structure number X from parser P_1 covers any of the input symbols included in output structure number X from parser P_2.

Figure 5.7 shows part of the analyses of Example (2.10) from Page 70 by the mate parser and FSPar. While FSPar's own segmentation step builds two separate structures from the input (cf. Figure 5.7a), the mate parser keeps the

token and sentence boundaries originating from a user-defined tokenization step, and tries to include the respective input symbols into one structure (cf. Figure 5.7b).

```
<article cluster=0 address=0 date=4>
<s>
0   Wagner
1   war
2   bereits
3   bei
4   der
5   U-21-Europameisterschaft
6   erfolgreich
7   :
</s>
<s>
0   2009
1   schoß
2   er
3   zwei
4   Tore
5   für
6   die
7   deutsche
8   Elf
9   .
</s>
</article>
```

(a) FSPar

```
4_1   Wagner
4_2   war
4_3   bereits
4_4   bei
4_5   der
4_6   U-21-Europameisterschaft
4_7   erfolgreich
4_8   :
4_9   2009
4_10  schoß
4_11  er
4_12  zwei
4_13  Tore
4_14  für
4_15  die
4_16  deutsche
4_17  Elf
4_18  .
```

(b) mate parser

Figure 5.7: Sentence segmentation: token position/ID and word form from analyses of Example (2.10).

To convey identifiers for the input segments, it might thus be necessary to either use additional markup which will not be corrupted by the parsing step, or have an additional processing step after the parsing to include the identifiers needed for the combination step. In Figure 5.7a the markup for article borders of FSPar is utilized to include the original sentence numbering into the attribute "date". For the mate parser, the token ID column can be used to include the sentence number in front of the token number making use of an additional separator, e.g. an underscore as in Figure 5.7b. This additional

information, introduced after the tokenization, is kept during the parsing step without influence on the parsing process.

Different parser output might also be based also on different tokenizations, however, the sentence numbering mechanism reduces the mapping problem for tokens to the span of one sentence. Furthermore, for several tasks no full mapping of the tokens is necessary, but only an extraction of the tokens constituting the task-related features.

In a second step, the relevant syntactic features have to be extracted from the result set of each parser involved in the combination. Information on how this can be done for a specific parser is based on the information gathered in the parser output inspection step, described in Section 5.2.1, e.g. which label is used to mark the subject.

In the third step, the actual evaluation takes place, when the extracted task-related features from each parser are compared to the manual annotation, e.g. by means of precision and recall as described in Section 1.2.8. However, as discussed in Section 5.2.2, the features extracted from the parsers might fit the optimal expectation to different degrees, such as returning only the head, the full span with all embeddings, etc. And since the manual annotation set has been created based on the needs of the task and not according to a specific annotation scheme of one of the parsers, it is possible that none of the parsers displays the needed features exactly as they are represented in the manual annotation set. In this case it has to be defined how the features of a single parser are compared to the expected features marked in the manual annotation set. One option is to compute the evaluation scores according to different aspects of the features, for example how well a parser performs when a subject is extracted (i) by means of its head (i.e. is the extracted head part of the expected span?), or (ii) by means of its full span. Another option applies especially when the ambiguity handling of the parsers plays a role in the evaluation: In the case that there are underspecified structures or labels, it can be useful to do several evaluations, taking or not taking information from the underspecified analysis parts into account, cf. Section 6.1.2 for an example.

5.2.4 Combination scheme

The combination scheme defines how to combine each task-relevant feature from the different parser outputs into a single target feature applied in the task which is more reliable than the individual features. The approaches to the design of such a combination scheme presented in the following are based on (qualified) votes of the parser output or on manually set up combination rules. Examples are shown in the case studies in Chapter 6, and related work regarding combination schemes, including additional approaches, is discussed in Section 2.2.

A simple combination scheme is majority voting, where the majority of the parsers has to agree on the extracted feature. Let the needed feature be again the subjects in the input data. In case that *Stroh-Engel und Behrens* is identified as the subject of the above example in the majority of parser outputs, this feature is passed on to the set of extracted features for the task. However, majority voting needs the possibility to build a majority, e.g. an odd number of parsers, or some heuristic for breaking ties.

An enhanced version of majority voting is weighted voting. In weighted voting the statements from each parser output get a weight, e.g. according to their reliability. For each feature the weights of the single statements are combined, and the statement with the highest weight is added to the set of extracted features. The weights result from the parser evaluation on the manual annotation set, or can be trained on this set, such that the optimal weighting scheme is found. Weighted voting can of course also include thresholds, such that in cases where all statements only receive weights below a certain threshold no feature is extracted at all.

Another way to define a combination scheme is to manually define combination rules based on the results of the task-based parser evaluation. An advantage of this approach is that also an even number of statements from the parsers can be applied and combined in a specific way. Designing a rule-based combination scheme requires some additional manual effort, but the scheme can be tailored more explicitly to the task and the results of the evaluation.

A basic assumption of the combination approach is that different parsers come with different error distributions, which will support the correct information in the combined output. Thus with the information from the parser evaluation, a strategic combination of parsers is possible, even if none of them alone returns exactly and completely the required feature. Assume that the features needed for the task be again the subjects of each main verb, but this time the full span is required, and the setting is precision-oriented. Let then there be a parser which, according to the evaluation, reliably returns the lexical head of the subjects, but tends to fail with the selection of the required span, and a second parser which does better with the spans, but not so well with the identification of the lexical head. In a precision-oriented setting, the combination allows to select those spans from the second parser where both parsers agree on the head, thus returning reliable spans, by including the information of a reliable lexical head.

Additionally, the combination scheme can already define fall-back strategies, when for a specific input segment information from one or more systems is missing. Depending on the setup, in these cases a fall-back strategy might take the feature from the most reliable single system into account or switch from a voting scheme to a set of combination rules.

5.2.5 Combination of parser output

The last step in the workflow, before the reliable features can be passed on as input for the task, is the actual combination step. This step requires an implementation of a combination scheme decided upon, cf. Section 5.2.4.

As mentioned in the introduction to the workflow steps in Section 5.2, this is also the latest point where the whole data set has to be processed by all parsers, such that the task-related features can be extracted.

The script or tool which implements the combination might either take the complete output of each parser for the same input expression, extract the features according to the annotation of each parser and apply the combination scheme. Another option is to already provide the features extracted from the single parser outputs as input for the combination, especially if the features have

been extracted from a database, cf. Chapter 4, or via an annotation query tool such as ICARUS (Gärtner et al. 2013) or TIGERSearch (Lezius 2002). Depending on the formats in which the parser output is provided, both options may apply within a set of parsers chosen for the combination.

Implementations for different combination scheme instantiations are exemplified in the case studies for Task I (Section 6.1) and Task II (Section 6.2).

5.3 Discussion of the workflow and its applicability

To summarize, this chapter presented the single steps of a workflow for task-based parser output combination on an abstract level. The next chapter demonstrates how this workflow can be instantiated, by means of two different example tasks: a precision-oriented task in Section 6.1 and a recall-oriented task in Section 6.2.

From the abstract description of the workflow, some properties of the workflow are evident. First, its template structure makes it adaptable to different tasks, in case the tasks fulfil the prerequisites defined in Section 5.1: (i) the task is clearly defined and requires syntactic information, and (ii) the relevant syntactic features can be defined in advance. If these two requirements are satisfied, it is not only evident from the task definition that syntactic information is needed, but also which information. Tasks where the task-relevant syntactic features are not known in advance and for which e.g. several syntactic features need to be tested for their benefit, might produce an overhead in the workflow: the parsers have to be inspected with respect to a larger set of features, and the manual annotation set has to capture more possible cases to evaluate against. If the trade-off between annotation work and expected gain of the workflow is still adequeate has to be decided based on the specific case. However, if the data set is from a very specific domain, on which the single parsers are expected to perform poorly, it might still be worth the effort of inspecting the parsers closely and providing a larger set of manual annotations.

The second property of the abstract workflow description is that the single steps build on each other by means of the sequence in which they have to be carried out, but are self-contained in the sense that they do not overlap. The users are thus provided with an explicit recipe, where the task influences each workflow step, but not the way in which the steps are combined.

The third property is that the workflow allows the users to treat the parsers in the way of a black box: only input and output of the system must be accessible to the users, and they do not have to adapt or manipulate the source code of the parser or its configuration. While it is of course possible to include systems which can also be internally adapted to the tasks by the users, this is not necessary for the application of the workflow. In this way, the workflow allows for the adoption of very different parsers, as well as for much flexibility in switching parsers or migrating to newer parsing systems. The users can employ off-the-shelf parsers or models and thereby exploit information and reuse resources which are already available. Furthermore they neither need to be parsing experts to trim the parser to a specific task, nor do they have to spend time to produce a fully annotated training set for the relevant domain.[14]

While the workflow allows users to reuse available parsing systems, an instantiated workflow is tailored to a specific task and a specific set of parsers. This means that switching a parser or including an additional one for the same task leads to additional work in the steps of parser output inspection, task-related parser evaluation and the combination scheme (bold arrow in Figure 5.8). Changing the task might even entail reinstantiating the whole workflow, depending on the overlap of the syntactic features needed for the old and the new task. However, extending the dataset with data from the same domain as before allows to skip the steps up to the actual combination and reuse the existing combination scheme (dashed frame in Figure 5.8).

It is also possible to apply a shortcut by skipping the steps of building a task-related manual annotation set and of task-related parser evaluation (dotted arrow in Figure 5.8). This short version of the workflow however restricts the combination scheme to a majority vote, since the task-related performance of

[14]Sections 6.1.3 and 6.1.4 show case studies where parsers from the news domain are successfully utilized in an output combination performed on web data.

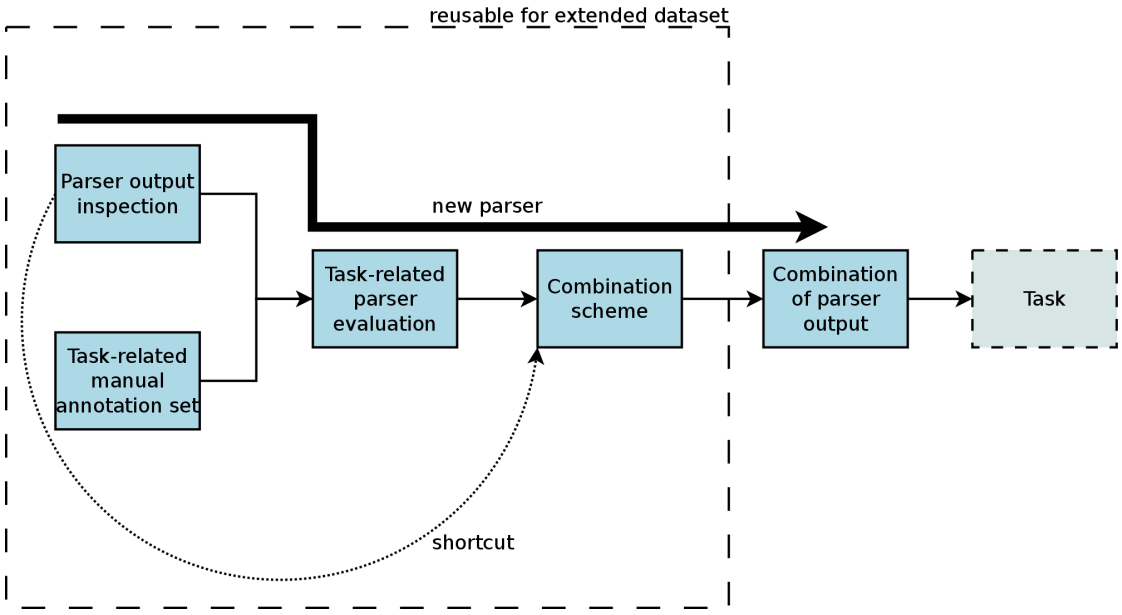


Figure 5.8: Alternative routes in the workflow.

the parsers is not known, and thus neither a weighting scheme nor combination rules can be defined.[15] Additionally, the definition of the acceptable range of returned features and the mapping of extracted features from the parser outputs to these expectations have to be included e.g. in the step of the combination scheme. While this shortcut restricts the applicability of the workflow, it fast-tracks its instantiation, e.g. for pilot studies. However it is to be expected that the set of parsers has to consist of sufficiently different parsers for any useful results to be obtainable in this setting.

Although the workflow has to be instantiated with new features for each task, the major advantage of the task-based approach is that the complex aspects, such as the interoperability of the annotation schemes of the parsers and the disambiguation of syntactic structures, are limited to those analysis parts which are directly relevant for the task. In most cases neither a full mapping of the outputs nor a full disambiguation will be necessary. Additionally, it can be

[15]This is the case unless the performance of the parsers for specific features on the relevant domain has been reported in the literature or can be assumed for other reasons.

assumed that there are several syntactic features which are relevant for more than one task.

Taking the handling of ambiguity closer into account, it is not only reduced to the task-relevant parts of the analysis, but can also play an important role for the relevant features themselves. Case study I.i (cf. Section 6.1.3) makes use of the fact that only the existence of a specific argument has to be known, not its actual span in the analysis, which can thus stay underspecified. Going one step further, also the information that an extracted feature was subject to disambiguation can be included in a combination scheme. In a weighted voting approach in which the weight reflects the confidence of an extracted feature, unambiguous parts of the analysis might receive different weights from parts where an ambiguity could not be fully resolved without forced guessing. Similarly to Case study I.i, different systems can be combined to make an informed decision on ambiguous cases, however this would profit especially from the availability of several parsing systems which allow for ambiguity information in their output. This can be done by applying underspecification, such as in FSPar, or by at least reporting the confidence of decisions from forced guessing, or a ranked list of analysis alternatives. This is thus also a call for more transparency regarding parser decisions. However, spelling out all alternatives for the whole analysis easily becomes excessive and underspecified output also poses challenges to further processing. Thus, again here a task-based focus for a narrowed set of features makes it easier to exploit the available information on ambiguity within an analysis, rendering the application of ambiguity-preserving systems in the workflow proposed in this work a promising combination.

And to come back to the beginning, even if the prerequisites for the application of the workflow are not met, for example because the current task is to build an annotated resource, based on which several studies will be conducted, it can be beneficial to provide multiple syntactic annotation layers from different parsers. Such a resource will provide future users with the possibility to instantiate a combination for their task at hand (cf. Eckart and Gärtner 2016).

Chapter 6

Case studies

We have motivated the idea of task-based output combination in Section 1.3, discussed aspects of interoperability between different systems taking part in a combination in Chapter 3, presented supporting infrastructure in Chapter 4 and introduced the abstract workflow of task-based parser output combination in Chapter 5. In this chapter we will now instantiate the abstract workflow with several case studies which concern two different tasks.

The two example tasks utilized in this chapter are based on project work conducted in the collaborative research centre SFB 732[1] and are thus not developed for the purpose of this thesis, but are linked to research questions encountered in independent work contexts. To emphasize that our workflow is applicable to a wide range of tasks, Task I is more precision-oriented while Task II focusses on recall.

Task I deals with German *nach*-particle verbs. In this task, information on the argument structure of these verbs is needed to inspect to what extent a theory on their behaviour at the syntax-semantics interface is supported by evidence from web corpus data. Task II addresses annotation of syntactic phrases, which serves as the basis of an information status annotation according to a hierarchical annotation scheme. This second task is carried out on the basis of a German radio news corpus.

[1] Cf. Section 1.1 for more information on SFB 732.

Sections 6.1 and 6.2 are dedicated to these two tasks; both sections have the same structure, in order to provide an easy overview of the tasks, of the language resources involved and of the conducted case studies. Each time, we describe the task as such in more detail, focussing on its own objectives apart from those of the combination approach. Since this chapter makes extensive reference to experimental data, we also discuss the respective resources involved, including a short summary stating which parsers and parser configurations have been applied in the case studies. A more detailed description of the parsers has already been given in Section 2.1.4. We spell out the technical workflow of the overall task, describe the performance of the parsers when they are applied individually, i.e. not in a combination approach, and present the different case studies, ordered by their combination type according to Section 3.3. At the end of each task section, we discuss the effects of the different combination types and combination schemes utilized in the case studies.

In Section 6.3 we compare the case studies of the different tasks with respect to their combination types and applied combination schemes. Based on this, we conclude that the combination of parsers is more effective if differing tools are involved, deviating with respect to their underlying theory or their processing techniques. Moreover, this finding strongly supports our task-based approach, which even allows to combine systems which cannot be combined in a full mapping approach due to the different linguistic theories and implementation strategies these systems are based on.

The tasks described here are part of project collaborations in SFB 732. Task I is joint work with Boris Haselbach, Wolfgang Seeker, Kurt Eberle and Ulrich Heid and has been published in Haselbach et al. (2012a) and partly in Eckart and Seeker (2013).[2] A pilot study not taking any combination settings into account has been published in Haselbach et al. (2012b). Task II is joint work with Arndt Riester.

[2]These papers utilize settings similar to Case study I.i, cf. Section 6.1.3.

6.1 Task I: Automatically reproducing classes of *nach*-particle verb readings on web corpus data

Haselbach (2011) presents theoretical linguistic analyses of German *nach*-particle verbs at the syntax-semantics interface. Haselbach relates the interpretation of the verbal particle *nach* ('after') to the interpretation and the argument structure of the base verb the particle is combined with. In Task I, we test his hypotheses against data from the "real world", i.e. a huge corpus of sentences from the web. In general, the approach of testing a theory against an independent data set might help to see if the data, processed by some natural language processing techniques, corroborate the predictions of the theory, and it might also help to pinpoint potential shortcomings in the theory-based account of the data. Therefore we apply this approach, trying to reproduce the theoretically motivated class distinction from Haselbach (2011) on a web corpus. In the next two paragraphs, we will get familiar with some aspects of the theory: the different *nach*-readings and the related argument structure. Thereafter the idea of how to test this theory is presented.

Readings of *nach*-particle verbs. According to Haselbach et al. (2012b) German particle verbs are in a syntactic and semantic grey area with respect to their argument structural behaviour, which might differ from that of the underlying verbs, i.e. the base verbs from which the *nach*-particle verbs are derived. In addition, the particle *nach* comes with several readings, depending on the base verb it combines with, e.g. a temporal one as in *nachfeiern* ('[to] celebrate later'), a directional one as in *nachlaufen* ('[to] run after'), an intensifying one as in *nachdenken* ('[to] reflect'), a continuative one as in *nachreifen* ('[to] continue ripening'), etc. (cf. Haselbach 2011; Haselbach et al. 2012b). The other way round, also a single *nach*-particle verb can be ambiguous with respect to its reading, e.g., *nachtanzen* might be interpreted, amongst others, as '[to] copy someone's dancing' as well as '[to] follow someone (by) dancing' (cf. Haselbach et al. 2012b).

Haselbach (2011) introduces a partial classification of *nach* for five of these possible readings. As for the semantics of *nach*-particle verbs, he uses DRT (Discourse Representation Theory, cf. Kamp and Reyle 1993; Roßdeutscher and Kamp 2010). As for the morphology of *nach*-particle verbs, he uses Distributed Morphology (Halle and Marantz 1993). For a detailed description of an approach combining DRT for semantic interpretation and Distributed Morphology for morphological realization, see Haselbach (2017). We do not go further into the details of this modelling here, but focus on the different readings and their presumed argument structure, which are both relevant for the task.

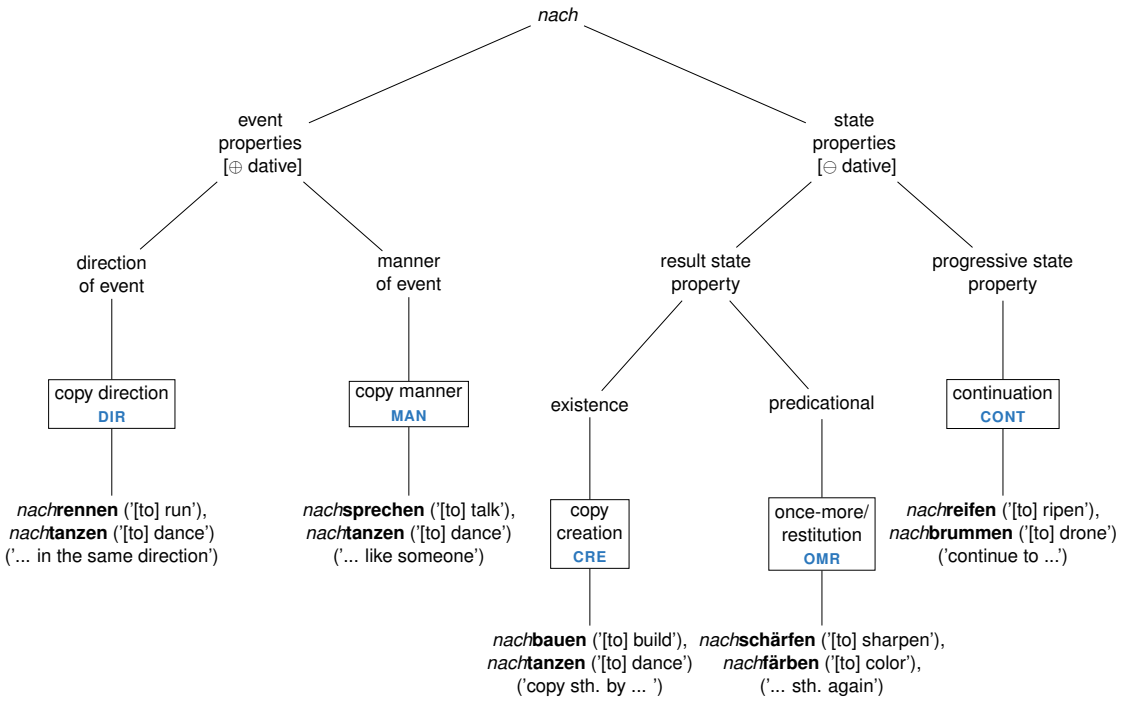


Figure 6.1: Partial classification of *nach* readings by Haselbach (2011).

Figure 6.1 shows the five *nach* readings and Haselbach's hierarchical classification.[3] The first distinctive criterion is based on properties accessed in the verbal phrase. Haselbach argues that the particle *nach* introduces a temporal

[3]Cf. the respective figures in Haselbach (2011); Haselbach et al. (2012a,b).

relation between an asserted eventuality and a presupposed eventuality; either two events or two states. Properties of the asserted eventuality are copied and ascribed to the presupposed eventuality. In the next step of the classification the copied event properties are distinguished according to whether the direction or the manner of the event is copied. *Jemandem nachschleichen* ('[to] follow someone in a sneaky manner'), does not necessarily presuppose, that the person being followed was also sneaking; rather it suggests that the follower is moving in the same direction. On the other branch of the tree, the state properties are distinguished into progressive state properties and result state properties. The result state properties are then again split into existential and predicational properties. In the case when existential properties are copied, a second (concrete or abstract) object is introduced into the discourse. In contrast, when predicational properties are copied, a restitutive predication is introduced into the discourse. This is a partial classification since it does not cover all possible readings of *nach*-particle verbs, such as the temporal reading or the intensifying reading mentioned above. However since the task focusses on this classification, in the following we only take these five readings into account.

We discuss the five readings in some more detail below, based on the example sentences in Examples (6.1) to (6.5), illustrating the different *nach*-particle verb readings in context. The respective *nach*-particle verbs are highlighted.

(6.1) *Der Hund* ***lief*** *dem Hasen* ***nach.***
the dog ran the hare after
'The dog ran after the hare.'
(Haselbach 2011: (1))

(6.2) *Die Schüler* ***sprachen*** *der Lehrerin* ***nach.***
the students spoke the teacher after
'The students repeated the speaking of the teacher.'
(adapted from Haselbach 2011: (19)b)

(6.3) *Die Kinder* ***tanzten*** *den Nussknacker* ***nach.***
the children danced the Nutcracker after
'The children danced (an instance of) The Nutcracker.'
(adapted from Haselbach 2011: (40)c)

(6.4) *Der Koch* ***würzte*** *die Soße* ***nach.***
the chef seasoned the sauce after
'The chef seasoned the sauce some more/again.'
(adapted from Haselbach 2011: (28)b)

(6.5) *Die Banane* ***reifte*** ***nach.***
the banana ripened after
'The banana continued ripening.'
(Haselbach 2011: (32))

In Example (6.1), *nach* triggers a directional reading (copy direction, DIR): The dog follows the hare, running into the same direction. This reading also occurs in more figurative settings, such as in Example (6.6). The signposts in Example (6.6)a point into a specific direction, but are of course not in motion themselves (cf. Haselbach 2011), and in Example (6.6)b, the sound is conceptualized as if it followed a group of people heading for the locker room. The latter group of sound existence verbs (cf. Levin 1993: pp. 252f.), which can have this additional directional reading, had originally not been part of Haselbach's predictions and has been identified in another corpus study described by Haselbach et al. (2012b); this is a finding which supports the idea of combining theoretical and data-driven approaches to provide new insights. Example (6.6)b originates from the deWaC web corpus (Baroni et al. 2009).

(6.6)

a. *Der Wanderer* ***marschierte*** *den Wegweisern* ***nach.***
the hiker marched the signposts after
'The hiker followed the signposts.'
(adapted from Haselbach 2011: (16))

b. *Ihr Begeisterungsjubel* ***klingt*** *uns bis in die Kabine* ***nach.***
their cheering sounds us till in the locker room after
'Their cheering follows us into the locker room.'
(Haselbach et al. 2012b: (16))

Example (6.2) illustrates a different reading of *nach*, with a different kind of copying operation. Here, not the direction but the manner of the event is

copied (copy manner, MAN): The students speak like the teacher does; in this case, they use the same words and maybe even try to imitate the rhythm and intonation of the teacher.

The third *nach*-reading does not relate event properties, but result state properties of bringing a copy of something into existence (copy creation, CRE). This can either be an abstract entity, such as the ballet piece *The Nutcracker* in Example (6.3), or a concrete object, such as a banknote in Example (6.7). Thereby it is not important how this copy came into existence: the forger did (highly likely) not repeat the process of creating a valid banknote and nothing is said about whose (previous) interpretation of *The Nutcracker* is performed by the children. Neither does the result have to be an exact copy of a banknote (hopefully) or of a specific staging of the ballet (probably).

(6.7) *Der Fälscher* ***machte*** *den Geldschein* ***nach***.
the forger made the banknote after
'The forger made a copy of the banknote.'
(Haselbach 2011: (22)b)

However the two readings, copy manner and copy creation, are sometimes hard to separate. Copying the manner of someone might very well lead to a (near) copy of the result state (cf. Haselbach et al. 2012a).[4]

Example (6.4) reflects the two aspects of the predicational reading (once-more/ restitution, OMR): Either the sauce had been spicy once, but has become flavourless, e.g., due to added ingredients or too long boiling and needs to be seasoned again (restitution), or the sauce is already spicy but the chef wants to add even more taste to it (once-more). This is also the case with *nachschleifen* ('[to] regrind') in Example (6.8)

(6.8) *Der Schmied* ***schärfte*** *das Messer* ***nach***.
the blacksmith sharpened the knife after
'The blacksmith resharpened the knife.'
(Haselbach 2011: (28)a)

[4]Cf. Example (6.19) in the discussion on accusative arguments later.

The fifth reading is assumed to target progressive state properties (continuation, CONT): In Example (6.5) the banana continues ripening, e.g., after being picked. Here it is important that the state lasted after a specific point in time. The state before the point in time has to show similar properties but does not have to be identical, cf. Example (6.9). The progressive state is described by an anti-causative verb, i.e. the verb-internal argument surfaces as subject (cf. Haselbach et al. 2012a).

(6.9) *Der Motor brachte die Maschine zum Klirren. Nach dem*
the motor made the machine to clatter after the
Abschalten ***brummte*** *sie noch eine Weile lang* ***nach****.*
turning-off droned it still a while long after
'The motor made the machine clatter. After turning it off, it continued droning for a while.'

In the following, the five categories are referred to by their abbreviations DIR, MAN, CRE, OMR, and CONT, as applied in Haselbach et al. (2012a,b).

***nach*-particle verbs and their argument structure.** As stated above, the particle *nach* introduces a temporal relation between an asserted eventuality and a presupposed eventuality where properties of the asserted eventuality are copied and ascribed to the presupposed eventuality. As for the argument structure of *nach*-particle verbs, Haselbach (2011) argues that *nach* creates an additional argument slot for a dative, denoting the agent of the presupposed event, if it targets eventive properties of the verb phrase, while in the case where *nach* targets stative properties no additional dative argument is created.[5] In this context, the term 'additional' concerns the distinction between the argument structure of the base verb and the argument structure of the *nach*-particle verb. This splits the five *nach*-particle verb readings into two groups: DIR and MAN on one side, accessing event properties and thereby triggering a dative, and CRE, OMR, and CONT on the other side, without dative. Examples (6.10) to (6.14) repeat the

[5]For readability reasons in the remainder of this chapter, the arguments are also referred to with the abbreviatory terms 'dative' and 'accusative'.

sentences from Examples (6.1) to (6.5) to test if an included noun phrase can be interpreted as a dative argument with the base verb or the *nach*-particle verb.[6]

(6.10)

a. *Der* *Hund* *lief.*
the.NOM dog ran

b. **Der* *Hund* *lief* ***dem*** ***Hasen.***
the.NOM dog ran the.DAT hare

c. *?Der* *Hund* *lief* *nach.*
the.NOM dog ran after
'The dog ran after [someone/something].'

d. *Der* *Hund* *lief* ***dem*** ***Hasen*** *nach.*
the.NOM dog ran the.DAT hare after
'The dog ran after the hare.'

(adapted from Haselbach 2011: (1))

(6.11)

a. *Die* *Schüler* *sprachen.*
the.NOM students spoke

b. **Die* *Schüler* *sprachen* ***der*** ***Lehrerin.***
the.NOM students spoke the.DAT teacher

c. *?Die* *Schüler* *sprachen* *nach.*
the.NOM students spoke after
'The students repeated the speaking.'

d. *Die* *Schüler* *sprachen* ***der*** ***Lehrerin*** *nach.*
the.NOM students spoke the.DAT teacher after
'The students repeated the speaking of the teacher.'

(adapted from Haselbach 2011: (19)b)

(6.12)

a. *Die* *Kinder* *tanzten* *den* *Nussknacker.*
the.NOM children danced the.ACC Nutcracker

[6] In these examples, the introduced dative is printed in boldface. Case information is added to the respective determiner, the used case abbreviations are: NOM – nominative, ACC – accusative, DAT – dative. Where applicable, grammaticality judgements are given at the beginning of a sentence: * – ungrammatical sentence, ? – unclear status of grammaticality.

b. *Die* *Kinder* *tanzten* ***der*** ***Oma*** *den* *Nussknacker.*
the.NOM children danced the.DAT granny the.ACC Nutcracker
'The children danced The Nutcracker for the granny.'

c. *Die* *Kinder* *tanzten* *den* *Nussknacker* *nach.*
the.NOM children danced the.ACC Nutcracker after
'The children danced (an instance of) The Nutcracker.'

d. *Die* *Kinder* *tanzten* ***der*** ***Oma*** *den* *Nussknacker* *nach.*
the.NOM children danced the.DAT granny the.ACC Nutcracker after
(i) 'The children danced The Nutcracker for the granny.'/
(ii) 'The children copied **the granny's dancing of** The Nutcracker.'

(adapted from Haselbach 2011: (40))

(6.13)

a. *Der* *Koch* *würzte* *die* *Soße.*
the.NOM chef seasoned the.ACC sauce

b. *Der* *Koch* *würzte* ***dem*** ***Lehrling*** *die* *Soße.*
the.NOM chef seasoned the.DAT apprentice the.ACC sauce
'The chef seasoned the sauce for the apprentice.'

c. *Der* *Koch* *würzte* *die* *Soße* *nach.*
the.NOM chef seasoned the.ACC sauce after
'The chef seasoned the sauce some more/again.'

d. *Der* *Koch* *würzte* ***dem*** ***Lehrling*** *die* *Soße* *nach.*
the.NOM chef seasoned the.DAT apprentice the.ACC sauce after
'The chef increased the seasoning of the sauce for the apprentice.'

(adapted from Haselbach 2011: (28)b)

(6.14)

a. *Die* *Banane* *reifte.*
the.NOM banana ripened

b. **Die* *Banane* *reifte* ***dem*** ***Apfel.***
the.NOM banana ripened the.DAT apple

c. *Die Banane reifte nach.*
the.NOM banana ripened after
'The banana continued ripening.'

d. **Die Banane reifte **dem** **Apfel** nach.*
the.NOM banana ripened the.DAT apple after

(adapted from Haselbach 2011: (2) and (32))

Examples (6.10) and (6.11) show that a dative argument is possible with the DIR (Example (6.10)d) and the MAN reading (Example (6.11)d) and that this dative argument is not introduced by the base verb (Example (6.10)a+b, Example (6.11)a+b) but triggered by the particle *nach*. The sentences in Example (6.10)c and Example (6.11)c, with the *nach*-particle verb but without the dative argument, are considered marginal (cf. Haselbach 2011), since they depend on a specific context to allow the omission of the dative argument, e.g., in an elliptic construction with an otherwise identified agent, as in Example (6.15)a[7] (cf. Haselbach 2011), and Example (6.15)b from the deWaC web corpus (Baroni et al. 2009).

(6.15)

a. *?Der Rattenfänger ging voran und alle rannten nach.*
the Pied Piper went ahead and everybody ran after.
'The Pied Piper went ahead and everybody followed (him).'
(Haselbach 2011: (18))

b. *Dann spricht der Mönch die Formeln vor, [...]. Die Laien sprechen nach: [...]*
then speaks the monk the set phrases prior the laymen speak after
'Then the monk speaks the set phrases aloud, [...]. The laymen repeat (them): [...]'
(deWaC)

None of the sentences in Examples (6.12) and (6.13) are ungrammatical. Nevertheless, the dative cannot be interpreted in any of the respective readings as being an argument of the *nach*-particle verb: In Example (6.12)b and Example (6.13)b+d the only grammatical reading for the sentences is one where the

[7] The ? in the beginning of the example denotes its unclear status of grammaticality.

dative is interpreted as a benefactive (McIntyre 2006).[8] In the case of Example (6.12)d the dative can either be read as benefactive, or as an argument of the *nach*-particle verb, where the latter then only allows a MAN reading of the sentence and no longer the CRE reading. Furthermore the presence of a benefactive dative is possible for the *nach*-particle verb (Example (6.12)d, Example (6.13)d) and the base verb (Example (6.12)b, Example (6.13)b) alike, but the benefactive is not interpreted as being part of the argument structure of the *nach*-particle verb itself, but rather as an independently triggered argument, e.g. by means of an applicative structure.

The example sentence for the CONT reading in Example (6.14) neither allows for a dative argument with the base verb (Example (6.14)b) nor with the *nach*-particle verb (Example (6.14)d).

The observation that the presence or absence of a dative separates two groups of readings, i.e. those relating event properties (DIR, MAN) and those relating state properties (CRE, OMR, CONT) is crucial for the corpus study: a context involving a dative argument can restrict the possible readings of a *nach*-particle verb. We make use of this as an indicator for the reading of a *nach*-particle verb in the corpus study.

Contrary to dative arguments, the presence or absence of an accusative argument with a *nach*-particle verb does unfortunately not provide an equally clear distinction but only some contextual indication (cf. Haselbach et al. 2012a).[9]

For the DIR reading, the occurrence of an accusative argument is independently triggered by the base verb, cf. Examples (6.16) and (6.17). Neither *laufen* ('[to] run'), cf. Example (6.16)a, nor *nachlaufen* ('[to] run after'), cf. Example (6.16)b, can appear with an accusative. On the other hand, *rollen* ('[to] roll') which has an accusative argument, cf. Example (6.17)a, also keeps the argument

[8] In the translations of the examples the underlining signals the benefactive readings.

[9] In the respective examples, i.e. Examples (6.16) to (6.21) the accusative is printed in boldface. Case information is again added to the respective determiner, making use of the same abbreviations as before. And, where applicable, grammaticality judgements are again given at the beginning of a sentence.

with the *nach*-particle verb *nachrollen* ('[to] roll after'), cf. Example (6.17)b.[10] Since the particle does not seem to be influential here, we do not regard the presence of an accusative argument as a specific indicator in favour of the DIR reading.

(6.16)

a. **Der* *Hund* *lief* ***den*** ***Ball.***
the.NOM dog ran the.ACC ball

b. **Der* *Hund* *lief* *dem* *Mädchen* ***den*** ***Ball*** *nach.*
the.NOM dog ran the.DAT girl the.ACC ball after

(adapted from Haselbach et al. 2012a: (4))

(6.17)

a. *Der* *Hund* *rollte* ***den*** ***Ball.***
the.NOM dog rolled the.ACC ball
'The dog rolled the ball.'

b. *Der* *Hund* *rollte* *dem* *Mädchen* ***den*** ***Ball*** *nach.*
the.NOM dog rolled the.DAT girl the.ACC ball after
'The dog rolled the ball after the girl.'

(adapted from Haselbach et al. 2012a: (4))

For the MAN reading there also does not seem to be a correlation with the accusative, as it can be present or absent even with the same predicate, (cf. Haselbach et al. 2012a) and Example (6.18)[11]. Still the presence of an accusative argument with an ambiguous *nach*-particle verb might help to disambiguate with respect to other readings, as in Example (6.19). Without the accusative argument *den Nussknacker*, the sentence is ambiguous between DIR and MAN (Example (6.19)a); both of these readings require a dative argument. If the dative is replaced with an accusative, the sentence switches to the CRE reading (Example (6.19)b), however an additional introduction of the accusative argument results in an ambiguity between the CRE reading with a benefactive reading of the dative (Example (6.19)c,i), and the MAN reading (Example (6.19)c,ii).

[10] However as stated before, the particle *nach* triggers a dative argument, instantiated in both examples as 'the girl'.

[11] The MAN reading triggers a dative argument, here 'the teacher'.

(6.18)

a. *Die* *Schüler* *sprachen* *der* *Lehrerin* *nach.*
the.NOM students spoke the.DAT teacher after
'The students repeated the speaking of the teacher.'

b. *Die* *Schüler* *sprachen* *der* *Lehrerin* ***ein paar*** ***Wörter*** *nach.*
the.NOM students spoke the.DAT teacher some.ACC words after
'The students repeated some words from the teacher.'

(adapted from Haselbach 2011: (19)b)

(6.19)

a. *Die* *Kinder* *tanzten* *der* *Oma* *nach.*
the.NOM children danced the.DAT granny after
(i) 'The children followed the granny dancing.'/
(ii) 'The children copied the granny's dancing.'

b. *Die* *Kinder* *tanzten* ***den*** ***Nussknacker*** *nach.*
the.NOM children danced the.ACC Nutcracker after
'The children danced (an instance of) The Nutcracker.'

c. *Die* *Kinder* *tanzten* *der* *Oma* ***den*** ***Nussknacker*** *nach.*
the.NOM children danced the.DAT granny the.ACC Nutcracker after
(i) 'The children danced The Nutcracker for the granny.'/
(ii) 'The children copied the granny's dancing of The Nutcracker.'

(adapted from Haselbach 2011: (40))

With CRE and OMR readings the accusative seems to be obligatory, cf. Examples Example (6.19)b+c and Example (6.20)c, however in cases where the *nach*-particle verb has a strong preference for the OMR reading, the accusative argument can be externally understood, cf. Example (6.20)a+b.

(6.20)

a. *Der* *Koch* *würzte* *nach.*
the.NOM chef seasoned after
'The chef seasoned (the sauce) some more/again.'

b. *Bei uns würzt der Koch nach, nicht der Gast.*
at us seasons the.NOM chef after, not the.NOM guest
'At ours it is the chef, who seasons (the food) some more/again, it is not the guest.'

c. *Der Koch würzte **die Soße** nach.*
the.NOM chef seasoned the.ACC sauce after
'The chef seasoned the sauce some more/again.'
(adapted from Haselbach 2011: (28)b)

With CONT there cannot be an accusative argument, as in Example (6.21). This is due to the aforementioned anti-causative verbs which seem to prefer the CONT reading, where the verb-internal argument surfaces as a subject, and which neither appears with a dative nor with an accusative argument, (cf. Haselbach et al. 2012a).

(6.21) **Die Banane reifte **den Apfel** nach.*
the banana ripened the.ACC apple after
(adapted from Haselbach et al. 2012a: (8))

indicator	DIR	MAN	CRE	OMR	CONT
dative	+	+	-	-	-
accusative	(-)	?	+	+	-

Table 6.1: Dative and accusative arguments as indicators for *nach*-particle verb readings.

See Table 6.1[12] for the combined indicator status of dative and accusative arguments. Although Haselbach (2011) stays agnostic about the role of the accusative argument as an indicator for *nach*-particle verb readings, we take both types of arguments as potential indicators into account. With regard to the DIR reading, we assume more base verbs to occur without an accusative argument, thus the DIR reading will be slightly dispreferred for *nach*-particle verbs occurring with an accusative. We do not assume any indicator status for the accusative argument with respect to the MAN reading, but consider its

[12]The table is taken from Haselbach et al. (2012a).

presence as an indicator for CRE or OMR, and its absence as an indicator for CONT, especially when the dative is absent as well.

Table 6.1 shows that this distribution of indicators does not allow for a clear assignment of specific *nach*-particle verbs to their respective reading solely on the basis of the presence or absence of dative and accusative arguments. Moreover, *nach*-particle verbs are often ambiguous with respect to their possible readings regarding the classification from Haselbach (2011), and they can therefore in principle instantiate several of the above-mentioned argument structure configurations.

Testing the hypotheses. In the above summary of Haselbach's theoretical approach, the correlation between semantic subclasses of *nach*-particle verbs and argument structure has been discussed. The task of validating this theory by means of reproducing the classes on "real world data" from corpora consists in turning around the argumentation and making use of argument structure as a contextual indicator. Figure 6.2 shows the workflow of this overall task.

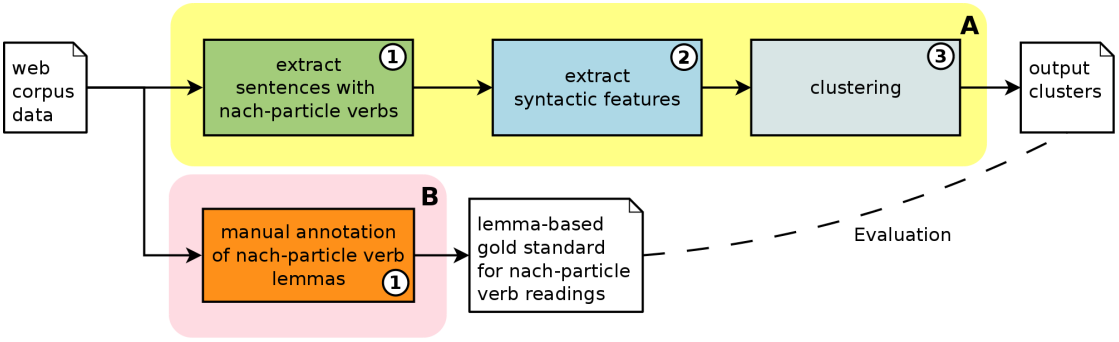


Figure 6.2: Workflow of overall task.

In this workflow there are two branches. Branch A predicts a set of clusters which classify *nach*-particle verb lemmas based on the frame of syntactic arguments they occur with in a web corpus. Branch B creates a gold standard to evaluate the result of Branch A. To prepare this evaluation, a list of *nach*-particle verb lemmas is constructed from candidates extracted from the web corpus. These lemmas are manually annotated by three independent annotators

(Box B① in Figure 6.2) for the readings from Haselbach (2011), producing a lemma-based gold standard.

For Branch A, sentences containing a *nach*-particle verb are extracted from the web corpus (Box A① in Figure 6.2). The sentence-based corpus context is then only utilized to extract the syntactic features with which a *nach*-particle verb lemma occurs (Box A②) along with their relative frequency. The features taken into account as the input for a standard clustering approach (Box A③: Ward's algorithm, Ward 1963) are the occurrence of a dative or of an accusative argument with the *nach*-particle verb and the word form of the respective argument.[13] The output clusters are evaluated against the lemma-based gold standard. Note that this is a type-based clustering approach: the candidates for the clustering are the *nach*-particle verb lemmas and the features combine the information from the corpus per lemma. The features containing the word form of the argument take the actual form of the argument into account. For the gold standard, the lemmas are annotated according to the predictions of the theory, without taking context from the corpus into account. For details on the evaluation see Section 6.1.2.

The case studies for the parser combination are instantiated at the step of extracting the syntactic features (Box A② in Figure 6.2) and make use of the different annotations by the parsers regarding dative and accusative arguments.

6.1.1 Resources used in the task and the case studies

In the following we briefly present the resources which are involved in this task and its combination case studies.

SdeWaC. The study of Task I is based on a subcorpus of the web corpus deWaC (Baroni and Kilgarriff 2006; Baroni et al. 2009), called SdeWaC (Faaß and Eckart 2013).[14] Since the type of data included in this study is of prime

[13] Haselbach et al. (2012a) takes some more features into account, but for the description of Task I we stick to the features which play a role in all of the parser combination schemes here.

[14] By SdeWaC we refer to SdeWaC-v3, which is the version described by Faaß and Eckart (2013) and made available via the WaCky initiative, like the original deWaC corpus. See the WaCky web page for further information on the initiative: `http://wacky.sslmit.unibo.it/doku.php`

importance for the performance of the parsers, we will first get some insight on deWaC and SdeWaC before discussing the resources derived from it for the task.

deWaC is a corpus of web data, i.e. downloaded documents from the web, and was developed between 2005 and 2007, cf. Baroni et al. (2009). The web crawling started from URLs which were retrieved by queries of content word pairs, designed to yield a variety of what Baroni et al. (2009) call public sphere documents and personal interest pages. Thereby the former includes journalistic and academic texts, while the latter refers to user-generated content, e.g. from blogs. The development of deWaC also included some cleaning steps, e.g. selection of documents based on mime type and size restrictions, removal of all documents with perfect duplicates, removal of boilerplate material[15] and source code like JavaScript from documents, filtering of documents with respect to language identification and the probability to contain continuous text, and a first attempt at removing near-duplicate documents. Due to the amount of data available on the web, the focus of the cleaning was on precision, thus, by means of the cleaning steps, only about 4% of 398 GB raw crawl data remained in the corpus, cf. Baroni et al. (2009). The remaining documents were annotated with part-of-speech tags and lemmas, utilizing the TreeTagger (Schmid 1994). However in the creation of the subcorpus SdeWaC this annotation was discarded and re-annotated at a later stage.

Baroni et al. (2009) report that in deWaC about 1,278 million tokens are identified, while Baroni and Kilgarriff (2006) report on a final set of about 1,710 million tokens. This might be due to different corpus versions or token definitions. However, counting all lines which are not sentence or document tags in the version available at the institute at the time of this thesis, 1,640 million tokens are identified, thus we go rather with the 1,710 million tokens for the comparison with the subcorpus.

SdeWaC is restricted to sentences which are useful as input for automatic natural language processing steps up to the level of syntactic parsing. Moreover

[15]The term boilerplate refers to recurring natural language material, such as headers, disclaimers, navigation bars, etc., cf. Baroni et al. (2009).

sentence duplicates were removed, if they were crawled from sources with the same domain name (cf. Faaß and Eckart 2013).

To prepare SdeWaC, several steps have been applied: (i) Only sentences from the top-level domain `.de` have been included, in order to e.g. differentiate between regional varieties. (ii) The sentences have been sorted uniquely, i.e., the order of the sentences in the corpus was discarded and the duplicates from sources with the same domain name were deleted. Furthermore, character-based heuristics based on the approach by Quasthoff et al. (2006) have been applied in several steps to distinguish between processable sentences and non-sentential character sequences. For example, the relation of the number of non-alphabetic characters to the total number of characters in a sentence can be such an indicator. (iii) Based on the underspecified output of FSPar, those sentences have been removed for which the parser was not able to assign a dependency structure.[16] Cf. Section 2.1.4 for a discussion on the output of FSPar.

The fact that we decided on the parsability of a sentence on the basis of FSPar is important here: since FSPar is also one of the parsers used in the task, we need to keep in mind that, in the absolute numbers of our baselines for the combination case studies, there is a bias towards decisions of FSPar, i.e. the quality of the FSPar output in comparison to the other parsers involved in the task might be artificially high. However we are not interested in the absolute performance of the individual tools, but we want to compare the different combination schemes with respect to their relative differences, and since FSPar is included in all case studies for the different combination schemes applied for this task, we can abstract away from this bias in our evaluation.

Furthermore, FSPar was chosen as a discriminator for sentences in SdeWaC in the first place due to the fact that its output can be underspecified and is therefore less restrictive: Forced disambiguation in a specific processing step, i.e. the omission of possible readings in favour of a single result,[17] may lead to a

[16] A certain range of tolerance was included, such that not all parts of a sentence have to fit into the structure in order for it to be considered as parsable. Cf. Faaß and Eckart (2013) for a description of the threshold procedure.

[17] Forced disambiguation usually relies on some sort of learned probability, such that the most likely result is "guessed".

corrupted syntactic analysis for an otherwise syntactically well-formed sentence; however, by making use of FSPar's underspecified output no sentences were lost due to forced guessing (Faaß and Eckart 2013).

Apart from this, the revision of the corpus from deWaC to SdeWaC was rather precision-oriented, i.e. potentially problematic sentences were more often rejected than not.[18] Thus SdeWaC is considerably smaller than deWaC (884 million tokens vs. 1,710 million tokens) but still reflects web-specific content. So far, parsers are usually trained on or built for "well-behaved" written text, such as newspaper corpora. Hence, web corpora constitute non-canonical input for these parsers, which has an influence on the baselines presented in Section 6.1.2. With regard to the task, however, the objective is to use texts covering diverse topics in order to find examples of all *nach*-particle verb subclasses, and as *nach*-particle verbs are a relatively rare phenomenon, using a very large corpus allows us to increase the number of candidate items. For these two reasons, SdeWaC as been chosen.

It is to be noted that one objective of the preparation of SdeWaC was also to remove duplicate sentences.[19] Due to unique sorting and removal of sentences, the size of the linguistic context available is restricted to the level of sentences (not sentence sequences or documents); this is however not a relevant restriction for Task I, since we focus on the argument structure of the *nach*-particle verbs which is typically considered to be determined at sentence level.

Parsers. This paragraph lists the parsers employed for Task I. For a general introduction to the parsers, their processing pipelines, and their output, see Section 2.1.4.

The mate parser: The mate parser, a data-driven dependency parser, was trained on a training set produced by means of a dependency conversion of the TIGER treebank. See Brants et al. (2004) for the original TIGER

[18] For example, in the final processing step, sentences where the conversion of the character encodings produced UTF-8 control characters were deleted completely, instead of only deleting the respective characters and keeping a potentially corrupted sentence.

[19] Sentence duplicates remained however in the corpus if they were crawled from sources differing in their domain name (cf. Faaß and Eckart 2013).

corpus and Seeker and Kuhn (2012) for the conversion. The standard preprocessing pipeline was applied with models which are also derived from the abovementioned conversion. Since the mate parser pipeline requires tokenized input, the tokenizing step was done in advance.

FSPar: The rule-based dependency parser FSPar was applied in the version checked out from its CVS[20] repository in February 2009. FSPar's own pre-processing pipeline was applied for tokenizing, lemmatization, morphological tagging and part-of-speech tagging. For the output the tabular underspecified output format was utilized, cf. Section 2.1.4.

BitPar: The data-driven constituency parser BitPar was applied in the version installed at the *Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart* in January and February 2013. After testing several configurations, the Latin 1 version of the parser was applied, i.e. the version which processes data encoded according to ISO/IEC 8859-1:1998, and information from SMOR (Schmid et al. 2004) was added in the creation of the parser lexicon. The parsing step also includes part-of-speech tagging, however there was no previous tokenization step involved, since the same tokenization which was used for the mate parser was utilized. Due to the size of the dataset, the input has been split into parts for parsing. However the parser lexicon has been built in advance over all data, such that the same lexicon was employed for every part.

Sub-corpus of sentences with *nach*-particle verbs. For the study, sentences containing at least one *nach*-particle verb were extracted from SdeWaC.[21] In the workflow of the overall task shown in Figure 6.2 this reflects the step of Box A①.

Since German particle verbs can appear either as one continuous string or with a separated particle in the sentence, automatic identification of *nach*-particle verbs cannot be done by a simple pattern matching approach. Depending on

[20] Concurrent Versions System.

[21] The extraction of the sentences happened before the preparation of SdeWaC was fully completed. Thus the *nach* sub-corpus might contain sentences which are no longer available in SdeWaC. They are however all part of deWaC.

the processing tools, a lexicon-based lemmatization, or a part-of-speech tagging step might already provide respective information, however for an informed linking of the separated verb particle to its base verb, the parsing step might be required.

To find the sentences containing at least one *nach*-particle verb we proceeded as follows: In a first step all sentences containing one or more character sequences which start with the string *nach* were extracted. This set comprises 3,786,615 candidate sentences. From this data set, input files for the mate parser, for FSPar and for BitPar were derived in their respective input formats. Sentences containing more than 300 tokens were excluded from these files for processing reasons, leaving 3,786,139 sentences remaining. The input data sets were parsed by the mate parser, by FSPar and by BitPar in the configurations described above.

To be able to compare the different parses of the same sentence afterwards, an alignment had to be created. This was done by numbering the input sentences.[22] For the mate parser, the sentence number was included in the input file as a prefix to the token number. For FSPar the article markup was utilized to include the common sentence numbering, by letting each sentence constitute exactly one article.[23] For BitPar the sentence numbering was included after the parsing, since no such information could be kept during the parsing itself. However, the input for BitPar was derived from the mate parser input to ensure that the same tokenization and sentence borders were applied, and since BitPar keeps the sentence borders during parsing, the sentences could be numbered after the parsing step without any loss of information.

Although it would in principle be possible to utilize the same tokenization for all three parsers this would result in a loss of performance for at least one of the parsers in each configuration, and thus decrease the possible gain expected

[22] Numbering starting from one.

[23] FSPar's processing pipeline does not necessarily keep the proposed sentence borders from the input, cf. Figure 2.13 (Page 82), thus the static article markup was utilized. On the one hand this might seem as a misuse of the article markup, which is designed to introduce document borders; on the other hand, no context exceeding the sentence level was available anyway here, due to the sorting and removal of duplicates in SdeWaC, and to the fact that only single *nach* sentences had been extracted.

of a combination step. FSPar heavily relies on the lexical information from its own tokenizing, such that it would suffer from omitting this step. However, the tokenization of FSPar tends to concatenate multiple words into one token, which would be disadvantageous for the mate parser and for BitPar since data-driven parsers usually work best on the tokenization style of the data they are based on, e.g. the whitespace-derived tokenization applied here.

In the resulting parses, sentences containing *nach*-particle verbs had to be identified. As mentioned above the identification of *nach*-particle verbs can happen by means of different annotation layers, e.g. with the help of lemma annotations, part-of-speech tags or dependency relations.[24] In the following we describe for each parser how the sentences containing *nach*-particle verbs were identified. For this we make reference to examples of the output of the parsers for two sentences with the verb *nachrennen* ('[to] run after') from SdeWaC, once with a separated verb particle, cf. Example (6.22), and once as a continuous verb form, cf. Example (6.23).[25]

(6.22) *Er* ***rennt*** *ihm* ***nach.***
he runs him after
'He ran after him.'

(6.23) *Dort kommen die Türen immer exakt an einer auf dem*
there come the doors always exactly at one on the
Bahnsteig markierten Stelle zum Halten, und kein Fahrgast
platform marked position to the stopping and no passenger
muß mit Gepäck der Zugtür ***nachrennen.***
has to with luggage the train door run after
'There, the doors always stop exactly at a marked position of the platform, and no passenger has to run after the door with luggage.'

FSPar: Since FSPar makes use of a large lexicon, information about verb particles does not only appear in the part-of-speech tag of a separated verb particle (PTKVZ, cf. Figure 6.3a, Token 3) but also in the verb lemma: the # character separates the particle and the base verb in the lemma form

[24] In the ideal case the annotations from these layers do not contradict each other.
[25] In these examples the *nach*-particle verb is highlighted.

annotated, e.g. *nach#rennen*. Moreover, this information is independent of the particle being separated in the sentence or not (cf. Figure 6.3a, Token 1 and Figure 6.3b, Token 24). To identify a sentence containing a *nach*-particle verb with regard to FSPar, this lemma annotation was utilized.

The mate parser: For tokens which start with (or consist of) the string *nach* or *Nach* the part-of-speech tag and the dependency label were taken into account. At least one of the following criteria had to be fulfilled to identify a sentence as having a *nach*-particle verb: (i) the dependency label had to be SVP, denoting a separated verb particle, (ii) the part-of-speech tag of *nach* had to be PTKVZ, ADJA, or ADJD, or, alternatively, (iii) the part-of-speech tag had to start with a 'V'. The first and second criteria aim at separated particles which have either been correctly recognized (cf. Figure 6.4a, Token 4), or mis-tagged, the third criterion aims at verbs with attached particles, which cannot be specifically identified, but are tagged as verbs, starting with *nach* (cf. Figure 6.4b, Token 25). All steps in the processing pipeline of the mate parser are based on trained models, i.e. also the lemmatization step is not based on a lexicon, which can result in errors like the capitalization error of Token 25 in Figure 6.4b. As mentioned in Section 2.1.4, for readability reasons only the columns containing information are shown in the examples; furthermore the sentence number encoded in front of each token position is omitted.

BitPar: From the output of BitPar, sentences with *nach*-particle verbs were either identified by means of the part-of-speech tag and the syntactic function of the separated particle, PTKVZ-SVP, appearing with *nach*, cf. Figure 6.5a, or by identifying a verb via its part-of-speech tag, and its form starting with the string *nach*, cf. Figure 6.5b, showing the subtree with the *nach*-particle verb.[26]

[26]For those familiar with regular expressions and bracketing formats utilized for constituency trees, such as in the Penn Treebank (Marcus et al. 1993), the patterns searched for in the actual output are $(PTKVZ\text{-}SVP\ nach)$ and $(V[\hat{}()]*\ nach[\hat{}()]+)$, the latter denoting that the part of speech tag starts with V and the token starts with *nach*, followed by one or more characters.

<s>

0	Er	PPER	er	Nom:M:Sg	1	NP:1
1	rennt	VVFIN	nach#rennen	3:Sg:Pres:Ind|2:Pl:Pres:Ind	-1	TOP
2	ihm	PPER	er|es	Dat:M:Sg|Dat:N:Sg	1	NP:4
3	nach	PTKVZ	nach	|	1	RK
4	.	$.	.	|	-1	TOP

</s>

(a) Sentence with separated verb particle (Token 3)

<s>

0	Dort	ADV	dort	|	1	ADJ
1	kommen	VVFIN	kommen		-1	TOP
2	die	ART	d	|	3	SPEC
3	Türen	NN	Türe	Nom:F:Pl	1	NP:1
4	immer	ADV	immer	|	1||5	ADJ
5	exakt	ADJD	exakt	|	1	ADJ
6	an	APPR	an	Dat	1	ADJ
7	einer	ART	ein	|	12	SPEC
8	auf	APPR	auf	Dat	11	ADJ
9	dem	ART	d	|	10	SPEC
10	Bahnsteig	NN	Bahn#@steig	Dat:M:Sg	8	PCMP
11	markierten	ADJA	markierenP	|	12	ADJ
12	Stelle	NN	Stelle	Dat:F:Sg	6	PCMP
13	zum	APPRART	zu	Dat:M:Sg|Dat:N:Sg	1||12	ADJ|PP/zu:4
14	Halten	NN	Halten	Dat:M:Sg|Dat:N:Sg	13	PCMP
15	,	$,	,	|	1|24/19	PUNCT
16	und	KON	und	|	1|24/19	KON
17	kein	PIAT	kein	|	18	SPEC
18	Fahrgast	NN	Fahr#@gast	Nom:M:Sg	24/19	NP:1
19	muß	VMFIN	müssenI		-1	TOP
20	mit	APPR	mit	Dat	24/19	ADJ|PP/mit:4
21	Gepäck	NN	Gepäck	Dat:N:Sg	20	PCMP
22	der	ART	d	|	23	SPEC
23	Zugtür	NN	Zug#@tür	Gen:F:Sg	21	GR
24	nachrennen	VVFIN	nach#rennen	Inf	24/19	RK
25	.	$.	.	|	-1	TOP

</s>

(b) Sentence with continuous particle verb (Token 24)

Figure 6.3: Excerpts of FSPar output for Examples (6.22) and (6.23).

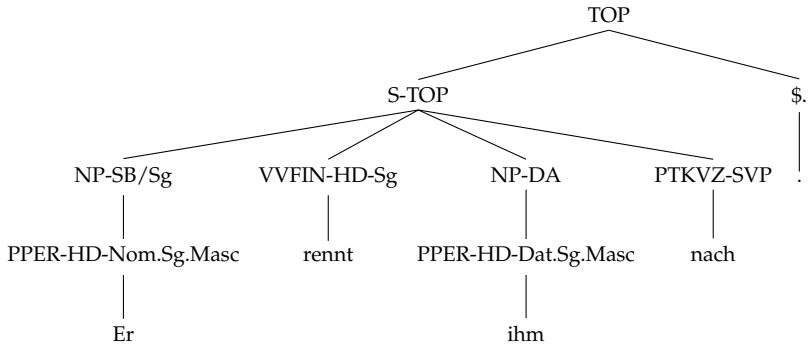
1	Er	er	PPER	nom|sg|masc|3	2	SB
2	rennt	rennen	VVFIN	sg|3|pres|ind	0	–
3	ihm	ihm	PPER	dat|sg|masc|3	2	DA
4	nach	nach	PTKVZ	_	2	SVP
5	.	–	$.	_	4	–

(a) Sentence with separated verb particle (Token 4)

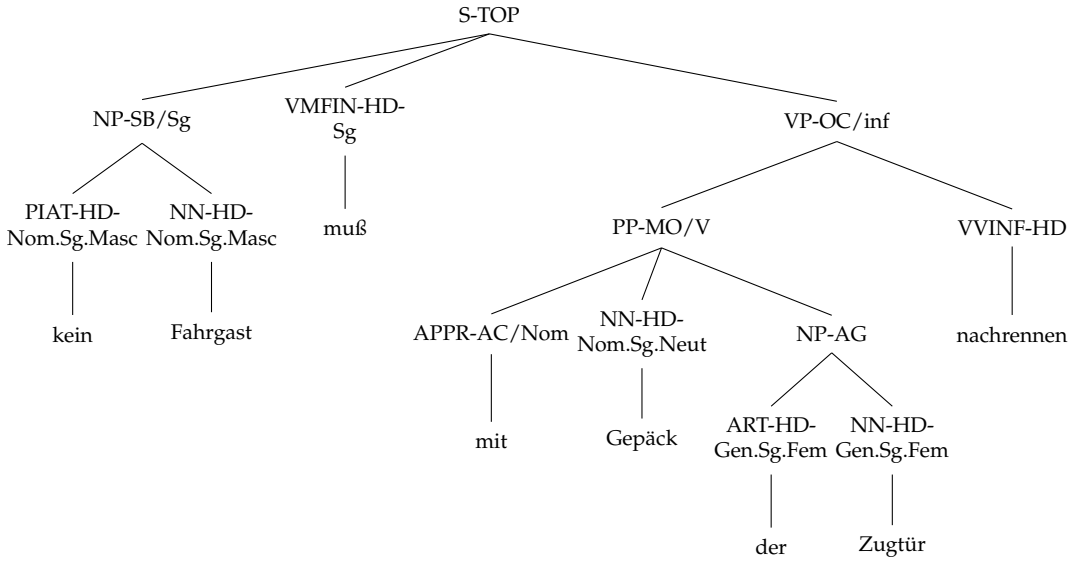
1	Dort	dort	ADV	_	2	MO
2	kommen	kommen	VVFIN	pl|3|pres|ind	0	–
3	die	der	ART	nom|pl|masc	4	NK
4	Türen	Tür	NN	nom|pl|masc	2	SB
5	immer	immer	ADV	_	6	MO
6	exakt	exakt	ADJD	pos	2	MO
7	an	an	APPR	_	2	MO
8	einer	ein	ART	dat|sg|fem	13	NK
9	auf	auf	APPR	_	12	MO
10	dem	der	ART	dat|sg|masc	11	NK
11	Bahnsteig	Bahnsteig	NN	dat|sg|masc	9	NK
12	markierten	markiert	VVFIN	pl|3|past|ind	13	NK
13	Stelle	Stelle	NN	acc|sg|fem	7	NK
14	zum	zu	APPRART	dat|sg|neut	2	MO
15	Halten	Halten	NN	dat|sg|neut	14	NK
16	,	–	$,	_	15	–
17	und	und	KON	_	2	CD
18	kein	kein	PIAT	nom|sg|masc	19	NK
19	Fahrgast	Fahrgast	NN	dat|sg|masc	20	SB
20	muß	müssen	VMFIN	sg|3|pres|ind	17	CJ
21	mit	mit	APPR	_	25	MO
22	Gepäck	Gepäck	NN	dat|sg|fem	21	NK
23	der	der	ART	gen|sg|fem	24	NK
24	Zugtür	Zugtür	NN	gen|sg|fem	22	AG
25	nachrennen	Nachrennen	VVINF	_	20	OC
26	.	–	$.	_	25	–

(b) Sentence with a continuous particle verb (Token 25)

Figure 6.4: Excerpts of output of the mate parser for Examples (6.22) and (6.23).



(a) Sentence with separated verb particle



(b) Part of a sentence with a continuous particle verb

Figure 6.5: Excerpts of BitPar output for Examples (6.22) and (6.23).

As an outlook to the handling of the arguments, note that while all three outputs for Example (6.22) correctly recognize the dative argument, all three outputs for Example (6.23) analyse *der Zugtür* as being in genitive case, and attach it to *Gepäck* or the prepositional phrase. Combining analyses in these cases would thus not improve the result over the single analyses. However in the case studies we will discuss how deviating analyses can be exploited to increase the number of correctly found arguments.

set	cardinality
$\mathcal{N}$	3786139
M	271326
F	250538
B	233148
$M \cap F$	246995
$M \cap B$	225606
$F \cap B$	215102
$M \cap F \cap B$	213507
$(M \cap F) \setminus B$	33488
$(M \cap B) \setminus F$	12099
$(F \cap B) \setminus M$	1595
$M \setminus (F \cup B)$	12232
$F \setminus (M \cup B)$	1948
$B \setminus (M \cup F)$	5947
$M \cup F \cup B$	280816
$\mathcal{N} \setminus (M \cup F \cup B)$	3505323

Table 6.2: Sentences with at least one *nach*-particle verb according to the parsers, where $\mathcal{N}$ is the set of input sentences for all parsers, M is the set of sentences identified by the mate parser, F is the set of sentences identified by FSPar, and B is the set of sentences identified by BitPar.

The above three passages describe the different patterns which were used to extract *nach*-particle verb sentences for each parser. According to these patterns, 280,816 of the 3,786,139 input sentences were identified to contain a *nach*-particle verb by at least one parser. Table 6.2 gives a more detailed overview of the subsets identified by the parsers and of their agreement.

The first column contains the identification of the subset we are looking at. The subset is described by means of sets of sentences and the set theoretic

binary operations union ($A \cup B$), intersection ($A \cap B$), and difference ($A \setminus B$). The utilized sets are: $\mathcal{N}$, which denotes the set of input sentences for all parsers, i.e. those sentences containing the string *nach* as described above; M, which denotes the set of sentences identified by the mate parser; F, which denotes the set of sentences identified by FSPar; and B which denotes the set of sentences identified by BitPar. The second column shows the cardinality of the subsets, i.e. the number of sentences which belong to the respective set.

The first set in the table is the set of input sentences for all parsers. The following three sets are based only on the results of one parser, independent of the decision of the other two. The next three sets are based on the agreement of two parsers, and the other described subsets display the configurations when all three parsers are taken into account. These last subsets are visualized as an Euler diagram scaled according to the set sizes in Figure 6.6. The set of sentences identified by the mate parser is represented by the circle with the solid contour, the set of sentences identified by FSPar is represented by the circle with the dashed contour, and the set of sentences identified by BitPar is represented by the circle with the dotted contour. The outer set of all input sentences is not explicitly represented. The overlap of the sets creates subsets which show the agreement between the parsers.

In this visualization it is easy to see that the largest subset is the one where all parsers agree on the existence of a *nach*-particle verb in a sentence: it contains 213,507 sentences. The largest subset of sentences only identified by one parser is the one of the mate parser, which is to be expected, since it is the subset of the parser identifying the most sentences overall. Moreover, for the mate parser we included the cases where a possibly separated *nach* particle has been (mis-)tagged as an attributive, adverbial, or predicative adjective which might also increase the number of identified sentences. The smallest subset of sentences only identified by one parser is the one of FSPar, since it only contains 1,948 sentences. Similarly, the set of sentences identified by FSPar and BitPar, but not by the mate parser, is very small with 1,595 sentences, and is barely visible on the Euler diagram.

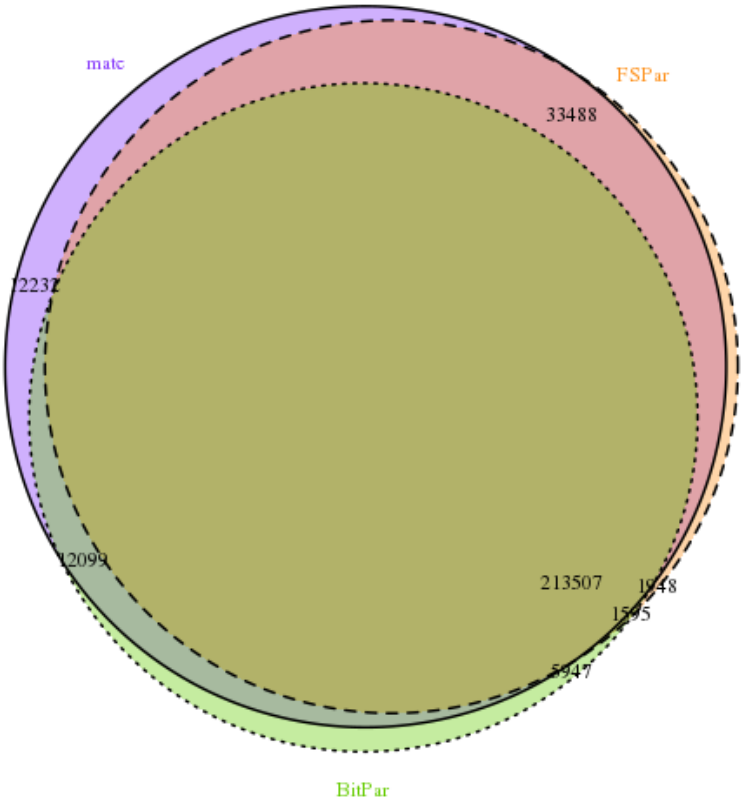


Figure 6.6: Euler diagram showing the agreement of the three parsers regarding the identification of sentences with *nach*-particle verbs.

For a small evaluation on false positives from the different parser combinations, and false negatives found by none of the parsers, see the paragraph on task-related parser evaluation in Section 6.1.2.

Lemma-based gold standard for *nach*-particle verb readings. To evaluate the results of Task I, i.e. the output of the clustering (cf. Figure 6.2), a gold standard for *nach*-particle verbs with respect to the five readings from Haselbach (2011) was created. The respective manual annotation reflects the step in Box B① of Figure 6.2.

475 *nach*-particle verb lemmas were extracted from the corpus without context and were manually classified by three independent annotators familiar with the five classes defined by Haselbach (2011). The annotation guidelines were provided by Haselbach (n.d.). A binary decision for each of the five readings was applied. The gold standard was then assembled by a simple majority decision for each lemma and each reading. The lemmas which, according to the majority decision, did not have any of the five tested readings were removed. These include *nach*-particle verbs which are part of classes not relevant to the classification of Haselbach (2011) such as the intensifying *nachdenken* ('[to] reflect') or the temporal *nachfeiern* ('[to] celebrate later'), but also cases where a single *nach* was erroneously interpreted as a separated verb particle. Table 6.3, adapted from Haselbach et al. (2012a), shows the distribution of the remaining 246 lemmas over the readings. Based on the binary decision for each reading, polysemy of a *nach*-particle verb is reflected by the creation of reading sets, combining all those readings which a lemma was assigned to.

The annotators were familiar with the five classes defined by Haselbach (2011) but the inter-annotator agreement is not very high regarding the kappa values, cf. Table 6.4.[27] Although the probability that all annotators agree on a particular class is rather high (P), the fact that the distribution of the *nach*-particle verb readings over the lemmas is rather unbalanced decreases the kappa value since it raises the probability of an accidental match with the binary decision (Pe).

Nevertheless, the annotation of the *nach*-particle verb readings to lemmas without context is not a trivial task. To prevent bias from the context and, in the best case, find all possible readings, the annotators had to come up with examples on their own, trying to fit a lemma with a specific reading. However they still could miss readings which are less common, or come up with rather coerced readings. An example for the first case was *nachlabern* (lit.: 'after'+'babble', paraphrase: derogatively reciting sth. in a babbling manner), which is labeled as exclusively belonging to the MAN class, while a subsequent investigation would also relate it to the CRE reading (Haselbach et al. 2012a).

[27] The table is taken from Haselbach et al. (2012a).

rank	reading set	# of lemma types
1	{OMR}	67
2	{DIR}	58
3	{MAN,CRE}	26
4	{MAN}	20
5	{CONT}	19
6	{MAN,OMR}	8
6	{DIR,MAN,CRE}	8
7	{DIR,MAN}	7
8	{OMR,CONT}	6
8	{MAN,CRE,OMR}	6
9	{DIR,CONT}	5
10	{CRE,OMR}	4
11	{DIR,CRE}	3
11	{DIR,OMR}	3
12	{DIR,OMR,CONT}	2
12	{CRE}	2
13	{MAN,CRE,CONT}	1
13	{DIR,MAN,CRE,CONT}	1
	overall	246

Table 6.3: Manual classification of *nach*-particle verb lemmas

As expected from the theory, many *nach*-particle verb lemmas are polysemous, i.e. related to several readings. Especially the predicted ambiguity between the MAN and the CRE reading (cf. Section 6.1, paragraph on readings of *nach*-particle verbs) is also reflected in the gold standard (cf. Table 6.3, rank 3). CRE even seems to rarely occur without another reading at all (cf. Table 6.3, rank 12). As in many other automatic processing steps, polysemy poses a problem for the automatic identification of a *nach*-particle verb reading. However, not all cluster combinations appear, which might suggest that the *nach*-particle verbs fall into a limited set of ambiguity classes. For this setting the clustering ap-

nach-particle verb reading	P	Pe	κ
DIR	0.889	0.689	0.642
MAN	0.716	0.642	0.205
CRE	0.865	0.774	0.403
OMR	0.755	0.626	0.346
CONT	0.876	0.811	0.344

Table 6.4: Inter-annotator agreement on *nach*-particle verb readings with respect to the lemmas

proach therefore sets the target number of clusters to 18, based on the different reading sets shown in Table 6.3.[28]

Gold standard for syntactic criteria. One additional resource has to be described here, which was not strictly necessary for the task, and is thus not part of the overview in Figure 6.2, but plays an important role for the parser combination:[29] A gold standard to evaluate each parser against, focussing on the syntactic criteria to be extracted. Chapter 5 describes the abstract workflow of task-based parser output combination, where a task-related parser evaluation takes place before the combination scheme is decided upon and applied. Since in Task I the relevant syntactic features are the dative and accusative arguments of a *nach*-particle verb lemma, the gold standard had to take the recognition of exactly these arguments into account.

The sentences for the gold standard were extracted based on 30 *nach*-particle verb lemmas equally distributed over three frequency ranges: high, middle, and low in the *nach*-particle verb sub-corpus described above. Furthermore, all of the five *nach*-particle verb readings from Haselbach (2011) had to be included. Table 6.5 shows the selected lemmas and their frequency in the sub-corpus. Where ten or more sentences per lemma were available, ten were

[28] See Section 6.1.5 for a short note on fuzzy clustering as an alternative setting.

[29] It also played a role for some insights in argument structure extension and reduction phenomena, which influence the indicator extraction, cf. Haselbach et al. (2012a).

HIGH, +man		**HIGH, -man**	
nachzeichnen	4168	nachfolgen	2741
nachbilden	1660	nachempfinden	2142
nachwachsen	1195	nachbessern	1776
nachlaufen	961	nachwirken	1088
nachfüllen	696	nachtrauern	739
MIDDLE, +man		**MIDDLE, -man**	
nachfließen	50	nachtanken	86
nachpflanzen	50	nachproduzieren	52
nachbasteln	34	nachreden	49
nachschärfen	25	nachpolieren	44
nachschreien	21	nachleuchten	31
LOW, +man		**LOW, -man**	
nachwandern	12	nachlackieren	9
nachtanzen	12	nachkolorieren	7
nachgären	8	nachturnen	7
nachfalten	5	nachfedern	6
nachölen	5	nachschminken	6

Table 6.5: Selected lemmas for syntactic gold standard.

chosen at random, otherwise all available sentences were extracted: all together 277 sentences.[30]

Each sentence was annotated for dative and accusative arguments of the indicated *nach*-particle verb by two linguistically trained annotators and three untrained annotators.[31] While from the untrained annotators only the information on the presence of a respective argument was taken into account, the trained annotators also marked the maximum extension of each argument.

Table 6.6, taken from Haselbach et al. (2012a), shows the inter-annotator agreement. There is not less than substantial agreement among the untrained

[30]While the frequency counts in Table 6.5 were based on the sub-corpus, the 277 sentences were extracted from the FSPar-parsed set. Due to differences in the identification of the particle verbs by the parsers, 9 sentences for *nachgären* and *nachkolorieren* and 7 sentences for *nachfedern* could be extracted.

[31]In case that there was more than one *nach*-particle verb in a sentence, only the indicated one from the lemma list had to be annotated.

indicator	κ (all annotators)	κ (trained annotators)
accusative	0.699	0.967
dative	0.869	0.985

Table 6.6: Inter-annotator agreement on indicators

annotators and an almost perfect agreement among the trained annotators (Landis and Koch 1977), the latter taking also the extension of the arguments into account.

The longest extension of the arguments was taken into the gold standard. Seven sentences were excluded, since they had been chosen due to wrongly analysed *nach*-particle verbs, e.g. Example (6.24) where the *nach* in the idiomatic *der Länge nach* ('lengthwise') was misinterpreted as a separated verb particle. 270 sentences remained in the gold standard.

(6.24) *Falten Sie das Papier der Länge nach.*
fold you the paper the length after
'Fold the paper lengthwise.'

6.1.2 Instantiated workflow and baselines

After having discussed the general workflow of Task I and the involved resources, we will now give a more technical analysis of the details of some workflow steps, especially where the parser output combination is concerned. For the task, the clustering uses syntactic features to find the semantically motivated reading classes of *nach*-particle verbs. The idea of the combination approach is to increase the reliability of the syntactic features to gain better predictions than when based on the output of single systems alone.

Task-related parser evaluation. To be able to decide upon a combination scheme for the parser output, the involved parsers have to be evaluated with respect to their coverage of those phenomena which are relevant for Task I,

i.e. the recognition of *nach*-particle verbs and of their dative and accusative arguments.

The first part of the task-related parser evaluation is a post hoc evaluation on the recognition of sentences with *nach*-particle verbs. It deals with the seven subsets described in the paragraph on the sub-corpus of sentences with *nach*-particle verbs in Section 6.1.1 and visualized in the Euler diagram in Figure 6.6: the subsets of sentences with the string *nach* for which at least one parser identifies a *nach*-particle verb. From each of these subsets and from the remaining set, in which none of the parsers identified a sentence to contain a *nach*-particle verb, 100 sentences were randomly extracted. These 800 sentences were manually annotated with three possible values:

1 in case at least one *nach*-particle verb occurred in the sentence,

0 in case no *nach*-particle verb occurred in the sentence, and

? in unclear or ambiguous cases.

As we utilize user generated content from the internet, many orthographic variants occur in the data sets, including e.g. forms written separately, where they should standardly be continuous, such as *nach zu fragen* instead of *nachzufragen* (to-infinitive of '[to] ask again / [to] check'). Thus the main guideline was to take every construction into account which showed the usage of a *nach*-particle verb. The annotation guidelines for this can be found in Appendix C.1. Five non-expert annotators were involved, and each sentence was rated by two different annotators. Conflicting cases and all cases which were marked as unclear or ambiguous by at least one annotator were decided upon by an expert (45 sentences), such that only the two annotation values **0** and **1** remained.

The quality of each of the combination subsets was then evaluated based on this manual annotation. For the different subsets of the parser agreement, the number of false positives was counted in the 100 sentence test set. For the evaluation set from the remaining sentences, which no parser identified to contain a *nach*-particle verb, the false negatives are counted. The result is shown in Table 6.7.

In the cases where all three parsers agree on either the presence (first row) or the absence (last row) of a *nach*-particle verb in a sentence, the decision of the parsers is rather reliable: there were no false negatives among the test

100 sentences from the set	# false positives
$M \cap F \cap B$	1
$(M \cap F) \setminus B$	4
$(M \cap B) \setminus F$	6
$(F \cap B) \setminus M$	9
$M \setminus (F \cup B)$	64
$F \setminus (M \cup B)$	31
$B \setminus (M \cup F)$	88
	# false negatives
$\mathcal{N} \setminus (M \cup F \cup B)$	0

Table 6.7: Evaluation of the parser combination approach with respect to recognizing sentences with *nach*-particle verbs. Number of false positives and false negatives respectively among 100 test sentences for every combination subset. As in Table 6.2, $\mathcal{N}$ is the set of all sentences containing the string *nach*, i.e. the input for each parser, M is the set of sentences with a *nach*-particle verb identified by the mate parser, F is the set of sentences with a *nach*-particle verb identified by FSPar, and B is the set of sentences with a *nach*-particle verb identified by BitPar.

sentences for the absence, and only one false positive among the test sentences for the presence. This false positive sentence is shown in Example (6.25)a. It contains the construction *sind nachgewiesen* ('are proven'), where *nachgewiesen* is a participle of the *nach*-particle verb *nachweisen* ('[to] prove/[to] verify'). In cases where this participle appears in combination with the passive auxiliary *werden* (lit.: '[to] become'), it denotes a verbal passive. Thus Example (6.25)b does indeed contain a *nach*-particle verb. However in combination with *sein* ('[to] be'), the participle *nachgewiesen* is part of an adjectival construction, and is no longer taken into account as a *nach*-particle verb.

(6.25) a. *Die Voraussetzungen des Satzes 1 Nr. 7 und 8* ***sind***
the requirements of the clause 1 no. 7 and 8 are
nachgewiesen*, wenn ein Integrationskurs erfolgreich*
proven when an integration course successfully
abgeschlossen wurde .
completed was

'The requirements of clause 1 no. 7 and 8 are proven, when an integration course was successfully completed.'

b. *Die Voraussetzungen des Satzes 1 Nr. 7 und 8* ***wurden nachgewiesen*** *, [...]*

When only two parsers agree on the presence of a *nach*-particle verb in a sentence, the number of false positives is still low, between 4 and 9 sentences. The errors of the mate parser/FSPar combination are mainly also cases of adjectival constructions as described above. The mate parser and BitPar tend to misinterpret the token *nach* as a separated verb particle, when it is used in an 'according to' reading, such as in *paraphrasiert nach [...]* ('paraphrased according to [...]') or in *Kunos Meinung nach* ('according to Kuno's oppinion'). The combination where only FSPar and BitPar agree on the presence of a *nach*-particle verb fails in cases where nouns or nominalisations are erroneously written with the first letter in lower case (such as verbs are in German) instead of upper case (as would be correct for nouns and nominalisations).

The most important result, however, is that the number of false positives increases substantially for the sets of sentences where only one parser identified a *nach*-particle verb in the sentence. For the mate parser and BitPar, the sets contain more false positives than true positives (64 and 88 of 100, respectively). The FSPar set contains only about a third false positives, which is due to the precision FSPar reaches on this phenomenon with its huge lexical knowledge base.

Each combination of even only two parsers already excludes many false positives from the output of the single parsers. Although we lose some instances of correctly identified *nach*-particle verbs, when we exclude the sets where only one parser identified a *nach*-particle verb, we also considerably reduce the number of false positives (in the case of the set where only BitPar identifies a *nach*-particle verb in the sentence, 12 correctly identified candidates would be lost, while 88 false positives could equally be removed). For mainly precision-oriented tasks such as Task I, this is highly beneficial.

Due to the fact that the tokenisation in the FSPar output deviates from the tokenization of the mate parser and of BitPar, and due to the differences in creating lemmata for a single *nach*-particle verb, we only evaluated if the parsers

recognize a sentence to contain a *nach*-particle verb. Although this evaluation does not focus on the recognition of a *nach*-particle verb lemma as such, it clearly suffices to assess which set of sentences should constitute the input for the feature extraction step in the clustering approach.

The evaluation regarding the argument structure was conducted on the gold standard for syntactic criteria described above. Since the extraction of the respective arguments presupposes that the parser correctly identified the *nach*-particle verb in the sentence, all 277 sentences were taken into account (i.e. including those, with a *nach* in the sentence, which could be misinterpreted as a separated verb particle), and the evaluation also states the recognition of *nach*-particle verbs for these sentences.

The two other features are the recognition of dative and accusative arguments of the *nach*-particle verb respectively. An annotation from any of the dependency parsers is counted as correct, if a respective argument was extracted and the annotated head of the argument is contained in the string from the gold standard, i.e. the longest possible extension for this argument. In case of the constituency parser, the extracted argument is a phrase. If the span of this phrase is contained in, contains or is equal to the longest possible extension for this argument from the gold standard the annotation is counted as correct. Since FSPar's output can be underspecified in that way that the right annotation can still be derived from the output, but is not the only possibility, there are three values taken into account for FSPar: (i) an upper bound, always counting a result as correct, if the correct annotation can be derived from the output, (ii) a lower bound, only counting a result as correct, if the annotation is correct and not underspecified, and (iii) a chance value, where in case of underspecification one of the possible values was selected by chance and taken into account for the evaluation.

Table 6.8 shows the results of the evaluation for each parser. Since the sentences for the gold standard were extracted from the FSPar-based set, FSPar has a recall of 100% for the recognition of the *nach*-particle verbs. However, as mentioned above, the sentences with the misinterpreted *nach* were kept, which is reflected by FSPar's precision being smaller than 100%. Regarding

	mate parser				FSPar			BitPar		
	prec	rec	f1		prec	rec	f1	prec	rec	f1
nach-particle verb recognition	93.62	95.65	94.62		97.53	100.0	98.75	96.51	80.07	87.52
dative recognition	87.18	56.67	68.69	upper bound	61.11	91.67	73.33	89.19	55.00	68.04
				lower bound	50.00	75.00	60.00			
				chance	52.22	78.33	62.67			
accusative recognition	53.12	67.11	59.30	upper bound	46.21	88.16	60.63	54.76	60.53	57.50
				lower bound	26.90	51.32	35.29			
				chance	32.41	61.84	42.53			

Table 6.8: Task-related evaluation of the single parsers on the gold standard for syntactic criteria.

the dative and accusative recognition, most values are below 70%, with the exception of BitPar's and the mate parser's precision of dative recognition, FSPar's upper bound for f1 in the dative recognition, and some of FSPar's recall values. However, lower values are to some extent expected due to the domain shift into web data. The goal of combination would be to increase the reliability of the features especially in these cases.

Extracting syntactic features. In Section 6.1.1 we have presented the criteria depending on which a sentence containing a *nach*-particle verb was identified by the parsers. Now we do the same regarding the extraction of the syntactic features for the clustering (Box A② in Figure 6.2).

For each *nach*-particle verb lemma we extract two kinds of features: (i) features on the occurrence of dative and accusative arguments with the *nach*-particle verb lemma, and (ii) features on the form of a dative or accusative argument head appearing with the *nach*-particle verb lemma. We denote the features from (i) 'has_dative' and 'has_accusative', and the features from (ii) as a combination of the form of the argument head and the case, e.g. 'dative-Hasen' for a dative argument with the head form *Hasen* ('hare.DAT') and 'accusative-Wörter' for an accusative argument with the head form *Wörter* ('words.ACC'). The weight of each feature is its relative frequency, i.e. the ratio of the number of its occurences with respect to a specific *nach*-particle verb lemma to the frequency of this lemma in the set of input sentences.

FSPar: To identify a relevant argument in the output of FSPar, we consider the dependents of the *nach*-particle verb. If the edge annotation contains NP:4 or NP:24 or in the case when the edge annotation contains ACMP or COMP and dative case is among the predictions of the morphology annotation, but the part-of-speech tag is not APPR or APPRART, we identify a dative argument. In the case when the edge annotation contains NP:8, NP:18, or NP:28 or in the case when the edge annotation contains ACMP or COMP and accusative case is among the morphology predictions, but the part-of-speech tag is not APPR or APPRART, we identify an accusative argument. We exclude however those accusative arguments which appear in a

passive construction, denoted by the annotation "PPart" in the morpology annotation of the *nach*-particle verb and by the presence of a head with the lemma "werdenP".

The mate parser: In the output of the mate parser, we extract relevant arguments by means of the dependency label DA for a dative argument and OA for an accusative argument. Thereby we exclude those accusative arguments, which occur in a passive construction, i.e. when the *nach*-particle verb has the part-of-speech tag VVPP denoting a participle, and its head is a verbal form of *werden* ('[to] become').

BitPar: To identify the accusative and dative arguments of a *nach*-particle verb in the BitPar output, we have to identify the siblings of the *nach*-particle verb with the syntactic functions OA for an accusative argument and DA for a dative argument.

Clustering. For the clustering step (Box A③ in Figure 6.2) we applied a standard approach: Ward's algorithm. Ward (1963) describes a hierarchical clustering approach which builds a tree based on a set of n input objects: The tree consists of n levels, where the nodes on each level l build a group of mutually exclusive subsets of the set of input objects – the result of the clustering for l target clusters.

Let n be the number of input objects k_i $(i\ in\ 1..n)$ and m_i the set of features pertaining to input object k_i. Then the output tree $\mathcal{W}$ is built such that:

- $\mathcal{W}$ consists of n levels;
- each node c of $\mathcal{W}$ is a set of input objects k_i;
- for each level l, the union of all nodes on that level, $c_{j,l}$, is the set of all input objects;
- the leaves $c_{i,n}$, consist of sets with exactly one input object k_i;
- for each level l $(l < n)$ exists a node $c_{j,l} = c_{a,l-1} \cup c_{b,l-1} (a \neq b)$ with $Z(c_{a,l-1}, c_{b,l-1}) = min(Z(c_{x,l-1}, c_{y,l-1}))(x \neq y)$, where Z is a function

describing the "information loss" based on the similarity of the features m_i, when treating the two subsets as one.

From this follows that the root, $c_{1,1}$, is the set of all input objects and therefore the result of the trivial clustering with the number of target clusters set to 1. Starting from the leaves, in each step the number of clusters is reduced by one, and those two subsets are combined, for which the information loss is minimal compared to the other possible combination pairs on that level.

In the implementation of our approach towards the classification of *nach*-particle verb readings, the clustering and the evaluation step only take those *nach*-particle verbs into account which appear in the evaluation set and the corpus data, and which have been manually labelled to belong to at least one of the five reading classes from Haselbach (2011).

Evaluation of the outcome of the clustering. We evaluate the resulting clusters against the lemma-based gold standard for *nach*-particle verb readings by means of the V-measure by Rosenberg and Hirschberg (2007). The V-measure is the harmonic mean of two cluster criteria: homogeneity, indicating if a cluster mainly consists of similar elements expected to be in the same cluster, and completeness, which states from an overall viewpoint whether all elements of the same type can be found in the same cluster. Based on the explanation of two trivial cases by Rosenberg and Hirschberg (2007) this trade-off between two competing goals can be illustrated for our *nach*-particle verb study: In case each *nach*-particle verb ends up in its own cluster, we obtain perfect homogeneity, since no cluster contains elements from different classes. However this decreases completeness, since the elements which belong to the same class are in different clusters.[32] The other trivial case would be, that there is only one cluster left, containing all *nach*-particle verbs. This would satisfy the completeness criterion, since all *nach*-particle verbs of the same class are in fact in the same cluster, however it minimizes homogeneity since one cluster contains elements from

[32]This is of course only true, if there are at least two elements of the same class part of the dataset for the clustering.

features (added up)	homogeneity			completeness			V-measure		
	mate	FSPar	BitPar	mate	FSPar	BitPar	mate	FSPar	BitPar
dat,acc	32.96	35.47	36.61	28.20	28.50	30.37	30.39	31.61	33.20
+ form	33.24	33.70	32.83	30.27	29.33	30.03	31.68	31.37	31.37

Table 6.9: Baselines: Clustering evaluation results when each parser is applied individually.

(all) different classes.[33] The V-measure therefore monitors the trade-off between those two criteria regarding the distance between the result of the clustering and the expected classification.

Baselines. As discussed in Chapter 1 and Section 1.3, the idea of system combination in natural language processing is based on the hypothesis that combined results outperform the result of the best single system involved in the combination. Thus our baselines for the task-based combination case studies are the results of the overall task, as achieved when only one of the parsers is applied for extraction of the syntactic features. Table 6.9 shows the values for homogeneity, completeness and the combined V-measure for the clustering result with respect to the different parsers BitPar, FSPar and the mate parser.

To create these baselines, the workflow described in the upper chain of Figure 6.2 was instantiated 3 times; for the extraction of sentences with *nach*-particle verbs and the extraction of the syntactic criteria, each time one of the parsers was used in isolation. Additionally, only *nach*-particle verb lemmas with a corpus frequency greater than 10 are taken into account.

As described in the evaluation procedure above, here we also distinguish between only taking the existence of a dative or accusative into account (Table 6.9, *dat,acc*) and additionally including information about the token form of the argument (+*form*), such as the head as described above in the paragraph of extracting syntactic features.

[33] This second case is of course only true, if there are at least two elements of different classes part of the dataset for the clustering.

The overall numbers reagarding the evaluation of Task I are not very high. Amongst others, this can be due to linguistic phenomena which influence the sentence structure, e.g. by triggering argument structure reduction or argument structure extension, and which thereby influence the possibility to automatically extract the expected indicators. For a discussion on this see Haselbach et al. (2012a). However, we are interested in the effect of combining several parsers for this task, which is evaluated in two case studies in Sections 6.1.3 and 6.1.4. That is, here we are interested in the difference between the numbers from the combined approach and the best single system rather than in the absolute numbers.

Comparing the baselines we see some differences with regard to the parsers and with regard to the two sets of features taken into account. For the first configuration (only presence of an argument) BitPar can be seen as the best system, followed by FSPar and the mate parser. For the second configuration (added form), the systems perform more similar, especially with respect to the V-measure. It should be noted, that these results are the evaluation baselines and played no role in the decision on the combination schemes utilized in the case studies (for Case study I.i they were not known in advance at all). The combination scheme was only decided upon based on the evaluation concerning the extraction of the syntactic features.

Role of the B3DB database. Since the scaling functionalities of the database have been developed in conjunction with the case studies of Task I, not all of the baseline and combination workflows could be executed completely in the database environment at that point of time. Parts of the creation of the gold standards, the extraction of the relevant test sentences, and the identification of the *nach*-particle verb form and the extraction of the syntactic features with respect to the BitPar output are based on B3DB queries and functions. Additionally, there are scripts utilized in the baseline and combination workflows which reflect that part of the work has been executed outside of the database.[34] It is

[34]The implementations of the scripts applied for the baselines and case studies are based on and adapted from the original implementation by Wolfgang Seeker applied for Haselbach et al. (2012a).

technically feasible to create database functions which conduct the workflow steps of the whole feature extraction and combination inside of the database, however, the clustering step and its evaluation would be conducted externally.

6.1.3 Case study I.i: combination type 2, two parsers, combination rules

For this case study we combine the output of the mate parser and FSPar. Both are dependency parsers and apply part-of-speech tags based on the STTS tag set in preprocessing. However, they employ different sets of concepts regarding the assigned dependency labels: the models we use for the mate parser are trained on a dependency conversion of the TIGER corpus, while the roles applied by FSPar originate from its own rule-based approach. Thus, this setting is of combination type 2, cf. Section 3.3, since the involved tools are similar with respect ot their output structure (dependency arcs), but different with respect to the semantics of their concepts (dependency labels).

As in Haselbach et al. (2012a), for the feature extraction we take the union of the set of sentences in which the mate parser identified a *nach*-particle verb and the set of sentences in which FSPar identified a *nach*-particle verb into account, and include only those *nach*-particle verb lemmas, which occur more than 10 times in the corpus. In the case that both parsers identify the *nach*-particle verb in the sentence, the mate parser based on the token number and FSPar based on the lemma, a set of combination rules is applied. In the case that only one parser recoginizes a *nach*-particle verb, the features from this parser are taken into account.

The combination scheme (cf. Section 5.2.4) we apply for the features is thus a set of combination rules, based on the findings of the task-related parser evaluation described in Section 6.1.2.[35] The lemma-based clustering with features from automatic parsing can thus be improved in two places: by increasing the reliabiltiy of the features stating the presence or absence of a dative or accusative argument for a *nach*-particle verb, and by increasing the reliability of

[35]The combination rules have been developed in a discussion of this author and Wolfgang Seeker.

the features indicating a specific argument form.

Tables 6.10 and 6.11 show the combination rules. In this combination scheme we distinguish between rules per annotation, i.e. the presence or absence of an argument for a *nach*-particle verb in a concrete sentence (Tables 6.10a and 6.10b), and rules per argument, i.e. the label each parser predicts for a specific argument head (Table 6.11). To illustrate the application of the rules, we use Example (6.26). In this example the *nach*-particle verb, marked in boldface, is *nachsingen* ('[to] sing (after)') and has a dative argument *Peer* and an accusative argument *ein Lied* ('a song'). Figures 6.7 and 6.8 show the analyses by the mate parser and FSPar, respectively.

(6.26) *[...] und endet mit dem Blick auf Solveig, die auf der*
[...] and ends with a view on Solveig who on the
dunklen Bühne sitzt und Peer in die Wüste Afrikas ein
dark stage sits and Peer.DAT in the desert Africa's a
Lied ***nachsingt.***
song.ACC sings after
'[...] and ends with a view on Solveig, who sits on the dark stage and sings a song which follows Peer into the desert of Africa.'

The rules per annotation refer to each *nach*-particle verb in the sentence and add or increase the occurrence value of the features 'has_accusative' and 'has_dative', respectively, for the *nach*-particle verb lemma in case an accusative or a dative argument for this *nach*-particle verb is extracted by the combination rules. For the sentence in Example (6.26), both feature values should be increased for the lemma *nachsingen*. The rules per argument refer to each argument of a *nach*-particle verb in the sentence. In case the combination rules decide for a specific argument to be either an accusative or a dative argument, the form of the argument head and its case are added as a feature to the *nach*-particle verb lemma. For the sentence in Example (6.26), the features 'accusative-Lied' and 'dative-Peer' should be added (or their occurrence value increased) for the lemma *nachsingen*.

```
<s>
     ...
 20  und        KON     und                     |                                              -1   TOP
 21  endet      VVIMP   enden                   3:Sg:Pres:Ind|2:Pl:Pres:Ind|2:Pl:Pres:Konj     -1   TOP
 22  mit        APPR    mit                     Dat                                            21   ADJ|PP/mit:4
 23  dem                ART                     d                                               |   24                SPEC
 24  Blick      NN      Blick                   Dat:M:Sg                                       22   PCMP
 25  auf        APPR    auf                     Dat|Akk                                    21||24   ADJ
 26  Solveig    NE      Solveig:H               Dat:M:Sg|Akk:M:Sg|Dat:F:Sg|Akk:F:Sg            25   PCMP
 27  ,          $,      ,                       |                                              21   PUNCT
 28  die        PRELS   d                       Nom:F:Sg|Akk:F:Sg|Nom:Pl|Akk:Pl             33&42   NP:1&NP:1|NP:8      26
 29  auf        APPR    auf                     Dat                                            33   ADJ
 30  der        ART     d                       |                                              32   SPEC
 31  dunklen    ADJA    dunkel                  |                                              32   ADJ
 32  Bühne      NN      Bühne                   Dat:F:Sg                                       29   PCMP
 33  sitzt      VVFIN   sitzen                                                                 26   RC
 34  und        KON     und                     |                                           33$42   KON
 35  Peer       NE      Peer:H|Peer:Stadt       Dat:Sg                                         42   NP:4
 36  in         APPR    in                      Akk                                        42||35   ADJ
 37  die        ART     d                       |                                              38   SPEC
 38  Wüste      NN      Wüste                   Akk:F:Sg                                       36   PCMP
 39  Afrikas    NE      Afrika:H|Afrika:Region  Gen:Sg                                         38   GR
 40  ein        ART     ein                     |                                              41   SPEC
 41  Lied       NN      Lied                    Nom:N:Sg|Akk:N:Sg                              42   NP:8|NP:1
 42  nachsingt  VVFIN   nach#singen                                                            26   RC
 43  .          $.      .                       |                                              -1   TOP
</s>
```

Figure 6.7: Excerpt of an analysis by FSParfor Example (6.26).

22	und	und	KON	_	11	CD
23	endet	enden	VVFIN	sg | 3 | pres | ind	22	CJ
24	mit	mit	APPR	_	23	MO
25	dem	der	ART	dat | sg | masc	26	NK
26	Blick	Blick	NN	dat | sg | masc	24	NK
27	auf	auf	APPR	_	26	MNR
28	Solveig	Solveig	NE	dat | sg | neut	27	NK
29	,	–	$,	_	28	–
30	die	der	PRELS	nom | sg | fem	35	SB
31	auf	auf	APPR	_	35	MO
32	der	der	ART	dat | sg | fem	34	NK
33	dunklen	dunkel	ADJA	dat | sg | fem | pos	34	NK
34	Bühne	Bühne	NN	dat | sg | fem	31	NK
35	sitzt	sitzen	VVFIN	sg | 3 | pres | ind	26	RC
36	und	und	KON	_	35	CD
37	Peer	Peer	NN	nom | sg | masc	44	OA
38	in	in	APPR	_	44	MO
39	die	der	ART	acc | sg | fem	40	NK
40	Wüste	Wüste	NN	acc | sg | fem	38	NK
41	Afrikas	Afrikas	NE	nom | sg | neut	40	AG
42	ein	ein	ART	acc | sg | neut	43	NK
43	Lied	Lied	NN	acc | sg | neut	44	OA
44	nachsingt	nachsingt	VVFIN	sg | 3 | pres | ind	36	CJ
45	.	–	$.	_	44	–

Figure 6.8: Excerpt of an analysis by the mate parser for Example (6.26).

Next to the representation in Tables 6.10 and 6.11 we display the combination rules in a notation similar to a formula in propositional logic. Each rule is represented as a conditional ($\rightarrow$), where the antecedent is composed of propositions about the decisions of the parsers and the truth-functional operators for negation ($\neg$), conjunction ($\wedge$), and disjunction ($\vee$). The consequent is the value returned by the combination rule. As for the antecedents, we let $\textsc{y}_{parserX}$ be the truth value of the proposition "the decision of parserX is y", where y can be of different forms, depending on the parser output (underspecified vs. fully specified) and on the rule type (per annotation vs. per argument). To give an example: $\ominus\textsc{dat}_{mate}$ is true, when the mate parser does not identify a dative argument for a specific *nach*-particle verb in a sentence, while $\oplus\textsc{dat} \ominus \textsc{acc}_{fspar}$ is true when FSPar identifies an argument of a *nach*-particle verb as a possible dative argument in an underspecified setting, while the alternatives do not comprise an accusative, but possibly, e.g. a genitive. We discuss the full set

of propositions per parser below. As values for the consequent we let $\ominus$VAL denote the absence of an argument of case VAL, $\oplus$VAL denote the presence of an argument of case VAL, or the identification of a specific argument as of case VAL; 0 denotes that a specific argument is neither an accusative nor a dative argument, and *parserX* denotes that the outcome of the rule is the decision of parserX among the values of $\{\oplus$DAT, $\ominus$DAT, $\oplus$ACC, $\ominus$ACC, $0\}$.

We start with the description of the combination rules per annotation, where we treat dative and accusative arguments separately, cf. Table 6.10. In each table, the first column to the left shows the decision of FSPar, while the first row shows the decision of the mate parser, which are also the possible values for the propositions in the rule formula. The mate parser has always two options: the presence of a respective argument ($\oplus$DAT, $\oplus$ACC) or its absence ($\ominus$DAT, $\ominus$ACC). For FSPar, we also differentiate in case of the presence of a respective argument between fully specified (s) and underspecified (u) occurrences, thus there are three options for FSPar. Since the concept of FSPar's upper bound reflects the possible presence of an argument, it fits the idea of the per-annotation-rules and is thus highly taken into account for these combination rules.

FSPar \ mate	$\oplus$DAT	$\ominus$DAT
$\oplus$DAT s	$\oplus$DAT	$\oplus$DAT
$\oplus$DAT u	$\oplus$DAT	$\oplus$DAT
$\ominus$DAT	$\oplus$DAT	$\ominus$DAT

(a) Dative

FSPar \ mate	$\oplus$ACC	$\ominus$ACC
$\oplus$ACC s	$\oplus$ACC	$\ominus$ACC
$\oplus$ACC u	$\oplus$ACC	$\oplus$ACC
$\ominus$ACC	$\ominus$ACC	$\ominus$ACC

(b) Accusative

Table 6.10: Combination rules per annotation.

Combination rules per annotation: dative. In the results of the task-related parser evaluation, cf. Section 6.1.2, we find high precision of the mate parser

with respect to dative recognition (87.18%) as well as high recall for FSPar's dative upper bound (91.67%). The deduced rules are:

$$\oplus\textsc{dat}_{mate} \vee \oplus\textsc{dat}_{fspar} \rightarrow \oplus\textsc{dat} \tag{6.27a}$$

$$\ominus\textsc{dat}_{mate} \wedge \ominus\textsc{dat}_{fspar} \rightarrow \ominus\textsc{dat} \tag{6.27b}$$

Whenever one of the two parsers recognizes a dative argument, we count this as the occurrence of a dative with the respective *nach*-particle verb (Equation (6.27)a). Only when both parsers aggree on the absence of a dative argument, we take this as the result of the combination rule (Equation (6.27)b). All combination cases which result from this are listed in Table 6.10a.

For Example (6.26), FSPar correctly identifies the dative argument (NP:4, Figure 6.7, Token 35) while the mate parser annotates an accusative argument instead (OA, Figure 6.8, Token 37). However, with the applicable combination rule Equation (6.27a), the presence of the dative is correctly extracted as a feature.

Combination rules per annotation: accusative. Based on the values of the accusative recognition, cf. Section 6.1.2, the only high value is the upper bound recall of FSPar (88.16%). The worst number is however FSPar's lower bound precision, which only takes the fully specified cases into account. Thus the combination rules for presence or absence of an accusative argument are as follows:

$$\ominus\textsc{acc}_{fspar} \vee \oplus\textsc{acc}\ \mathtt{u}_{fspar} \rightarrow fspar \tag{6.28a}$$

$$\oplus\textsc{acc}\ \mathtt{s}_{fspar} \rightarrow mate \tag{6.28b}$$

We opt for FSPar's decision in the underspecified case as well as in the case of the predicted absence of an accusative argument (Equation (6.28)a), and we opt for the decision provided by the mate parser, when FSPar's result is based on a fully specified annotation (Equation (6.28)b). All combination cases which result from this are listed in Table 6.10b.

For Example (6.26), the mate parser correctly annotates the accusative argument (OA, Figure 6.8, Token 43) and FSPar's analysis is underspecified, allowing for a subject or the (correct) accusative argument (NP:8|NP:1, Figure 6.7, Token 41). In this case, Equation (6.28a) is applied. Again the correct feature, the presence of an accusative, is extracted from this combination.

FSPar \ mate	⊕DAT	⊕ACC	0
⊕DAT S	⊕DAT	⊕DAT	⊕DAT
⊕DAT ⊖ACC	⊕DAT	⊕DAT	0
⊕ACC S	⊕ACC	⊕ACC	⊕ACC
⊕ACC ⊖DAT	⊕DAT	⊕ACC	0
⊕DAT ⊕ACC	⊕DAT	⊕ACC	0
0	⊕DAT	⊕ACC	0

Table 6.11: Combination rules per argument.

When we change the viewpoint and focus on the token forms which are the specific argument heads, i.e. look at the predictions which each parser can make for them, we have to take some more possibilities into account: The mate parser can predict an argument to be a dative (⊕DAT), an accusative (⊕ACC) or neither (0). For FSPar, we have to differentiate between fully specified and underspecified analyses. In the fully specified cases it can predict an argument to be a dative or an accusative as its single option (⊕DAT S, ⊕ACC S). Within FSPar's underspecified annotations, there are four possiblities: an argument can be (i) a dative, but no accusative (⊕DAT⊖ACC), (ii) an accusative but no dative (⊕ACC⊖DAT), (iii) an accusative or a dative (⊕DAT⊕ACC); the last possibility (iv) is that FSPar does not predict a head to be a dative or accusative argument at all (0). In this setting FSPar's upper bound recall values are no longer telling, since for the per-argument-rules, the specific argument form is taken into account rather than the mere presence or absence of the respective argument type.

Combination rules per argument. The values of the mate parser for dative as well as accusative recognition are higher than FSPar's respective chance values (but for the recall value of the dative recognition). However, for both parsers,

dative recognition is easier than accusative recognition. Thus we derive the following combination rules:

$$\oplus\textsc{dat}_{mate} \wedge (\neg \oplus \textsc{acc}\ \mathtt{s}_{fspar}) \rightarrow \oplus\textsc{dat} \tag{6.29a}$$

$$\oplus\textsc{dat}_{mate} \wedge \oplus\textsc{acc}\ \mathtt{s}_{fspar} \rightarrow \oplus\textsc{acc} \tag{6.29b}$$

$$\oplus\textsc{acc}_{mate} \wedge (\oplus\textsc{dat}\ \mathtt{s}_{fspar} \vee \oplus\textsc{dat} \ominus \textsc{acc}_{fspar}) \rightarrow \oplus\textsc{dat} \tag{6.29c}$$

$$\oplus\textsc{acc}_{mate} \wedge (\neg(\oplus\textsc{dat}\ \mathtt{s}_{fspar} \vee \oplus\textsc{dat} \ominus \textsc{acc}_{fspar})) \rightarrow \oplus\textsc{acc} \tag{6.29d}$$

$$0_{mate} \wedge (\neg(\oplus\textsc{dat}\ \mathtt{s}_{fspar} \vee \oplus\textsc{acc}\ \mathtt{s}_{fspar})) \rightarrow 0 \tag{6.29e}$$

$$0_{mate} \wedge (\oplus\textsc{dat}\ \mathtt{s}_{fspar} \vee \oplus\textsc{acc}\ \mathtt{s}_{fspar}) \rightarrow fspar \tag{6.29f}$$

Again we go with the dative if at least one of the parsers considers an argument to be a dative rather than an accusative (Equation (6.29)a,c). There are only two exceptions to this: When FSPar rules out an underspecified setting and opts for an accusative (Equation (6.29)b), and when the mate parser predicts an argument to be neither accusative nor dative, while FSPar's output is underspecified anyway (Equation (6.29)e). For the remaining combination cases we also opt for the decision of the mate parser most of the time (Equation (6.29)d,e), except for the case when again FSPar rules out an underspecified setting in favour of an accusative, while the mate parser neither predicts an accusative nor a dative argument (Equation (6.29)f). Thus, overall we go with the decision of the mate parser most of the time. All combination cases which result from this are listed in Table 6.11.

For Example (6.26), we find the case where for the potential argument *ein Lied*, the mate parser predicts an accusative argument (OA, Figure 6.8, Token 43) while FSPar's prediction is underspecified accusative, but no dative (NP:8|NP:1, Figure 6.7, Token 41), thus the token form is correctly extracted as an accusative argument (Equation (6.29)d). The mate parser also predicts *Peer*, to be an accusative argument (OA, Figure 6.8, Token 37), while FSPar's single analysis is the correct dative argument (NP:4, Figure 6.7, Token 35). Again the respective combination rule (Equation (6.29)c) correctly extracts the dative argument.

Regarding the analyses for Example (6.26), both parsers did not produce the single perfect analysis (the mate parser predicted two accusative arguments, instead of an accusative and a dative, and FSPar's analysis of the accusative argument was part of an underspecified analysis). However with the combination rules, both, the occurrence and the token forms of the arguments were correctly extracted.

Task-based evaluation of the combination approach. Table 6.12 shows the results of the *nach*-particle verb clustering with the combined syntactic features. It also repeats the baselines for the mate parser and for FSPar, to allow for an easy comparison to the combination results.[36] The rows marked with *dat,acc* show the results which take only the presence or absence of a respective argument with a *nach*-particle verb lemma into account, while the rows marked with *+form* additionally take the form of the argument head into account. The combined V-measure outperforms the better single system in both cases. This is also true for the homogeneity values and the completeness values.

features (added up)	homogeneity		completeness		V-measure	
	mate	FSPar	mate	FSPar	mate	FSPar
	combined		combined		combined	
dat,acc	32.96	35.47	28.20	28.50	30.39	31.61
	38.72		30.88		34.36	
+ form	33.24	33.70	30.27	29.33	31.68	31.37
	36.43		32.09		34.12	

Table 6.12: Clustering: features from the mate parser, FSPar and their combination.

[36]The values shown here partly deviate form the values reported by Haselbach et al. (2012a). This is due to the fact, that the setting here explicitly excludes annotations with part-of-speech tag APPR and APPRART when an argument is extracted from an FSPar analysis based on the morphology information (see the section on extracting syntactic features in Section 6.1.2) and due to a reimplementation of one of the per argument rules. While the baseline for FSPar decreases for the presence of an argument and increases for its form, the overall combination values increased.

6.1.4 Case study I.ii: combination type 3, three parsers, majority vote

For the second case study in Task I, we add information from BitPar. Since the output of BitPar are constituency trees, this case study is of combination type 3, combining parser output of different structures (dependency and constituency syntax) and different label semantics. Similar to the mate parser and FSPar, BitPar also applies part-of-speech tags based on STTS in pre-processing. Furthermore, its grammar was derived from the TIGER corpus, such that there is a similarity between the labels of the mate parser and those of BitPar. However the label set is different from the labels used in FSPar's rule-based approach.

Combination scheme. For the combination we apply a simple majority voting scheme. To be able to compare the case studies, we utilize the same set of sentences which was applied in Case study I.i, i.e. the union of the sentences in which FSPar recognizes a *nach*-particle verb with the sentences in which the mate parser recognizes a *nach*-particle verb. Information from BitPar is only taken into account for those sentences where all three parsers identify a *nach*-particle verb in the sentence. In all other cases the script falls back to the combination scheme from Case study I.i. Again, only *nach*-particle verb lemmas with a corpus frequency greater than 10 are taken into account.

features (added up)	homogeneity			completeness			V-measure		
	mate	FSPar	BitPar	mate	FSPar	BitPar	mate	FSPar	BitPar
	combined			combined			combined		
dat,acc	32.96	35.47	36.61	28.20	28.50	30.37	30.39	31.61	33.20
	39.30			32.69			35.69		
+ form	33.24	33.70	32.83	30.27	29.33	30.03	31.68	31.37	31.37
	36.56			32.98			34.67		

Table 6.13: Clustering: features from the mate parser, FSPar, BitPar and their combination.

Task-based evaluation of the combination approach. Table 6.13 shows the results of the second case study. The baselines of the mate parser, FSPar and BitPar are shown in the table as well. The *dat,acc* row shows the results when only the feature of presence or absence of an argument is taken into account, the row marked with *+form* adds the form of the argument head as a feature. Again, all combined values for homogeneity, completeness and the V-measure outperform the respective best single system. Section 6.1.5 compares the results of the two conducted case studies for Task I and discusses the applicability of the workflow for task-based parser output combination with respect to this task.

6.1.5 Discussion of Task I

In the context of this thesis, Task I serves as an exemplification of the task-based parser output combination workflow and methodology introduced by this work, cf. Chapter 5. Thus, we need to discuss some issues, such as the applicability of the workflow, where we find the proposed steps of the combination workflow in the description of Task I and if the combination approach yields benefits when applied for the task.

The preconditions for the applicability of the combination workflow as defined in Chapter 5 are that (i) the task has to be known in advance, and (ii) only specific features from the parser output are relevant for the task. These preconditions are met by Task I:

(i) the task, known in advance, is to classify *nach*-particle verbs, based on their argument structure as predicted by Haselbach (2011) and on their occurrence in web text, and

(ii) the relevant features from the parser output are *nach*-particle verbs and their dative and accusative arguments.

Next, we go through the abstract steps of the proposed combination workflow (cf. Figure 5.1, Page 183) and link them to their appearance in the actual workflow of Task I (cf. Figure 6.2). Since the task workflow consists of many steps, the steps of the combination workflow are somewhat hidden within the description in Section 6.1. The step of **parser output inspection** with respect

to the relevant features is described in two parts: for the representation of the analysed *nach*-particle verbs in the parser output, see the paragraph on the sub-corpus of sentences with *nach*-particle verbs in Section 6.1.1; for the representation of dative and accusative arguments see the paragraph on extracting syntactic features in Section 6.1.2. The **task-related manual annotation set** comprises 800 sentences from the sub-corpus of sentences with *nach*-particle verbs, as well as the gold standard for syntactic criteria described in Section 6.1.1.The **task-related parser evaluation** based on these manual annotation sets is described in Section 6.1.2, and the way the combination schemes were decided upon and applied is described in the subsections of the case studies, i.e. Sections 6.1.3 and 6.1.4. Although well hidden, we find all steps from our abstract combination workflow within Task I.

The third aspect which is important for this discussion is then, if our prediction for the usefulness of the combination approach holds for Task I. It holds, since in both case studies the results based on the combined features outperform the best single system with respect to the V-measure – the combination of the two cluster properties homogeneity and completeness. Table 6.14 shows this by repeating the V-measures from the case studies and the baselines of the single systems.

	features (added up)	mate	FSPar	BitPar
single systems	dat,acc	30.39	31.61	33.20
	+ form	31.68	31.37	31.37
combination type 2, rule-based	dat,acc	34.36 (+2.75)		
	+ form	34.12 (+2.44)		
combination type 3, majority vote	dat,acc	35.69 (+2.49)		
	+ form	34.67 (+2.99)		

Table 6.14: Summary of the V-measures: baselines from the single systems and results of the case studies with the difference between combination and best involved single system in brackets.

Interestingly, in both case studies the gain of the combination over the respective best single system involved in this combinations setting is similar, indicating that the parsers applied for the combination type 2 study were already

distinct enough to produce a notable gain in combination. This similarity between the two settings might be also due to the fact, that the case study with all three parsers employs the rule-based combination scheme as a fallback solution. However there is still a small improvement for the combination type 3 study over the study of combination type 2.

Even though creating a rule-based combination scheme includes more manual effort than applying a majority vote, Case study I.i is an example of the fact that also in cases where only two systems are available, benefits can be achieved by creating a small gold standard and inferring combination rules from it. It is however also important that the systems diverge to a certain extent (such as the rule-based FSPar and the data-driven mate parser), such that the results of the single systems do not show errors in the same cases.

Albeit the task is meant as an exemplification in the context of this work, let us have some last remarks on the results with respect to the task of classifying *nach*-particle verbs itself. As we have seen before, the results of the clustering range between a V-measure of 30.39 for one of the single systems taking only the occurrence of the arguments into account and 35.69 for the combination of all three systems, also in the setting where only the occurrence is taken into account. Overall this are not high numbers. With regard to this, Haselbach et al. (2012a) show results for single target clusters where the quality of the clustering varies between the different classes, and they discuss argument structure reduction and extension phenomena which influence the ability to extract the correct features from the parses. One further technical aspect is the choice of the clustering method, also hinted at by Haselbach et al. (2012a). A fuzzy clustering approach and a respective evaluation such as described by Utt et al. (2014) might be suited to represent the ambiguity of many *nach*-particle verb lemmas, while reducing the number of target clusters to match the number of the different inspected readings. Another interesting factor is that adding the form of the argument as a feature to the clustering does only enhance the results in the case of the mate parser applied as a single system, i.e. the presence or absence of the arguments is already an important indicator for a specific reading in this setting.

6.2 Task II: Recognizing phrases for information status annotation

Baumann and Riester (2012) propose a two-dimensional annotation scheme, the RefLex scheme, to classify discourse expressions according to their givenness. They claim that two different notions of givenness play a role in the prosodic realisation of the respective phrases, referential givenness and lexical givenness.

Referential givenness is connected to the existence of a coreference relation, cf. Example (6.30). Lexical givenness is based on the reappearance or entailment of a lexical unit, which does not necessarily refer to the same entity, cf. Example (6.31).[37]

(6.30) While chatting, we suddenly felt watched by someone from the bar.
The man with the small dog came up to us.

(6.31) Look at the funny dog over there! It makes me think of Anna's dog.

When annotating RefLex labels, referential information status is assigned to referring expressions, i.e. on the syntactic level of noun phrases (NP), determiner phrases (DP), prepositional phrases (PP)[38] and their embedding. Lexical information status is assigned at the word level. Thus the full annotation is a hierarchical one, where phrases and their subphrases can each carry independent information status labels, cf. Example (6.32) in the context of Example (6.30).

(6.32) (the (man)$_{\text{L-ACCESSIBLE}}$ with (the (small)$_{\text{L-NEW}}$ (dog)$_{\text{L-NEW}}$)$_{\text{R-UNUSED}}$) $_{\text{R-GIVEN}}$

To annotate the information status labels and to be able to inspect them with respect to their syntactic position, the recognition of the phrases and their embedding is thus essential.

Since we are interested in the relations between the information status of a discourse item, its prosodic realisation and the syntactic structure of the sentence it appears in, the task is to enhance corpus data with the respective

[37] All examples in this paragraph are based on examples from Baumann and Riester (2012); Riester and Baumann (2017).

[38] Cf. (Riester and Baumann 2013: p. 227) for a comment on prepositional phrases.

annotation layers. For this example task, we focus on the annotation of information status according to the RefLex scheme and based on syntactic constituents.

Table 6.15 shows an overview of the coarse grained information status categories, which are part of the RefLex annotation scheme. Next to the basic distinction between referentially given (R-GIVEN) and new discourse items (R-NEW), there is an R-UNUSED category for definite discourse-new expressions, R-BRIDGING for discourse items which can be inferred from the context (such as the referee when the discourse topic is a football match), and a category for generic expressions. With respect to the lexical information status, there is the additional category L-ACCESSIBLE, which states that the lexical unit is at least related to another one which appeared in the context before, cf. Examples (6.30) and (6.32).

Referential information status		Lexical information status	
Units: referring expressions (NP/DP, PP)		Units: nouns, verbs, adjectives, adverbs	
Label	Description	Label	Description
R-GIVEN	coreferential anaphor	L-GIVEN	word identity / synonym / hypernym / holonym / superset
R-BRIDGING	non-coreferential context-dependent expression	L-ACCESSIBLE	hyponym / meronym / subset / co-hyponym / related
R-UNUSED	definite discourse-new expression	L-NEW	unrelated expression (within current news item)
R-NEW	specific indefinite		
R-GENERIC	generic definite or indefinite		
OTHER	e.g. cataphors		

Table 6.15: Overview of basic RefLex labels, based on table from Riester and Baumann (2013).

For most of these categories there are subcategories available, such as R-UNUSED-KNOWN vs. R-UNUSED-UNKNOWN to mark if a definite discourse-new expression is considered to be known to the hearer (in the sense of world

knowledge like names of major cities or active politicians) or unknown respectively. The subcategory R-BRIDGING-CONTAINED includes bridging anaphors whose anchor is a syntactic argument of the head noun. For a full reference see (Baumann and Riester 2012).[39]

6.2.1 Resources used in the task and the case studies

In the following we briefly present the resources which are involved in the combination case studies of Task II.

DIRNDL. Task II is based on DIRNDL, the Discourse Information Radio News Database for Linguistic analysis. The data set contains German radio news from March 2007. These news were broadcast hourly and the primary data consists of an audio file and a manuscript for each of the broadcasts. Based on the audio files, the broadcasts have been automatically annotated for phoneme, syllable and word boundaries (Rapp 1995) and manually annotated for pitch accents and prosodic boundaries (GToBI(S), Mayer 1995). The written manuscripts were parsed with an LFG grammar by Rohrer and Forst (2006) and the XLE system (Crouch et al. 2011). The constituency trees with the highest rank were manually annotated according to the RefLex scheme by two trained annotators. Their results were subsequently compared and homogenised.

This process showed that only 84% of the phrases which should receive an information status label according to the RefLex scheme had been recognized in the highest ranked parses although the syntactic annotation was based on the written manuscripts and the analysis did not suffer from artefacts of spoken data, such as slips of the tongue or repetitions.

To be able to use both annotation layers, i.e. the syntactic information and the information status labels, each RefLex label was either attached to the correctly

[39] We refer to the respective state of the RefLex scheme, because it relates to the dataset applied in the case studies. Furthermore Table 6.15 relates L-NEW to the span of a news item, which is also a reference to the dataset. The table from Riester and Baumann (2013) relates L-NEW to a span of five preceding clauses. However, the scheme has been further developed and the full annotation guidelines have been published (Riester and Baumann 2017).

recognized phrase if available, or to all recognized subphrases which constitute the target phrase.

Additionally, the word segmentation layer resulting from the processing of the audio files is not identical to the tokenized manuscript, based on linguistic reasons as well as technical reasons: (i) the spoken version of the news contains slips of the tongue and other speech-related phenomena, which are not reflected in the manuscripts, (ii) manuscript and audio are sometimes slightly deviating in content such as in the application of a different wording or in omitting news items and (iii) tokenization conventions differ between speech and text processing, such as omission of non-spoken punctuation and explicit transcription of spoken punctuation[40] (Eckart et al. 2012; Noha 2016).

Additionally, most of the weather forecasts the broadcasts usually end with have not been taken into account for the information status annotation since they make use of domain specific terminology and sentence structure, e.g. verbs are systematically omitted.

To be able to jointly query the annotation layers of intonation, syntax and information status the flexible linking mechanism of the B3DB has been applied (Eckart et al. 2012).[41]

Parsers. This paragraph lists the parsers employed for Task II and includes the step of parser output inspection as described in Section 5.2.1 and by means of Example (6.33)[42]. For a general introduction to the parsers, their processing pipelines, and their output, see Section 2.1.4.

(6.33) *Kevin Kuranyi schoß in Prag beide Tore für die deutsche*
Kevin Kuranyi kicked in Prague both goals for the German
Elf.
eleven
'Kevin Kuranyi scored in Prague both goals for the German football team.'
DIRNDL, sentence 26

[40] Text tokenization, one token: 2,5 vs. speech segmentation, three tokens: 2|Komma|5.

[41] Noha (2016) provides a semi-automatic mapping tool and a fine-grained mapping scheme for DIRNDL. The dataset as applied in this study refers to an older version of the mapping.

[42] This example sentence inspired the second part of Example (2.10), Page 70.

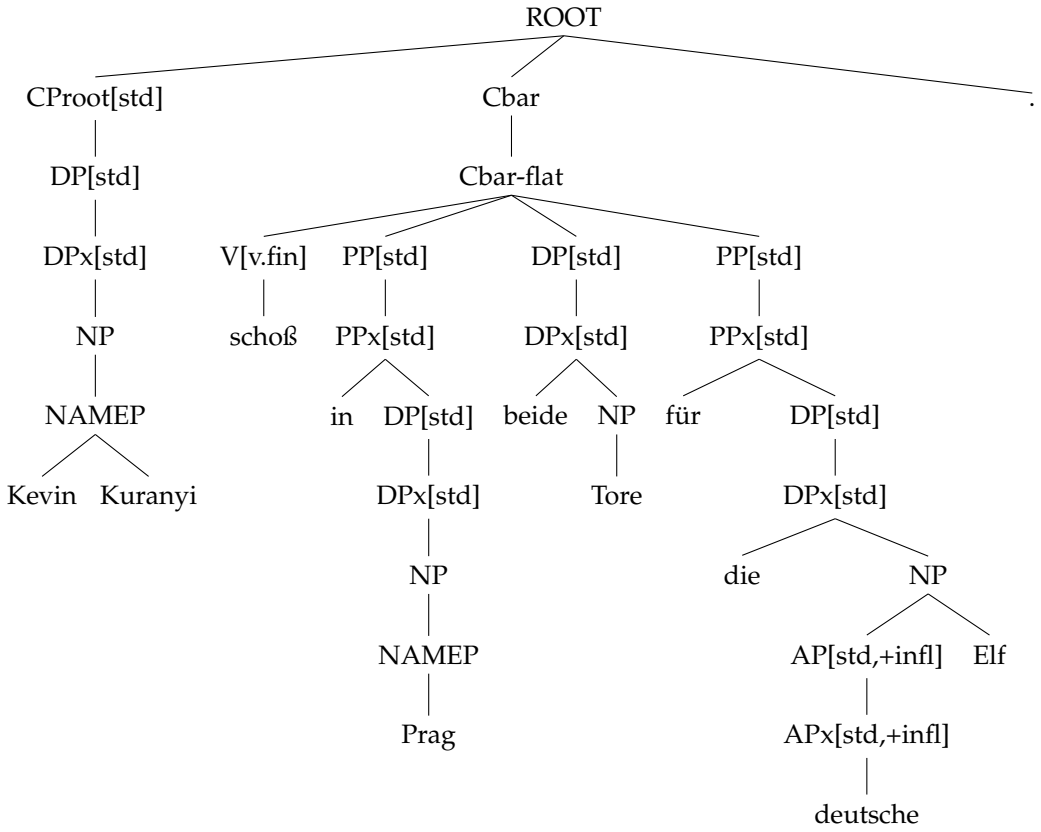


Figure 6.9: Highest ranked constituency tree from LFG parser.

The LFG parser: The LFG parser refers to the LFG grammar by Rohrer and Forst (2006) applied with the XLE system (Crouch et al. 2011). For the combination studies we take the constituency trees with the highest rank into account, which are part of the original DIRNDL annotations. They provide deeply nested structures and show a high frequency of unary chains, e.g. DP[std] → DPx[std] → NP → NAMEP in Figure 6.9.[43]

[43]Since for DIRNDL the original outputs of the LFG parser have been converted into the TIGER-XML representation format to be utilized in a manual annotation tool for further annotation, the father node of each terminal node is encoded in a part-of-speech layer and is not shown as an additional node in the tree visualization of the annotation tool. To be consistent with the input for the annotators, in this section the respective visualisations of the LFG parser output also adhere to this convention and differ in this respect from the examples in Section 2.1.4.

token	frequency
mehr_als	32
unter_anderem	32
Sri_Lanka	15
vor_allem	13
bis_zu	11
New_York	8
Die_Welt	7
im_Laufe	5
New_Yorker	3
mit_Ausnahme	3
von_Seiten	3
ohne_dass	2
Bad_Arolsen	2
British_Airways	2
Sierra_Leone	2
so_dass	2
Washington_Post	1
Gorch_Fock	1

Table 6.16: Complex tokens in LFG parser analysis of the complete written data set from DIRNDL.

The parsing pipeline takes sentence segmented data as input, but realizes its own tokenization. There are two artefacts in the token layer of the LFG parser analyses which distinguish it from the basic whitespace tokenization expected by the other parsers applied in the case studies here, BitPar and IMS-SZEGED-CIS parser. Firstly, a defined set of whitespace separated segments is merged to combined tokens by an underscore. Table 6.16 shows the resulting complex tokens appearing in DIRNDL. Secondly, the grammar is punctuation-sensitive (Forst and Kaplan 2006), i.e. punctuation symbols are treated like all other tokens and embedded into the parse tree. Since some of those punctuation marks are not written, when they are followed by another punctuation mark in the sentence, the tokenizer inserts operational commas (_,) in those places, where e.g. a

hyphen or a comma is left out. These operational commas are marked by the part-of-speech tag HAP-COMMA, referring to haplology (Forst and Kaplan 2006) where from two consecutive punctuation marks one is left out.

BitPar: BitPar is a parser for probabilistic context-free grammars (Schmid 2004), with a grammar extracted from the TIGER treebank (Brants et al. 2004). The TIGER treebank applies a flat structure, thus the constituency trees which are output by BitPar are also rather flat, cf. Figure 6.10. BitPar takes tokenized input, thus the tokenization was derived from the LFG parser analysis of DIRNDL by splitting the complex tokens shown in Table 6.16 and by omitting the operational commas. From BitPar we take the four best ranked outputs for each sentence into account.

The IMS-SZEGED-CIS parser: The IMS-SZEGED-CIS parser is a data-driven parsing pipeline from the shared task on parsing morphologically rich languages in 2013 (Seddah et al. 2013). It is based on products of context free grammars with latent annotations and discriminative reranking (Björkelund et al. 2013b). The constituents are also based on the TIGER treebank, thus the parser also produces flat structures, cf. Figure 6.11. However the tags do not include the functional part as in BitPar outputs, and from the simple inspection, the produced structures tend to be slightly flatter than those of BitPar. See also the discussion in Section 2.1.4. The tokenization is the same as for BitPar.

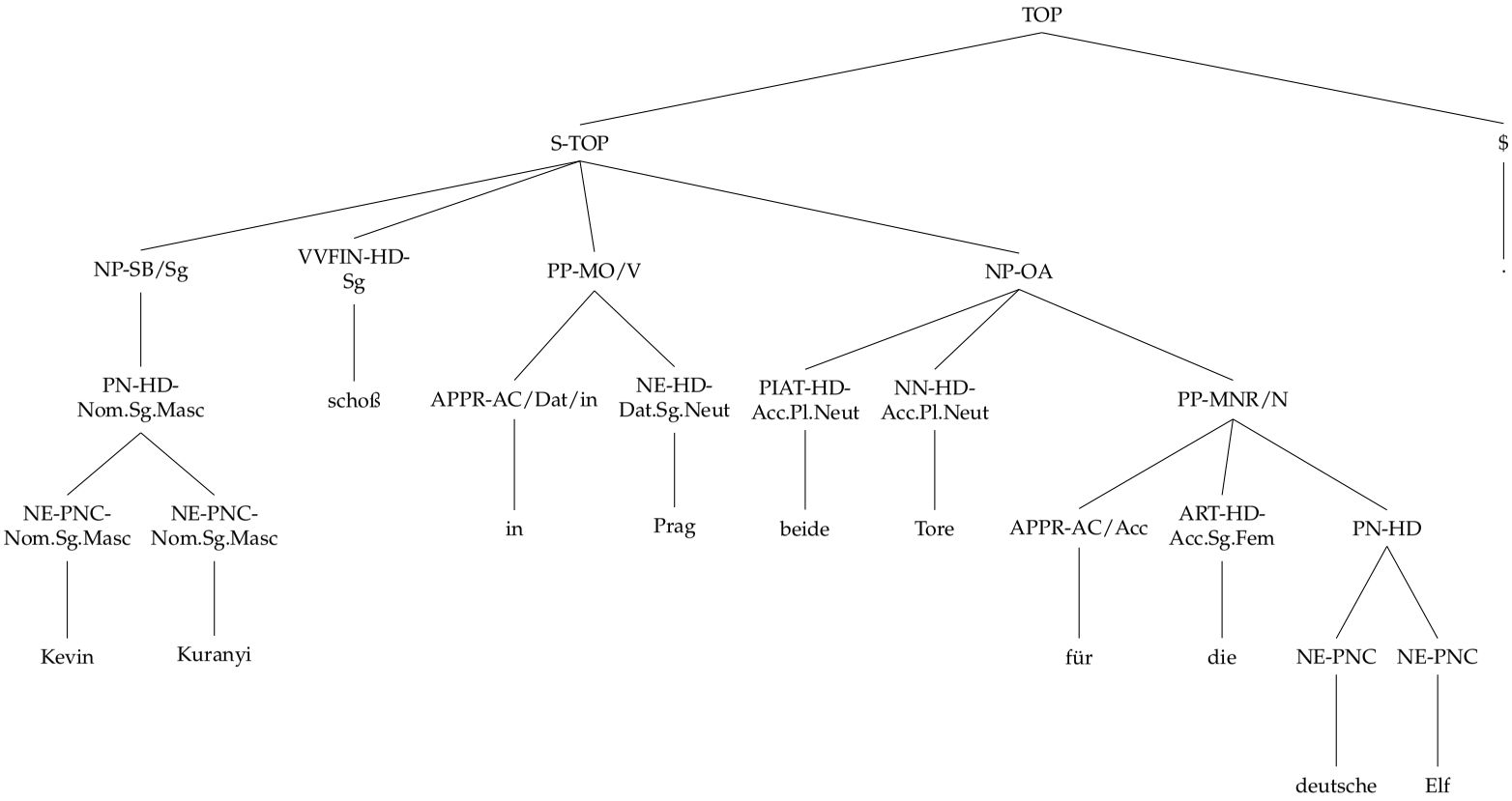


Figure 6.10: Example analysis from BitPar.

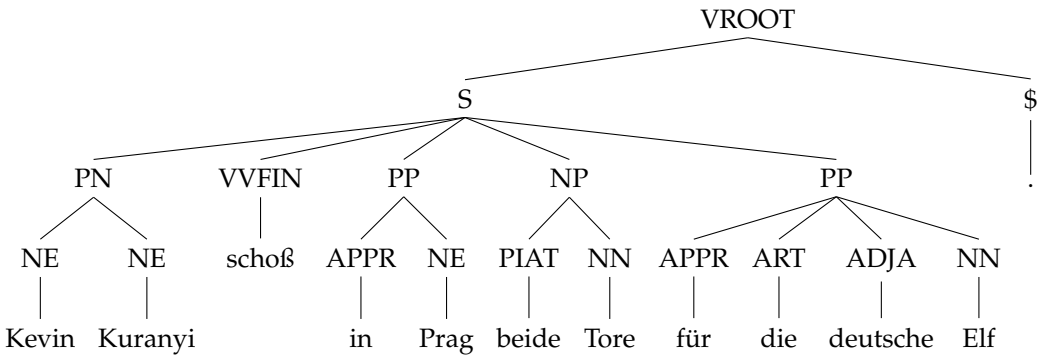


Figure 6.11: Example analysis from the IMS-SZEGED-CIS parser.

Extracted gold standard for annotated phrases. In this case, a full gold standard for the task of recognizing relevant phrases for the information status annotation comes directly with the resource, so no explicit workflow step of creating a task-related manual annotation set as described in Section 5.2.2 is needed.

In the step of manually annotating RefLex labels, the annotators were allowed to link a label to several tokens and phrases, whenever the correct syntactic phrase to be annotated had not been found by the parser. Figure 6.12[44] shows a case where an incorrectly split phrase from Example (6.34) is repaired by an R-BRIDGING label. The correct syntactic analysis would have attached the PP *in Peking* ('in Beijing') as an argument of *Regierung* ('government').

(6.34) *Er wird von der Regierung in Peking unterstützt.*
he is by the government in Beijing supported
'He is supported by the government in Beijing.'
DIRNDL, sentence 307

Extracting information about the token spans which were annotated with an information status label results in exactly the set of phrases which should have been available from the syntactic analysis, i.e. a gold standard for the task.[45]

[44]Screenshot from the utilized SALTO annotation tool:
`http://hdl.handle.net/11858/00-246C-0000-0005-BD14-0`.

[45]This gold standard is based on the information status annotation from DIRNDL 1.4.5.

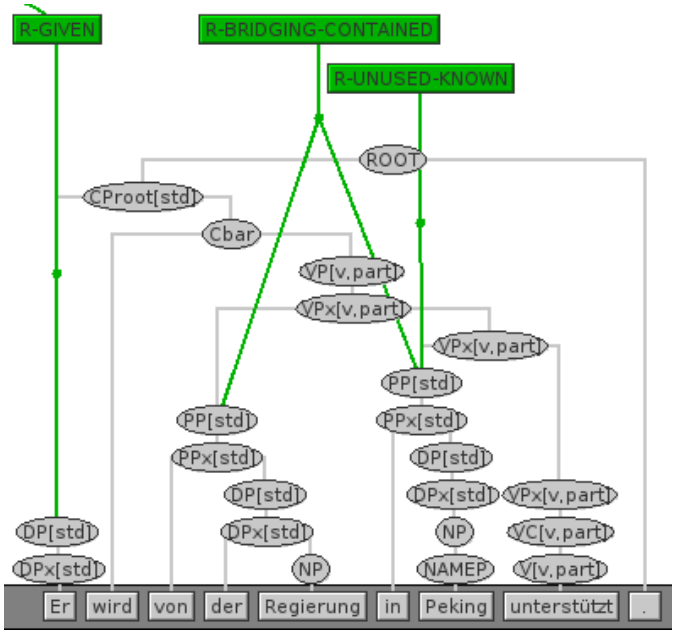


Figure 6.12: Information status annotation over incorrectly identified syntactic phrases.

For the combination case studies on Task II, this gold standard is split into two parts, a development set and a test set, and applied for two purposes: The development set serves as the task-related manual annotation set (Section 5.2.2) to evaluate the single systems and in this case also to find the best combination schemes. The test set serves as the gold standard to evaluate the results of the task against. This twofold application is possible, because the relevant syntactic features here, i.e. the phrase boundaries, coincide with the evaluable part of the actual task.

From the 3221 sentences of the written part of DIRNDL, about one-fifth (645 sentences) were designated as the test set. Since the news were broadcast hourly, many of the news features are repeated with the same or similar sentences from the previous hour. To keep the number of unseen words realistic, the test set does not contain a random selection, which would likely include many sentences appearing also in the development set, but consists of the continuous last part of the dataset (sentences 2577 to 3221). As mentioned above, most of

the weather forecast parts of the broadcast have been left out in the information status annotation, which then leaves 554 annotated sentences in the test set and 2201 annotated sentences in the development set.

The gold standard only takes phrases annotated by R-labels into account. Most of the L-labels refer to single tokens, which are either trivially accessible or subject to tokenization differences, however the latter shall not be evaluated here. For the same reason, the gold standard excludes all single token phrases with R-labels with the exception of those which are marked as complex tokens by the underscore, cf. Table 6.16.

This results in 1955 phrases in the test set and 8072 phrases in the development set.

Another aspect of the tokenization in the LFG parser output is the HAP-COMMA, or operational comma, which is inserted as an additional token (_,) where a sentence internal punctuation mark does not appear due to a following punctuation mark, cf. the inspection of the LFG parser above and Figure 2.24 (Page 96). Since most systems do not produce such additional tokens, these tokens are ignored when extracting the phrase boundaries and gold phrase boundaries from the LFG parser output.

We identified the phrases which where annotated with information status labels by listing the first and last token for each span, annotated by an R-label. This set constitutes the task-based gold standard.[46]

6.2.2 Instantiated workflow and baselines

Evaluation of the phrase recognition. For the workflow step of task-related parser evaluation, cf. Section 5.2.3, we evaluate the recall of the individual systems on the development part of the extracted gold standard for annotated phrases, which contains 8072 phrases. Table 6.17 shows the results of this evaluation of the single systems.

The deep structure of the LFG parser analyses returns the most phrases. BitPar outputs from rank 2 and 3 perform similarly, while outputs with the

[46]For the combination step, differing information about first and last tokens due to different tokenizations in BitPar, the IMS-SZEGED-CIS parser and the LFG parser was mapped to provide for a common scheme for evaluation.

system		recognized phrases	
		recall	absolute
LFG		84.01	6781
BitPar	rank 1	82.52	6661
	rank 2	82.04	6622
	rank 3	82.02	6621
	rank 4	82.38	6650
IMS-SZEGED-CIS		82.23	6638

Table 6.17: Task-based evaluation of single systems.

lower rank 4 seem to be better suited for the task than 2 and 3. Still BitPar rank 1 performs best of the BitPar outputs, while IMS-SZEGED-CIS parser is between BitPar rank 2 and BitPar rank 4.

It has to be noticed that the LFG parser parser is slightly biased with respect to the task-based evaluation set, since the information status annotation was created based on the LFG parser analyses, and the syntactic information was only overridden where necessary.

Baselines. For two reasons, the LFG parser output is chosen as the baseline: (i) it is the system originally applied for the task, i.e. if the performance cannot be improved by the combination, the extrinsic evaluation has to fail; (ii) the combination approach claims to be better than the best single system, which, according to the task-based parser evaluation is the LFG parser, even including a slight bias as explained in the preceding paragraph.

Role of the B3DB database. For Task II, all of the annotations are stored in the B3DB database, described in Chapter 4. Additionally, the combination schemes and evaluation routines are implemented by means of PostgreSQL-internal functions and tables.

A specific schema `syn` contains the tables and functions for the combination case studies of Task II. The set of gold phrases described in Section 6.2.1 and

the extracted and normalized phrases from the parser outputs are stored in two tables of this schema:

- `syn.eval_gold_phrases`
- `syn.candidate_phrases`

Both tables are directly accessed by the functions implementing the combination algorithm and share a common structure of attributes. However, to prevent that the gold phrases are accidentally mixed with the candidate phrases, the sets are kept in two tables. The tables contain the following information:

- database internal identifiers for the annotation graph which contains the token sequence (attribute `graph_id`), for the node which denotes the first token of the phrase (attribute `first_node_id`) and for the node which denotes the last token of the phrase (attribute `last_node_id`),
- the sentence number on the basis of which the different outputs can be combined (attribute `s_pos`),
- the token number in the sentence of the first and the last token of the phrase (attributes `first_node_n_seq` and `last_node_n_seq`) respectively; however these are the normalized token positions as described above, such that the phrases from the different outputs can be compared by these token numbers and also evaluated with respect to the gold standard;
- the word forms of each token in the phrase (attribute `tokens`),
- an unordered set of labels which are annotated to the phrase (attribute `labels_unordered`), i.e. the categories annotated to the non-terminal nodes in the constituency outputs and the RefLex R-labels for the gold phrases,
- a specification of the set type a phrase belongs to (attribute `set_type`), to distinguish development set from test set
- and an output name which distinguishes the different outputs for the same sentence (attribute `output_name`); while for the candidate phrases the output name refers to the system which generated the respective output, the gold phrases are marked by an output name specific to this gold data set (default: *gold*).

The combination algorithm for Case studies II.i (Section 6.2.3) and II.ii (Section 6.2.4) is also implemented database-internally:

- Function `syn.phrase_voting` returns the phrases with a majority of votes.
- Function `syn.find_phrase_voting_scheme` applies `syn.phrase_voting` to several combination settings and also includes an evaluation step. It can be utilized to find the most promising outputs for a combination on a development set and to order the combination settings in the evaluation.

6.2.3 Case study II.i: combination types 1 and 2, majority vote

Combination scheme. This case study involves output from the LFG parser, BitPar and the IMS-SZEGED-CIS parser. The combination scheme relates to the approach of *constituent voting* by Henderson and Brill (1999). Their approach is based on a majority vote over constituents, and all constituents which are voted for by the majority of the participating systems are added to the result. They apply an odd number of parser outputs, such that each vote produces a majority decision. Additionally they prove the following lemma:[47]

Lemma 6.1 (No Crossing Brackets): *If the number of votes required by constituent voting is greater than half of the parsers under consideration the resulting structure has no crossing constituents.*

quoted literally from (Henderson and Brill 1999: 188)

We also apply a majority voting over all phrases from any of the participating parser outputs. That way, we comply with the above lemma such that for each pair of returned phrases, they are either embedded or completely disjoint. Moreover, although we are only interested in the phrases which are needed in information status annotation, we guarantee that these phrases can still be fit into a constituent tree, e.g. for joint inspection of information status and syntax annotations.

[47] They apply a proof by contradiction, cf. Henderson and Brill (1999: 188).

To find the best combination scheme of outputs we test the combination sets on the task-related manual annotation data. We test all combinations by assigning either a vote or no vote to an output and take only those cases into account which allow for a majority in any case, i.e. which contain an uneven number of votes.

Since some of the combinations are only among BitPar outputs, these combinations are of combination type 1 (cf. Section 3.3): the outputs utilize the same structure, i.e. rather flat constituent trees and the same concepts in their tagsets, as they are all produced by the same parser. The combinations which include (also) the IMS-SZEGED-CIS parser and the LFG parser are of combination type 2 (cf. Section 3.3). The IMS-SZEGED-CIS parser is very similar to BitPar and might still qualify for a type 1 combination with BitPar, however the depth of the structures differs slightly and regarding the applied tagsets, even if both are based on the TIGER treebank, the tags from the IMS-SZEGED-CIS parser do not explicitly represent the functional information. A combination of BitPar, the IMS-SZEGED-CIS parser and the LFG parser is clearly of type 2 since all parsers apply constituency trees, but the LFG parser applies a very different set of labels for the non-terminal nodes.

Table 6.18 shows the results from all tested combinations on the task-based manual annotation set. Additionally, we provide the information where the combination identifies more correct phrases than the best participating system (underlined scores in Table 6.18) and we indicate an oracle setting. For the oracle, each correct phrase appearing in at least one of the outputs was returned.

Only in 13 of the 26 combination cases, the combination identifies more correct phrases than the best participating system. Three combinations exceed the baseline, with the best combination consisting of the output from the LFG parser, the IMS-SZEGED-CIS parser and BitPar rank 4. We thus decide on this setting for the combination scheme.

It is interesting to see that the four best performing combination schemes include the LFG parser, the IMS-SZEGED-CIS parser and exactly one of the BitPar outputs. This finding supports the assumption of combination approaches, that the more the combined systems differ, the higher the possibility to gain by combination. Adding further outputs from BitPar does not increase the

results, presumably because the error distribution is still similar across ranks. The upper bound presented by the oracle setting shows that the recall on the task-based manual annotation set could rise to 97.26%.

LFG	IMS-SZEGED-CIS	BitPar rank				recall
		1	2	3	4	
0	0	1	0	1	1	82.48
0	0	1	1	1	0	82.61
0	0	0	1	1	1	82.61
0	0	1	1	0	1	82.68
1	0	1	0	1	0	82.69
1	0	1	1	1	1	83.13
1	0	1	1	0	0	83.14
1	0	0	1	1	0	83.15
0	1	1	1	0	0	83.18
0	1	0	1	1	0	83.23
0	1	1	0	1	0	83.23
1	0	0	1	0	1	83.33
0	1	0	0	1	1	83.34
1	0	0	0	1	1	83.35
0	1	0	1	0	1	83.36
0	1	1	0	0	1	83.41
1	0	1	0	0	1	83.41
0	1	1	1	1	1	83.46
1	1	1	0	1	1	83.52
1	1	1	1	1	0	83.67
1	1	0	1	1	1	83.71
1	1	1	1	0	1	83.76
1	1	0	0	1	0	83.93
1	1	0	1	0	0	84.13
1	1	1	0	0	0	84.17
1	1	0	0	0	1	84.22
Baseline: LFG						84.01
Oracle						97.26

Table 6.18: Possible combinations for a majority vote (1: output participates in combination, 0: output does not participate), ordered by their performance on the task-based manual annotation set.

Task-based evaluation of the combination approach. The chosen combination scheme is evaluated on the 1955 phrase test part of the extracted gold standard for annotated phrases. Table 6.19 shows that combining the outputs from the LFG parser, the IMS-SZEGED-CIS parser and BitPar rank 4 returns more phrases on the test set than the LFG parser output alone, which is the best involved single system and taken as baseline.

system	recognized phrases	
	recall	absolute
BitPar rank 4	83.63	1635
IMS-SZEGED-CIS	85.47	1671
Baseline: LFG	85.73	1676
LFG + IMS-SZEGED-CIS + BitPar rank 4	87.37	1708
Oracle	98.11	1918

Table 6.19: Evaluation of the combination scheme.

For the task of information status annotation this means that less phrases have to be syntactically corrected by attaching labels to split phrases. Since the RefLex scheme is complex, its annotation is more efficient the less the annotators are absorbed by a second task. Furthermore, evaluation of the RefLex annotation is more straightforward if syntactic aspects do not have to be taken into account. However, given the oracle setting there is still room for improvement of the combinations.

6.2.4 Case study II.ii: combination type 2, three parsers, weighted voting

This case study involves output from the LFG parser, BitPar and the IMS-SZEGED-CIS parser. Since only cases where all three systems are included are taken into account, this is a case of combination type 2 (cf. Section 3.3): all parsers are similar with respect to their high-level structure (all three parsers generate constituent trees), but the LFG parser clearly differs with respect to the concepts it uses, i.e. the sets of labels annotated to the constituents.

Combination scheme. For this combination scheme, each output o_i (i in $1..n$) receives a specific number of votes w_{o_i}, called the weight of output o_i. In our case, we allow the weight of a system to be between 0, i.e. information from the output will definitely not influence the overall decision, and n, i.e. the weight is equal to the number of participating systems.

In weighted voting, the *quota* is the number of votes which a proposal needs to pass. A participant which has a weight w_j greater than or equal to the quota q is called a *dictator*, because the number of votes of this participant alone allows a proposal to pass. A participant is said to have a *veto*, if this participant is not a dictator but can prevent a proposal from passing, i.e. if the the weight of this participant is smaller than the quota, but the sum of the weight of all other participants is smaller than the quota as well.[48]

To fulfil the tree constraint by means of the lemma of *No crossing Brackets*, cf. Section 6.2.3, the quota to decide if a phrase is part of the output will still need the majority of votes, i.e. the quota q is defined as

$$q = \lfloor \frac{\sum_{i=1}^{n} w_{o_i}}{2} \rfloor + 1 \tag{6.35}$$

Note that with this quota, no participant can have a veto: if the sum of the weights of the remaining systems does not meet the quota, the weight of the respective participant must be equal to or greater than the quota and the participant is thus a dictator by definition.

As in Case study II.i, we intend to find the best combination scheme by means of the development set. For this case study there are two factors involved: the participating outputs and the weight vector. Note that, depending on the weight distribution, it is possible that an output will never play a role in the decision, even if its weight is greater than 0.[49]

To find the best combination scheme of outputs we test the combination sets on the task-related manual annotation data, i.e. the development set from the extracted gold standard for annotated phrases. We test all combinations by

[48] See e.g. Lippman (2013) for an introduction or a terminology overview at: `http://www.ctl.ua.edu/math103/POWER/WtVtTerm.htm`

[49] For example in the case of four outputs ($n = 4$) and weight vector $(3, 3, 1, 4)$, the quota is 6. o_3 with weight 1 will never be able to change the outcome in any combination.

assigning weights to the outputs and take only those cases into account which allow for a majority in any case, i.e. which contain an uneven number of votes. Since we apply the same data set, baseline and oracle are the same as in Case study II.i. Due to the high number of combination cases, Table 6.20 only shows the three best results, each being the result of several weight vectors. All of the shown combinations are better than the baseline and thus the best included single system.

Since the last nine weighted combinations perform equally, we need to choose one combination from this set. Inspecting the weights shows, that the flat structures from the IMS-SZEGED-CIS parser are mostly taken highly into account: in six of the nine cases, its weight is the maximum of 6 votes. In seven of the nine cases, the BitPar ranks build two groups by their weight: rank 1 gets the same weight as rank 3 and rank 2 gets the same weight as rank 4. Thereby the group of ranks 2 and 4 is weighted higher than the group of ranks 1 and 3. We thus choose a combination setting, where each system gets the weight which is most often assigned to this system in the best nine combinations: 3 votes for BitPar rank 2 and rank 4, 1 vote for BitPar rank 1 and rank 3, 6 votes for the IMS-SZEGED-CIS parser. The LFG parser gets a weight of 4 and a weight of 5 in the same number of cases, however only the weight vector (5,6,1,3,1,3) is part of the best results table, so we choose this combination for the evaluation setting.

LFG	IMS-SZEGED-CIS	BitPar rank 1	2	3	4	recall
5	6	3	2	3	2	
4	6	3	1	2	1	
4	6	3	2	3	1	
5	6	4	1	4	1	
3	4	2	1	2	1	
3	5	3	1	2	1	
4	5	2	1	2	1	
4	6	3	1	3	2	
4	6	4	1	3	1	84.37
3	6	4	1	2	1	
5	6	3	1	3	1	
4	6	1	3	1	4	
4	6	1	2	1	3	
3	5	1	2	1	3	
5	6	2	1	2	1	
3	6	1	2	1	4	
4	5	3	1	3	1	
4	5	2	3	1	2	
5	6	2	3	1	2	84.39
5	6	3	4	1	2	
5	6	2	4	1	3	
4	6	2	3	1	3	
5	6	1	4	1	4	
3	4	1	2	1	2	
4	6	1	3	2	3	
5	6	2	3	2	3	84.40
5	6	1	2	1	2	
4	5	1	3	1	3	
4	5	1	2	1	2	
5	6	1	3	1	3	
Baseline: LFG						84.01
Oracle						97.26

Table 6.20: Best weighted combinations on the development set.

Task-based evaluation of the combination approach. The chosen combination scheme is evaluated on the 1955 phrase test part of the extracted gold standard for annotated phrases. Table 6.21 shows that also here, like in Case study II.i, combining the outputs from the LFG parser, the IMS-SZEGED-CIS parser and BitPar returns more phrases on the test set than any of the single systems, including the LFG parser output, which is the best involved single system. Again, baseline and oracle values are the same for both case studies.

system (vote)	recognized phrases	
	recall	absolute
BitPar rank 4 (1)	83.63	1635
BitPar rank 2 (1)	84.14	1645
BitPar rank 3 (1)	84.60	1654
IMS-SZEGED-CIS (1)	85.47	1671
BitPar rank 1 (1)	85.63	1674
Baseline: LFG (1)	85.73	1676
LFG (5) + IMS-SZEGED-CIS (6) + BitPar rank 1 (1) + BitPar rank 2 (3) + BitPar rank 3 (1) + BitPar rank 4 (3)	87.93	1719
Oracle	98.11	1918

Table 6.21: Evaluation of the combination scheme.

6.2.5 Discussion of Task II

Table 6.22 summarizes the results from the case studies of Task II. The single systems are sorted by their performance with the LFG parser as the baseline system and the best single system.

Case study II.i took combinations of type 1, i.e. with same structure and same tagset, and type 2, i.e. with similar structure but differing tagsets, into account. However since the best combination on the development set was of type 2, the type 1 combinations were not taken into account for the evaluation on the test set. On the development set, the combination of BitPar ranks 1, 2 and 4 was the best combination among only BitPar analyses. Table 6.22 shows the performance of this combination on the test set: it is very close to the best involved single rank, and the LFG parser baseline.

Applying a combination of all parsers and combination type 2, the result clearly improves over the baseline and the best involved single system. Adding a simple weighting scheme to the type 2 combination also improves the result, but overall the benefit of the weighting is small, compared to the increased number of combinations to be tested to find the best setting on the development set and it does still not bridge the gap to the oracle setting.

As discussed before, a combination of the IMS-SZEGED-CIS parser and BitPar ranks could be seen as combination type 1 or 2. The best combination of these systems on the development set included all BitPar ranks and the IMS-SZEGED-CIS parser. To inspect the influence of differences, Table 6.22 also adds the result of this combination on the test set to the table, here as combination type 2, but in grey to mark the somehow vague status. In contrast to the BitPar only setting, adding the IMS-SZEGED-CIS parser clearly exceeds the best involved single system (BitPar rank 1) as well as the baseline, although the LFG parser wasn't even involved. This argues for the importance of adding different systems, even if they are based on similar structures and tagsets. However when adding a parser for a clear type 2 combination, i.e. the LFG parser, the result improves again, and only one rank of BitPar is needed, thus supporting the importance of differing systems even more.

system	recognized phrases	
	recall	absolute
single systems		
BitPar rank 4	83.63	1635
BitPar rank 2	84.14	1645
BitPar rank 3	84.60	1654
IMS-SZEGED-CIS	85.47	1671
BitPar rank 1	85.63	1674
Baseline: LFG	85.73	1676
combination type 1		
BitPar rank 1 + BitPar rank 2 + BitPar rank 4	85.68	1675
combination type 2		
IMS-SZEGED-CIS + BitPar rank 1 + BitPar rank 2 + BitPar rank 3 + BitPar rank 4	86.75	1696
LFG + IMS-SZEGED-CIS + BitPar rank 4	87.37	1708
LFG (5) + IMS-SZEGED-CIS (6) + BitPar rank 1 (1) + BitPar rank 2 (3) + BitPar rank 3 (1) + BitPar rank 4 (3)	87.93	1719
Oracle	98.11	1918

Table 6.22: Overview of the results from case studies. For the weighted voting, the votes are added in brackets after the system name.

6.3 Discussion of all case studies

This last section of the chapter focusses on the comparison between the case studies for Task I and the case studies for Task II. The results shown Table 6.23 repeat results from Table 6.14 and Table 6.22[50]. Of course there are fundamental differences between the two tasks. Next to the fact that they refer to different phenomena (German *nach*-particle verbs vs. information status), Task I is more precision-oriented, i.e. the correctness of the extracted features is of higher importance than finding all instances, while Task II is more recall-oriented, i.e. finding all phrases to be annotated is more important than extracting only the correct phrases.[51] Based on the task and the orientation they are also evaluated with different measures. Thus, if these two case studies are compared, it is with respect to the application of task-based parser output combination as a method.

Since the case studies are classified according to the combination types introduced in Section 3.3, we are able to assess the effect of the degree to which the combined outputs differ. Three combination results refer to case studies of combination type 2 (similar structure, different concepts), one to combination type 1 (same structure, same concepts) and one to combination type 3 (different structures, different concepts). For the case studies of combination type 2 and 3 conducted here, we see a clear improvement of the combination over the best involved single system. This is not the case for the combination of type 1. The conclusion is thus that it is of high importance for the combination approach, how much the systems differ. For Task I, the sensible choice of the parsers for a combination of type 2 resulted in a similar combination gain over the best involved single system as for the combination of type 3. However the combination of type 3 still sightly improves the overall value.

The second dimension regarding the combination approaches is – next to the choice of the parsers and thus the combination type – the choice of the com-

[50] For Task II, the combination of the IMS-SZEGED-CIS parser with BitPar was excluded to focus on clear type 2 studies.

[51] Nevertheless, a certain number of instances is needed for the clustering in Task I and there is a restriction for Task II such that the extracted phrases still have to fit into a tree structure. Thus, the tasks are still reasonably balanced; trivial approaches, such as extracting only very few instances for Task I or marking every span as a phrase for Task II cannot be applied.

bination scheme. Case study I.i applies a rule-based combination, Case study I.ii applies a majority vote with a rule-based fallback, Case study II.i applies a simple majority vote and Case study II.ii applies weighted voting. We observe that the more sophisticated approaches (in our case, rule-based combination and weighted voting) do not perform decisively differently from their basic majority voting counterparts, which is in line with the findings of Surdeanu and Manning (2010) regarding weighting and additional classifiers. However, it is to be noted that the rule-based combination allows for a combination in which no majority setting is necessary, i.e. also a combination of only two parsers can be successfully applied. Thus the rule-based setting also played a role as a fallback for the majority scheme in Case study I.ii.

The last aspect relates to the comparison of the precision-oriented task with the recall-oriented task. Based on the two example tasks applied here, the method of task-based parser output combination can be successfully applied in both cases. Even with the majority settings, which might intuitively support a selection rather than a broader extraction of cases, the combination approach is also successful in the recall-oriented task.

It should be noted that the conclusions drawn here are based only on two tasks and their respective case studies and are thus restricted to these settings. However, due to the task-based focus of the method and of this work as a whole, it was in line with the approach to select two very different tasks for an (external) evaluation by means of case studies. While the tasks were chosen independently, there is however a similar outcome with respect to the gain of combination as such and the importance of differing systems taking part in the combination. The latter also supports the classification of combination types introduced in Section 3.3. Thus, the task-based evaluation conducted here supports the usefulness of task-based parser output combination as a valid method to increase the reliability of syntactic features for further processing tasks.

	features (added up)	V-measure		
		mate	FSPar	BitPar
single systems	dat,acc	30.39	31.61	33.20
	+ form	31.68	31.37	31.37
Case study I.i				
combination type 2, rule-based	dat,acc	34.36		
	+ form	34.12		
Case study I.ii				
combination type 3, majority vote	dat,acc	35.69		
	+ form	34.67		

(a) Results of the case studies for Task I: clustering of *nach*-particle verb lemmas, according to their accusative and dative arguments

	system	recognized phrases	
		recall	absolute
single systems	BitPar rank 4	83.63	1635
	BitPar rank 2	84.14	1645
	BitPar rank 3	84.60	1654
	IMS-SZEGED-CIS	85.47	1671
	BitPar rank 1	85.63	1674
	LFG	85.73	1676
Case study II.i			
combination type 1, majority vote	BitPar rank 1 + BitPar rank 2 + BitPar rank 4	85.68	1675
combination type 2, majority vote	LFG + IMS-SZEGED-CIS + BitPar rank 4	87.37	1708
Case study II.ii			
combination type 2, weighted voting	LFG (5) + IMS-SZEGED-CIS (6) + BitPar rank 1 (1) + BitPar rank 2 (3) + BitPar rank 3 (1) + BitPar rank 4 (3)	87.93	1719

(b) Results of the case studies for Task II: recognition of phrases for information status annotation

Table 6.23: Overview of the outcome of all case studies.

Chapter 7

Conclusion

The objective of this work was to increase the reliability of automatically generated syntactic information for subsequent processing steps.

We started from two assumptions: (i) generating syntactic analyses is not an isolated task, and (ii) parsing, as an automatic processing step to retrieve syntactic information, cannot be perfect. These assumptions are motivated in Chapter 1.

Based on these assumptions, this work presents task-based parser output combination as a device to reach the objective. As main contribution Chapter 5 presents an abstract workflow for task-based parser output combination, which can be instantiated for different tasks and different parsing systems.

To support this workflow from an infrastructural point of view, Chapter 4 discusses requirements for and an implementation of the relational database B3DB. The database allows for a detailed tracking of the workflow by means of process metadata (Section 1.2.4), and is able to handle several different analyses for the same input data, independently of the linguistic framework the analyses are based on. By means of its generic data structures, which are based on a serialization of the Linguistic Annotation Framework (ISO 24612:2012), it supports representational interoperability. Splitting interoperability aspects into representational differences, structure-related differences of the content and concept-related differences of the content, as done in our refined concept proposed in Section 3.2, paves the way to a classification of the different types

of combination settings when resources are combined. Regarding parser combination, this classification reflects the trade-off between combination effort and gain due to diverging error distributions.

An essential approach for this work is to take the task into account. To show that the presented aspects indeed work together and can be applied, Chapter 6 presents case studies for two different tasks, a precision-oriented one and a recall-oriented one. Additionally, the chosen tasks are very different with respect to the studied phenomena and their evaluation. For both cases, applying a task-based combination approach with output produced by sufficiently diverging systems, i.e. a combination setting of type 2 or 3 (Section 3.3), provided a gain for the task, cf. the discussion in Section 6.3.

Nevertheless, two example tasks, even if they differ in their setup and orientation, do not as such cover all issues regarding applicability. Thus, we should answer the question of **applicability** on a more abstract level.

A **prototypical setting** for the application of task-based parser output combination is one where the following factors apply:

- The syntactic annotation is an auxiliary device for the task, which means
  (a) the focus is on a task which is different from parsing and
  (b) the task profits from a specific set of syntactic features (cf. Section 1.2.7 for the use (and meaning) of the term 'feature' in this work).
- The set of syntactic features which are needed for the further processing steps is known in advance (so the combination can be optimized with respect to these features).
- A set of different parsers is available.

If one moves away from the prototypical setting, the applicability of task-based parser output combination in **less typical settings** has to be discussed. In cases where the task does not differ from generating a syntactic analysis or where all aspects of the syntactic analysis are needed for the task, task-based parser output combination introduces an overhead where instead classical combination approaches straightforwardly apply (cf. Section 2.2 for related work on parser combination) and are thus to be preferred.

In cases where the needed features or even the task are not known in advance, the abstract workflow for task-based parser output combination cannot be fully instantiated. However, in these cases the output from different parsers, or several n-best lists can be provided as a resource, e.g. in an annotated corpus, such as the SFB732 silver standard collection (Eckart and Gärtner 2016). These annotation layers provide analyses in a horizontal relation (cf. Section 1.2.5), i.e. several, probably differing, annotations of the same layer for the same primary data. On the basis of such data, the proposed workflow for task-based parser output combination can be easily instantiated for upcoming extraction tasks which meet the criteria of applicability stated above.

If no parsers are available or none of the available parsers present the required features in the expected granularity, it might be necessary to build or train a parser explicitly for the task. In the latter case, when the available parsers do not represent the needed features, e.g. a distinction between animate and inanimate direct objects, an alternative is to apply task-based parser output combination with respect to more coarse-grained features, e.g. direct objects, and add an additional annotation or extraction step afterwards, e.g. manually annotate animacy or extract cases based on syntactic markers where possible, such as in Spanish differential object marking[1].

For many tasks however, no special parsers or adaptation of existing ones is required, but the task-relevant features can be extracted from off-the-shelf parsers and can be combined within the presented workflow, especially due to its focus on the output. So as an advantage of task-based parser output combination, off-the-shelf parsers are perfectly applicable in the workflow.

A further aspect regarding the applicability of task-based parser output combination is the instantiation of the abstract workflow. The workflow is set up in a generic way, such that it can be applied to a variety of tasks. However, this means that it has to be instantiated for each task. Next to the fact that instantiation can be done with minimal effort as shown in the case studies, Section 5.3 discusses the reuse of parts of an instantiated workflow in cases where tasks share the same features or in cases where only the data set is extended.

[1] An additional preposition *a* ('at/to') can mark animate direct objects.

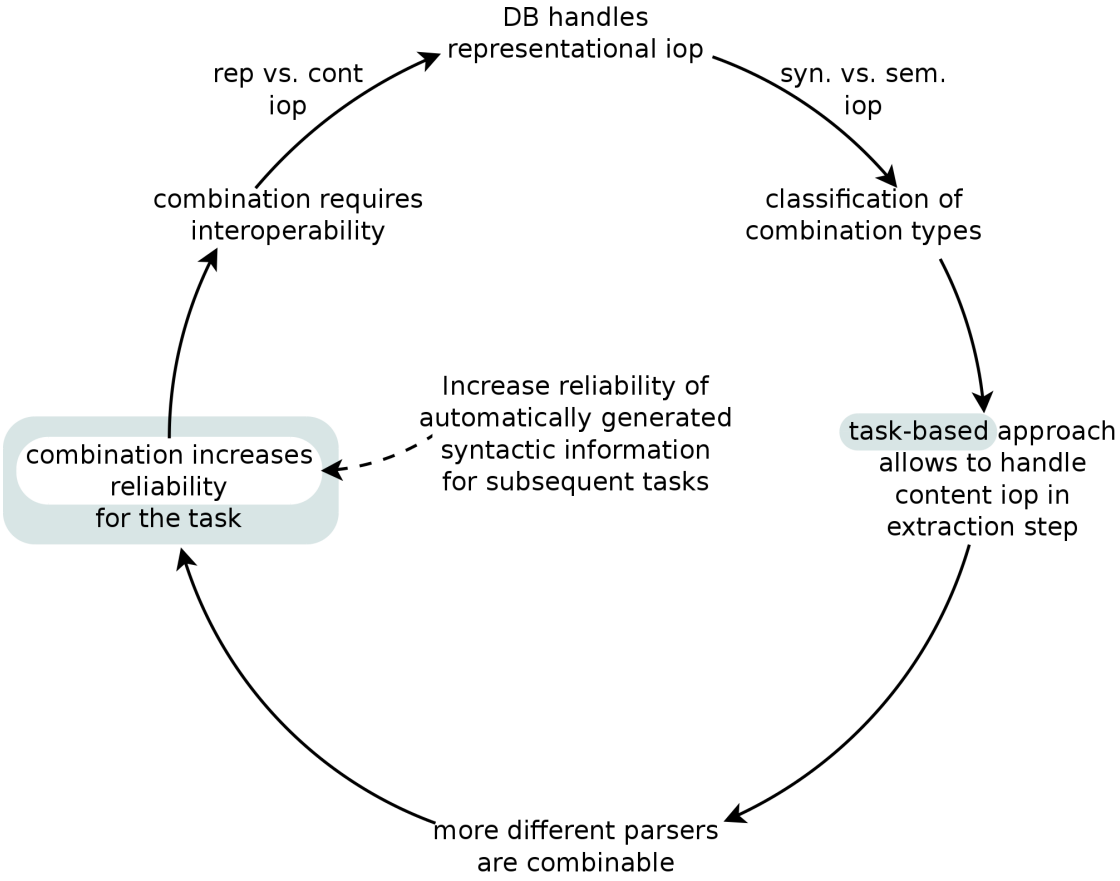


Figure 7.1: How the contributions of this work help to reach the objective.

Figure 7.1 sums up how the several aspects of this work come together to increase the reliability of automatically generated syntactic information for subsequent tasks. Starting from related work, we learned that system combination increases the reliability of an analysis step. However, combination requires interoperability, which brings us to the clearly defined and refined categories of interoperability, which we introduced to allow for a separate treatment of the categories. We implement full representational interoperability by means of the database, so that all remaining interoperability issues are actually related to the content. For these, we utilize the classification of combination types based on the separation of syntactic and semantic interoperability to denote the trade-off

between combination effort and gain and shift our focus to the task, such that content interoperability can be handled in the extraction step and has only to be accomplished for the features which are relevant for the task. By taking away the need to establish interoperability for the complete analysis, we are able to combine parsers which are more different than the ones which usually take part in a combination approach. And while the combination gain increases with the difference of the parsers, we enhance reliability, not for the whole analysis, but more specifically for the task, which was the objective of this work.

Taking up the **research question** from the introduction in Chapter 1, this summary provides an answer to the question of how to combine different automatic syntactic analyses to increase the reliability of information needed for the respective further processing steps, i.e. increase the analysis quality of those bits of information which are utilized in the subsequent processing. Chapter 1 split this question into several aspects, repeated in the following.

Q 1 referred to the fact that the usefulness of the combination depends on the different error distributions of the involved systems and thus on their dissimilarity. So the question was: How can such different systems become interoperable (i) with respect to the linguistic assumptions they are based on and (ii) with respect to the different representations of the linguistic information they apply?

The answer are the refined categories of interoperability and combination types, mapping representational interoperability to part (ii) of the question and content-related interoperability to part (i). Additionally, focussing content-related interoperability on the task-based features also prevents loosing relevant information for the combination, where an attempt to find a common set of categories would result in very coarse-grained tagsets, cf. the discussion on the project PASSAGE in Section 3.4.

Q 2 referred to the discussion, that it can be hard to determine what should be the single perfect analysis of an input, based on factors such as ambiguities requiring world knowledge for them to be resolved, register, creation time or

even the underlying syntactic theory. The question was: If perfectness depends on so many context variables, how can the combination approach be evaluated?

The answer to this question takes the task-based approach into account and triggers an extrinsic evaluation, which fixes the context variables with respect to the task.

Q 3 then raised the important question of genericness and applicability of combination: Which steps are necessary to find a combination approach which benefits from synergetic effects of the participating systems, and can these steps be defined in an abstract way?

The answer and relating thereto a main contribution of this work, is the abstract workflow for task-based parser output combination, presented in Chapter 5, which provides guidelines to instantiate a combination approach for different tasks.

Q 4 finally added an infrastructural aspect by asking: What are the requirements an infrastructure needs to fulfil in order to support and document complex procedures, as they are targeted by a combination approach?

The answer is given in Chapter 4 by means of the requirements of the supporting infrastructure applied in this work: the relational database B3DB, which is able to support and document the procedures for the combination approach.

Shifting the focus to a broader angle still, this work contributes to several **research debates**. Firstly, it is a contribution to the topic of **interoperability and sustainability**. Interoperability is often subject to terminological variation, either due to implicit definitions, or due to several explicit definitions as presented in Section 3.1. While adding an explicit definition, this work contributes systematic criteria for refined categories of interoperability, to which many of the discussed definitions can be allocated. Figure 7.2 summarizes this correlation.

Regarding interoperability and sustainability, this work chooses an interlingua approach for handling representational interoperability. That is, instead of providing full mappings from one resource format into another, or in the

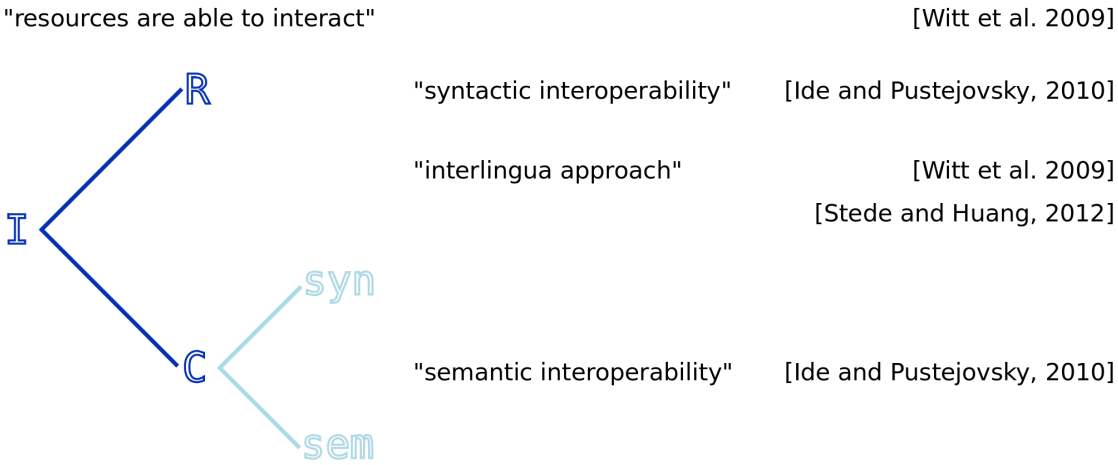


Figure 7.2: Refined classification of interoperability and the state of the art (I: Interoperability, R: representational interoperability, C: content-related interoperability, syn: interoperability of structure decisions, sem: interoperability of concept definitions).

worst case providing $n(n-1)$ mappings for n resource formats for pairwise transfer interoperability, several formats are mapped to a generic pivot format. Examples for this approach are the Linguistic Annotation Framework (ISO 24612:2012) and the Pepper Converter Framework with the meta model Salt (Zipser and Romary 2010)[2]. In applying this approach to the aspect of representational interoperability, this work supports the generic approach to interoperability and thereby sustainability, in taking away the need to handle several representations which might in the future be no longer accessible or no longer common. By choosing a relational database infrastructure, this effect is increased, since the representational interface is reduced to the knowledge of the widespread multi-purpose query language SQL.

For a last aspect of sustainability we return to the concept of process metadata from Section 1.2.4. This work advocates the transition from a simple presentation of the results to the monitoring of the whole workflow. This is evident from the main contribution, which is a workflow in itself as well as

[2]Pepper PID: `http://hdl.handle.net/11022/1007-0000-0000-8E9E-F`

from the supporting infrastructure. The infrastructure inherently allows for a flexible and detailed tracking of the workflow, i.e. of the included manual or automatic steps, the included or applied resources, e.g. tools, tool components, corpora, lexicons, as well as the respective versions in which the resources were applied. Keeping track of workflow details increases reproducibility and interpretability of the results and allows other users to decide if a resource or a part of a workflow can be reused for their purpose, thus also increasing the sustainability of existing resources. Fortunately the awareness of the need for thorough workflow tracking is increasing, as can be seen in recent projects and initiatives such as LAUDATIO[3] (Krause et al. 2014), WebLicht[4] (Hinrichs et al. 2010) and RePlay-DH[5] (Hahn et al. 2017).

The second major research debate this work contributes to is obviously **system combination**. Three aspects have been introduced in this work, (i) combination types based on content-related interoperability, (ii) rule-based combination and (iii) task-based combination as a method. The combination types reflect how different the outputs of the parsers are with respect to the underlying theoretical approach after detracting all aspects of representational interoperability. The three combination types take differences in the linguistically motivated structure and tagset into account and distinguish combination settings with most similar parsers from combinations of output with similar structure but different tagsets, and from settings where most different parsers are combined and interoperability may be comparatively hard to achieve. The case studies are classified according to the combination types and we see improvement of the results for the task along the combination types, i.e. the more different the involved systems are.

The case studies for Task I also introduce rule-based combination, which has been rarely used in the literature so far. Zeman and Žabokrtský (2005) test some settings which can be seen as rule-based, but the rules we defined for Task I are more sophisticated, especially with respect to the task. Two common arguments against rule-based approaches are that creating a set of good rules takes too

[3]LAUDATIO PID: `http://hdl.handle.net/11022/1007-0000-0000-8E65-F`

[4]`https://weblicht.sfs.uni-tuebingen.de/weblichtwiki/index.php/Main_Page`

[5]`https://www.ub.uni-stuttgart.de/replay/`

much time and that even with good rule sets, hand-crafted rules often cannot cover unexpected data. However, taking the task into account, we created a small data set for parser evaluation and created the combination rules based on this evaluation. Since evaluation and rules only have to take the task-relevant features into account, we have a fixed number of combination cases which can appear and thus a set of rules which covers all these cases. Furthermore the set of combination rules could be set up in little time, based on the recall and precision values of the evaluation. Thus, rule-based combination can be carried out efficiently when the task-relevant features restrict the possibilities of combination. In addition, rule-based combination broadens the application possibilities as it allows e.g. for a sophisticated combination of two very different parsers, or for a fall back solution in voting, cf. Section 6.1.

The most important aspect this work introduces is the task-based focus of the combination and thus task-based combination as a method. Taking the task into account does not only allow for a combination of more different systems, and thus for more gain from the combination, it also provides for the best possible outcome for the actual task. Parsing is usually just a step in a processing chain and its results are exploited in a specific task. Improving the syntactic analysis by general means of combination usually improves the overall parse, but might go at the expense of the features needed for the task. Taking the task into account broadens the focus beyond parser evaluation and supports tasks such as research questions on semantics, discourse or prosody, or furthers text- or speech-based fields in the Digital Humanities (e.g. relation extraction). As mentioned above the task-based focus also allows for a combination of less interoperable analyses because it lifts the burden of having to find a mapping which applies to all analyses from systems which are based on different theoretical frameworks. Such mappings are often impossible to find, leaving only the possibility of generalizing until the categories become very coarse and in this process loosing the important information which would have been available in one system or another. The introduced abstract workflow facilitates task-based combination as a method and can be applied to various tasks.

Lastly, this work contributes to the research debate on **evaluation**. A task-based focus of the analyses also shifts the evaluation focus from treebank reproduction to downstream or "real-live" tasks. Shared tasks on parsing (e.g. Buchholz and Marsi 2006; Hajič et al. 2009; Petrov and McDonald 2012; Seddah et al. 2013) have fostered the creation and development of many different parsing systems by letting them compete for accuracy on treebanks from several languages or domains. Now it is time to exploit the differences of the systems and the available knowledge in tasks beyond parsing as such.[6] Furthermore, with extrinsic evaluation, the objective can hardly be to find the one best single system which suits all tasks, since tasks are too different. Thus extrinsic evaluation will provide information on reliable features for a given task. And even if there are many different tasks to be handled, each of which could require its own extrinsic evaluation, it is not necessary to tailor a parser for each specific task, because there are many good systems, and their knowledge can be exploited for different tasks by means of combination.

This also gives us a good transition to an **outlook** beyond this thesis. Alternative routes to the ones chosen within this work have been discussed in several chapters and above in this conclusion regarding the applicability of the abstract workflow of task-based parser output combination. This last section thus focusses on a few selected aspects.

One path of further exploration would be concerned with more tasks, more use cases and further inspection of parser differences along the lines of the classification of combination types, e.g. based on more different training data[7]. This also includes taking further combination schemes into account, such as stacking, which allows us to play with the proportions of the preparation steps before the decision on the combination scheme.

A simple combination which applies the shortcut discussed in Section 5.3 skips the steps of creating a task-related manual annotation set and the respec-

[6]This aspect is based on a point formulated in Eckart and Gärtner (2016), moving from typical data which can be adequately handled by specific tools to handling several types non-typical data.

[7]This aspect was inspired by an anonymous reviewer.

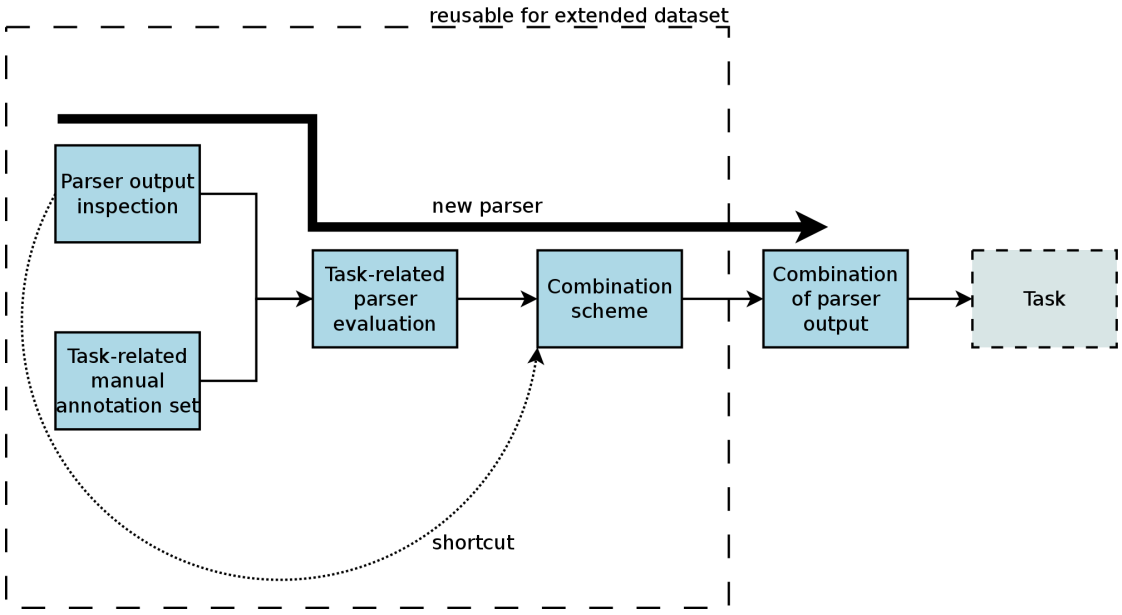


Figure 7.3: Alternative routes in the workflow.

tive task-related parser evaluation (see Figure 7.3 which repeats Figure 5.8 from Page 203). As mentioned in Section 5.3 this shortcut restricts the possible combination schemes, but a study on which tasks could still benefit from this easily instantiated setting would nevertheless give more insight into the usefulness of combinations.

Another aspect, which has also been mentioned in Section 5.3, is to make use of more systems producing underspecified analyses in the combination workflow, e.g. Eberle et al. (2008). These systems keep information about ambiguity where it cannot be resolved instead of conducting forced guessing. Taking ambiguous structures and labels into account provides additional information which can be exploited in the combination scheme. Underspecification is thus in line with the idea of task-based output combination in making use of knowledge already available within the systems while focussing on the task-relevant features omits the need to spell out all alternatives of the whole span of an analysis.

A task-based focus could also be included into approaches to the combination of combination systems (Le Roux et al. 2012; Björkelund et al. 2014a), or open up the perspective to vertical analysis relations (Section 1.2.5) taking features from different but independent annotation layers into account. In the SFB732 silver standard collection (Eckart and Gärtner 2016) vertical analysis relations can be used to find contradicting or supporting information which results in an assignment of tentative confidence. Dannenberg et al. (2016) describe a study comparing prosodic and syntactic phrase boundaries. Information from these two annotation layers could thus support precision-oriented tasks in demand of reliable phrase boundaries within a combination approach of vertically related analyses.

It would also be a possibility to tune single systems with respect to a task and apply a combination approach including these systems. While this approach requires more insight into the single systems and their configuration, it would be interesting to compare the benefit of already task-aware single systems in a task-based combination to the setting of this work.

Coming back to the horizontal analysis relations, which have been utilized in this work by means of parser output combination, another path opens up when loosening the prerequisites for the applicability of the workflow. As discussed above, in cases where the task is not known in advance, the task-based steps of the workflow cannot be instantiated. However, as soon as a specific task emerges, being able to resort to a resource which already provides several analyses of the same layer and for the same primary data is a promising concept. That is, raising the awareness of exploiting horizontal analysis relations already in resource creation could provide a fundament for various extraction tasks. This is also an objective of the SFB732 silver standard collection, where outputs from several tools for the same layer shall be provided together with the primary data. Users with specific tasks can instantiate a combination procedure based on the already available annotation layers. Additionally, a basic confidence estimation can be provided with the resource, e.g. marking parts of the analyses where all tools agree or where all tools deviate, such that either only the former cases can be taken into account (e.g. in a precision-oriented setting) or where

the latter cases might indicate difficult and therefore interesting cases for a linguistic analysis (Eckart and Gärtner 2016).

As a last path I would like to shift the focus to the infrastructural aspects, highlighting sustainability. As discussed in Section 1.2.4 and implemented with the database in Chapter 4, process metadata is applied to track the workflow. A thorough study of minimal requirements on such information to provide for several aspects of sustainability, such as reproducibility, comparability, reusability and applicability would foster exchange and discussion on the outcome of analyses and studies based on them and could pave the way towards generically applicable concepts of process metadata, their representation and use in projects from natural language processing to the Digital Humanities.

Furthermore, in Section 4.3 we discussed advantages of SQL databases in the context of the workflows of computational linguistic project work. While structures related to the Resource Description Framework (RDF)[8] are based on a similar concept as graph databases, they come with more standard descriptions regarding encoding and querying, and additional possibilities to handle aspects of semantic interoperability, e.g. by means of OWL[9] ontologies. LAF, on which the data structures of the micro layer of the B3DB are based, has already been discussed in connection with RDF (Cassidy 2010; Ide and Suderman 2012). By means of RDF a modelling of different analyses for the same primary data and several combination approaches could be made accessible in a distributed fashion. Of course, process metadata would have to be made available as well to provide for sustainability of the distributed data.

[8]Cf. the paragraph on linked data in Section 3.4 and
`https://www.w3.org/TR/rdf11-concepts/`

[9]Web Ontology Language:
OWL `https://www.w3.org/TR/owl-ref/`
OWL 2 `https://www.w3.org/TR/owl2-overview/`

Appendix A

Tagsets

A.1 The TIGER treebank

A.1.1 Part-of-speech tags in TIGER treebank

The part-of-speech tags in the TIGER treebank are based on the STTS tagset by Schiller et al. (1999), with only a few modifications. The English descriptions are taken from Smith (2003).

part-of-speech tag	description
ADJA	adjective, attributive
ADJD	adjective, adverbial or predicative
ADV	adverb
APPR	preposition; circumposition left
APPRART	preposition with article
APPO	postposition
APZR	circumposition right
ART	definite or indefinite article
CARD	cardinal number
FM	foreign language material
ITJ	interjection
KOUI	subordinate conjunction with *zu* and infinitive
KOUS	subordinate conjunction with sentence

KON	coordinate conjunction
KOKOM	comparative conjunction
NN	common noun
NE	proper noun
PDS	substituting demonstrative pronoun
PDAT	attributive demonstrative pronoun
PIS	substituting indefinite pronoun
PIAT	attributive indefinite pronoun without determiner
PPER	non-reflexive personal pronoun
PPOSS	substituting possessive pronoun
PPOSAT	attributive possessive pronoun
PRELS	substituting relative pronoun
PRELAT	attributive relative pronoun
PRF	reflexive personal pronoun
PWS	substituting interrogative pronoun
PWAT	attributive interrogative pronoun
PWAV	adverbial interrogative or relative pronoun
PROAV	pronominal adverb
PTKZU	*zu* before infinitive
PTKNEG	negative particle
PTKVZ	separable verbal particle
PTKANT	answer particle
PTKA	particle with adjective or adverb
SGML	SGML markup
SPELL	letter sequence
TRUNC	word remnant[1]
VVFIN	finite verb, full
VVIMP	imperative, full
VVINF	infinitive, full
VVIZU	infinitive with *zu*, full
VVPP	perfect participle, full

[1] Also: first part of composition.

VAFIN	finite verb, auxiliary
VAIMP	imperative, auxiliary
VAINF	infinitive, auxiliary
VAPP	perfect participle, auxiliary
VMFIN	finite verb, modal
VMINF	infinitive, modal
VMPP	perfect participle, modal
XY	non-word containing non-letter
$,	comma
$.	sentence-final punctuation mark
$(	other sentence-internal punctuation mark

A.1.2 Categories of constituents in the TIGER treebank

The list of categories of constituents from the TIGER treebank annotation is taken from Smith (2003).

constituent tag	description
AA	superlative phrase with *am*
AP	adjective phrase
AVP	adverbial phrase
CAC	coordinated adposition
CAP	coordinated adjective phrase
CAVP	coordinated adverbial phrase
CCP	coordinated complementiser
CH	chunk
CNP	coordinated noun phrase
CO	coordination
CPP	coordinated adpositional phrase
CS	coordinated sentence
CVP	coordinated verb phrase (non-finite)
CVZ	coordinated infinitive with *zu*
DL	discourse level constituent

ISU	idiosyncratic unit
MTA	multi-token adjective
NM	multi-token number
NP	noun phrase
PN	proper noun
PP	adpositional phrase[2]
QL	quasi-language
S	sentence
VP	verb phrase (non-finite)
VZ	infinitive with *zu*

A.1.3 Function labels in the TIGER treebank

The list of function labels from the TIGER treebank annotation is taken from Smith (2003).

function tag	description
AC	adpositional case marker
ADC	adjective component
AG	genitive attribute
AMS	measure argument of adjective
APP	apposition
AVC	adverbial phrase component
CC	comparative complement
CD	coordinating conjunction
CJ	conjunct
CM	comparative conjunction
CP	complementizer
CVC	collocational verb construction (*Funktionsverbgefüge*)
DA	dative
DH	discourse-level head
DM	discourse marker

[2] In the text usually referred to as prepositional phrase.

EP	expletive *es*
HD	head
JU	junctor
MNR	postnominal modifier
MO	modifier
NG	negation
NK	noun kernel element
NMC	numerical component
OA	accusative object
OA2	second accusative object
OC	clausal object
OG	genitive object
OP	prepositional object
PAR	parenthetical element
PD	predicate
PG	phrasal genitive
PH	placeholder
PM	morphological particle
PNC	proper noun component
RC	relative clause
RE	repeated element
RS	reported speech
SB	subject
SBP	passivised subject (PP)
SP	subject or predicate
SVP	separable verb prefix
UC	unit component
VO	vocative

A.2 FSPar

This list of dependency labels applied by FSPar is based on documentation provided with the tool by Michael Schiehlen.[3]

role tag	description
&	separator for secondary edges
ACMP	adjunct or complement
ADJ	adjunct
APP	apposition
COMP	complement
GL	prenominal genitive
GR	postnominal genitive
HD	head (c-structure role only)
KON	coordinating conjunction
KON1	first conjunct of a coordinating conjunction
NKP	floating quantifier
NULL	[4]
PCMP	complement of a preposition or a subordinating conjunction
PD	predicative noun
PM	PTKZU *zu* VVINF
RC	attachment of relative clause (*Relativsatzanbindung*)
RK	*rechte Satzklammer*
SPEC	specifier
SUBJ	subject
TOP	[5]
VO	vocative
NP:1	subject

[3] File: example.info.

[4] No explanation given in the documentation, probably empty slot in ambiguous relation labels.

[5] No explanation given in the documentation, utilized for attachment to virtual root node.

NP:2	genitive object
NP:4	dative object
NP:8	accusative object
NP:11	expletive as subject
NP:18	expletive as accusative object
NP:24	subcategorized reflexive pronoun in dative case
NP:28	subcategorized reflexive pronoun in accusative case
PP/auf:4	subcategorized PP with head *auf* and dative case
PP/auf:8	subcategorized PP with head *auf* and accusative case
PP/auf:c	subcategorized PP with head *auf* and dative or accusative case
PP/an:4	subcategorized PP with head *auf* and dative case
PP/an:8	subcategorized PP with head *an* and accusative case
PP/aufhin:8	subcategorized PP with head *aufhin* and accusative case
PP/aus:4	subcategorized PP with head *aus* and dative case
PP/bei:4	subcategorized PP with head *bei* and dative case
PP/durch:8	subcategorized PP with head *durch* and accusative case
PP/für:8	subcategorized PP with head *für* and accusative case
PP/gegen:8	subcategorized PP with head *gegen* and accusative case
PP/gegenüber:4	subcategorized PP with head *gegenüber* and dative case

PP/hinter:4	subcategorized PP with head *hinter* and dative case
PP/in:4	subcategorized PP with head *in* and dative case
PP/in:8	subcategorized PP with head *in* and accusative case
PP/mit:4	subcategorized PP with head *mit* and dative case
PP/nach:4	subcategorized PP with head *nach* and dative case
PP/neben:4	subcategorized PP with head *neben* and dative case
PP/ohne:8	subcategorized PP with head *ohne* and accusative case
PP/um:8	subcategorized PP with head *um* and accusative case
PP/unter:4	subcategorized PP with head *unter* and dative case
PP/unter:8	subcategorized PP with head *unter* and accusative case
PP/von:4	subcategorized PP with head *von* and dative case
PP/vor:4	subcategorized PP with head *vor* and dative case
PP/vor:8	subcategorized PP with head *vor* and accusative case
PP/wider:8	subcategorized PP with head *wider* and accusative case
PP/zu:4	subcategorized PP with head *zu* and dative case
PP/zwischen:4	subcategorized PP with head *zwischen* and dative case
PP/über:4	subcategorized PP with head *über*

	and dative case
PP/über:8	subcategorized PP with head *über* and accusative case
S/ZU	subcategorized infinitive with *zu*
S/daß	subcategorized sentence with *daß*
S/decl-vsec	subcategorized verb-second sentence
S/ob	subcategorized sentence with *ob*
S/w	subcategorized wh-sentence
dabei, dadurch, dafür, dagegen, dahinter, damit, danach, daran, darauf, daraufhin, daraus, darin, darum, darunter, darüber, davon, davor, dazu	correlating pronominal adverb

A.3 The LFG parser

The following describes the tags applied in the examples of the LFG parser output, based on the grammar by Rohrer and Forst (2006).

tag	description
A[-infl]	adjective (not inflected)
A[+infl]	adjective (inflected)
Acard	cardinal number
ADVfoc	focusing adverb
ADVP[std]	adverbial phrase
AP[std,-infl], APx[std,-infl]	adjective phrase and intermediate projection, adjective not inflected
AP[std,+infl], APx[std,+infl]	adjective phrase and intermediate projection, adjective inflected
Cbar	intermediate projection of complementizer phrase

Cbar-flat	intermediate projection of complementizer phrase with a flat structure
COLON	colon
COMMA	comma
CONJco	coordinating conjunction
CProot[std]	root of complementizer phrase
CPdep[rel]	dependent complementizer phrase with relative pronoun
D[std]	determiner
DP[rel], DPx[rel]	determiner phrase and intermediate projection with relative pronoun
DP[std], DPx[std]	determiner phrase and intermediate projection
DPtime	time expression
HAP-COMMA	operational comma (consecutive punctuation marks)
N[comm]	common noun
NAME	proper noun
NAMEP	noun phrase with proper noun
NP	noun phrase
P[pre]	preposition
PERIOD	period
PP[rel], PPx[rel]	prepositional phrase and intermediate projection with relative pronoun
PP[std], PPx[std]	prepositional phrase and intermediate projection
PREDP[std]	predicative argument
PRON[rel]	relative pronoun
PRON[std]	pronoun
ROOT	root node
V[coh,fin], Vx[coh,fin]	verb and intermediate projection, coherent construction, finite
V[cop,fin], Vx[cop,fin]	verb and intermediate projection, copular

	verb, finite
V[v,fin], Vx[v,fin]	verb and intermediate projection, full verb, finite
V[v,inf], Vx[v,inf]	verb and intermediate projection, full verb, infinitive
V[v,part]	verb, full verb, participle
VC[coh,fin]	verb complex, coherent construction, finite
VC[coh,inf]	verb complex, coherent construction, infinitive
VC[v,inf]	verb complex, full verb, infinitive
VC[v,part]	verb complex, full verb, participle
VP[coh,fin]	verb phrase, coherent construction, finite
VP[coh,inf]	verb phrase, coherent construction, infinitive
VP[v,part], VPx[v,part]	verb phrase and intermediate projection, participle
VPART	verb particle
YEAR	year

Appendix B

Resources

The following sections give more details on the actual processing steps which produce the examples shown in this work. Resources are given by means of name and version or by means of an additional persistent identifier, where available. If no explicit versioning could be found it is replaced by the download date. The resources are listed in the order of the examples displayed in the text.

B.1 Chapter 1

The tools applied for the analyses in Figures 1.1 and 1.6.

- Phrase structure tree by BitPar (Figure 1.1a)
  - Tool metadata PID:
    `http://hdl.handle.net/11022/1007-0000-0000-8E30-A`
  - Tool download: 09-08-2016
  - Grammar: trace grammar extracted from Penn Treebank
  - Grammar download: 13-09-2016
  - Application details: parsing script includes tokenizer
- Dependency tree by mate parser (Figure 1.1b)
  - Tool metadata PID:
    `http://hdl.handle.net/11022/1007-0000-0000-8E2B-1`
  - Tool version: anna-3.3.jar

  - Models:
    * CoNLL2009-ST-English-ALL.anna-3.3.lemmatizer.model
    * CoNLL2009-ST-English-ALL.anna-3.3.postagger.model
    * CoNLL2009-ST-English-ALL.anna-3.3.parser.model

- F-structure by LFG parser (Figure 1.1c)
  - XLE-Web at INESS page: `http://clarino.uib.no/iness`
  - Grammar: English
  - Processing date: 10-01-2017
  - Visualisation mode: PREDs only

- Phrase structure trees from n-best list by BitPar (Figure 1.6)
  - Tool metadata PID:
    `http://hdl.handle.net/11022/1007-0000-0000-8E30-A`
  - Tool download: 09-08-2016
  - Grammar: German UTF-8 grammar extracted from version 2[1] of the Tiger treebank
  - Grammar download: 09-08-2016
  - Application details: automatic tokenization of the input, n-best list and undone markovisation for the output representation

B.2 Chapter 2

The tools applied for the analyses of Examples (2.6) to (2.10). Figure 2.4 is an excerpt of the analysis in Figure 1.1b.

- mate parser
  - Tool metadata PID:
    `http://hdl.handle.net/11022/1007-0000-0000-8E2B-1`
  - Tool version: anna-3.61.jar
  - Models:
    * lemma-ger-3.6.model

[1]The version was not specified in more detail, 2.1 is assumed.

    - tag-ger-3.6.model
    - morphology-ger-3.6.model
    - parser-ger-3.6.model

- FSPar
  - Tool metadata PID:
    `http://hdl.handle.net/11022/1007-0000-0000-8E3D-D`
  - In-house CVS, checkout on 16-06-2017
  - Application details:
    - Changed TreeTagger call to installed version TreeTagger 3.2.1
      - Tool: tree-tagger-linux-3.2.1.tar.gz
      - Parameter file: german-par-linux-3.2-utf8.bin.gz
    - Output type: -x (tabular format similar to the CoNLL2006 or CoNLL-X format)

- BitPar
  - Tool metadata PID:
    `http://hdl.handle.net/11022/1007-0000-0000-8E30-A`
  - Tool download: 09-08-2016
  - Grammar: German UTF-8 grammar extracted from version 2[2] of the Tiger treebank
  - Grammar download: 09-08-2016
  - Application details: manual tokenization of the input and undone markovisation for the output representation

- IMS-SZEGED-CIS parser
  - Tool and models: in-house version (25-09-2013)

- LFG parser
  - Tool version: XLE release 07-05-2013 13:40
  - German LFG from 16-03-2009

[2]The version was not specified in more detail, 2.1 is assumed.

B.3 Chapter 3

The configuration applied for the BitPar analysis in Figure 3.1.

- Tool metadata PID:
  `http://hdl.handle.net/11022/1007-0000-0000-8E30-A`
- Tool download: 09-08-2016
- Grammar: German UTF-8 grammar extracted from version 2[3] of the Tiger treebank
- Grammar download: 09-08-2016
- Application details: parsing script includes tokenizer

B.4 Chapter 4

B3DB, PID: `http://hdl.handle.net/11022/1007-0000-0007-BFEA-B`

B.5 Chapter 5

The analyses displayed in this chapter refer to the analyses in Section 2.1.

B.6 Chapter 6

Corpora applied in the case studies of Task I and Task II:

- SdeWaC, PID:
  `http://hdl.handle.net/11022/1007-0000-0000-8E58-E`
- DIRNDL, PID:
  `http://hdl.handle.net/11022/1007-0000-0000-8E54-2`

The syntactic analyses have been created by the parsers as described in the resource sections of the tasks (Sections 6.1.1 and 6.2.1).

[3]The version was not specified in more detail, 2.1 is assumed.

Appendix C

Annotation guidelines

C.1 Recognition of sentences with *nach*-particle verbs

The following shows the annotation guidelines for the post hoc evaluation of the recognition of sentences with *nach*-particle verbs, described in Section 6.1.2 as a part of the task-related parser evaluation for Task I. Since the subject of annotation are German *nach*-particle verbs, the annotation guidelines are in German.

Annotationsrichtlinien zu *nach*-Partikelverbsätzen

Kerstin Eckart

23. Juli 2015
Version 1.1

Bei dieser Annotation geht es um die Frage:

> Enthält der Satz mindestens ein *nach*-Partikelverb?

Die Menge der möglichen Annotationswerte ist $\{\mathbf{0}, \mathbf{1}, \mathbf{?}\}$. Dabei wird für jeden Satz genau ein Wert vergeben.

0: Der Satz enthält kein *nach*-Partikelverb.

1: Der Satz enthält ein oder mehrere *nach*-Partikelverben.
(Im Prinzip kann nach dem ersten gefundenen *nach*-Partikelverb **1** annotiert und der Rest des Satzes ignoriert werden.)

?: Es ist nicht klar, ob der Satz ein *nach*-Partikelverb enthält, oder es liegt eine Ambiguität vor.

Dabei sind *nach*-Partikelverben Verben, die mit der Partikel *nach* beginnen, aber auch getrennt vorkommen können. Beispiele (1) und (2) sind Beispiele für *nach*-Partikelverben:

(1) Der Hund läuft dem Hasen nach. ⇒ **1**

(2) Um die Suppe nachzukochen, werden noch Zwiebeln benötigt. ⇒ **1**

Weiterhin sind folgende Punkte zu beachten:

- Es spielt keine Rolle, ob das entsprechende *nach*-Partikelverb bekannt ist, z.B. aus dem Duden-Wörterbuch, nur ob die Verwendung der eines *nach*-Partikelverbs entspricht. Beispiele (3) und (4) sollten also mit **1** annotiert werden:

(3) Die eine Kuh käut der anderen nach. ⇒ **1**

(4) Er wollte nicht alle Aufgaben nochmal nachmultiplizieren. ⇒ **1**

- Groß- und Kleinschreibung, Tippfehler sowie ungewöhnliche Zusammen- oder Getrenntschreibung spielen ebenfalls keine Rolle. Beispiele (5) und (6) sollten auch mit **1** annotiert werden:

(5) "Lauf ihm NACH!!!" wollte ich noch rufen. ⇒ **1**

(6) *Sie hatte vergessen nach zu fragen. ⇒ **1**

- Alle Sätze enthalten mindestens ein Wort, das mit *nach* anfängt bzw. das Wort *nach* selbst. Daher ist genau darauf zu achten, wann es sich um eine Verwendung als Partikelverb handelt. Beispiele (7) und (8) enthalten keine *nach*-Partikelverben:

(7) Er faltet das Blatt der Länge nach. ⇒ **0**
([der Länge nach], nicht: [nachfalten])

(8) Die Nachmeldung kann noch bis kurz vor dem Start erfolgen. ⇒ **0**
(nachmelden ist zwar ein nach-Partikelverb, Nachmeldung ist jedoch die Nominalisierung)

- Im Zweifel **?** vergeben, diese Fälle können später aufgelöst werden.

Appendix D

Graphics

The diagrams and graphics of Figures 1.2 to 1.5, 2.1, 2.2, 3.4, 4.1, 4.2, 4.4, 4.6, 4.8, 4.9, 5.1, 5.8, 6.2 and 7.1 to 7.3 have been created with Dia[1].

[1] https://wiki.gnome.org/Apps/Dia/

Bibliography

Abney, S. (1996). Partial Parsing via Finite-State Cascades. *Journal of Natural Language Engineering*, 2(4):337–344. 76

Albert, S., Anderssen, J., Bader, R., Becker, S., Bracht, T., Brants, S., Brants, T., Demberg, V., Dipper, S., Eisenberg, P., Hansen, S., Hirschmann, H., Janitzek, J., Kirstein, C., Langner, R., Michelbacher, L., Plaehn, O., Preis, C., Pußel, M., Rower, M., Schrader, B., Schwartz, A., Smith, G., and Uszkoreit, H. (2003). *TIGER Annotationsschema*. Universität des Saarlandes, Universität Stuttgart, Universität Potsdam, Germany. 62, 69, 158

Ballesteros, M. (2012). *Analyzing, Enhancing, Optimizing and Applying Dependency Analysis*. PhD thesis, Complutense University of Madrid, Spain. 103, 104, 105, 107, 108

Baroni, M., Bernardini, S., Ferraresi, A., and Zanchetta, E. (2009). The WaCky wide web: a collection of very large linguistically processed web-crawled corpora. *LRE*, 43(3):209–226. 210, 215, 221, 222

Baroni, M. and Kilgarriff, A. (2006). Large linguistically-processed web corpora for multiple languages. In *EACL-2006 Posters & Demonstrations*, pages 87–90, Trento, Italy. 221, 222

Baumann, S. and Riester, A. (2012). Referential and lexical givenness: Semantic, prosodic and cognitive aspects. In Elordieta, G. and Prieto, P., editors, *Prosody and Meaning*, volume 25 of *Interface Explorations*, pages 119–162. De Gruyter, Berlin, Germany. 50, 185, 263, 265

Björkelund, A., Çetinoğlu, Ö., Faleńska, A., Farkas, R., Müller, T., Seeker, W., and Szántó, Z. (2014a). The IMS-Wrocław-Szeged-CIS Entry at the SPMRL 2014 Shared Task: Reranking and Morphosyntax Meet Unlabeled Data. In *The First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*, Dublin, Ireland. 88, 300

Björkelund, A., Cetinoglu, O., Farkas, R., Mueller, T., and Seeker, W. (2013a). (Re)ranking Meets Morphosyntax: State-of-the-art Results from the SPMRL 2013 Shared Task. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 135–145, Seattle, Washington, USA. Association for Computational Linguistics. 88

Björkelund, A., Çetinoğlu, Ö., Farkas, R., Mueller, T., and Seeker, W. (2013b). (Re)ranking Meets Morphosyntax: State-of-the-art Results from the SPMRL 2013 Shared Task. In *Proceedings of the Fourth Workshop on Statistical Parsing of Morphologically-Rich Languages*, pages 135–145, Seattle, Washington, USA. Association for Computational Linguistics. 269

Björkelund, A., Eckart, K., Riester, A., Schauffler, N., and Schweitzer, K. (2014b). The Extended DIRNDL Corpus as a Resource for Coreference and Bridging Resolution. In Nicoletta Calzolari (Conference Chair), Choukri, K., Declerck, T., Loftsson, H., Maegaard, B., Mariani, J., Moreno, A., Odijk, J., and Piperidis, S., editors, *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, Reykjavik, Iceland. European Language Resources Association (ELRA). 35

Bohnet, B. (2010). Top Accuracy and Fast Dependency Parsing is not a Contradiction. In *Proceedings of the 23rd International Conference on Computational Linguistics (Coling 2010)*, pages 89–97, Beijing, China. 22, 47, 66, 71, 158

Bohnet, B. and Kuhn, J. (2012). The Best of BothWorlds – A Graph-based Completion Model for Transition-based Parsers. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 77–87, Avignon, France. Association for Computational Linguistics. 71

Bohnet, B. and Nivre, J. (2012). A Transition-Based System for Joint Part-of-Speech Tagging and Labeled Non-Projective Dependency Parsing. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1455–1465, Jeju Island, Korea. Association for Computational Linguistics. 71

Brants, S., Dipper, S., Eisenberg, P., Hansen-Schirra, S., König, E., Lezius, W., Rohrer, C., Smith, G., and Uszkoreit, H. (2004). TIGER: Linguistic Interpretation of a German Corpus. *Research on Language and Computation*, 2(4):597–620. 69, 224, 269

Brants, S., Dipper, S., Hansen, S., Lezius, W., and Smith, G. (2002). The TIGER Treebank. In *Proceedings of The First Workshop on Treebanks and Linguistic Theories (TLT2002)*, Sozopol, Bulgaria. 69

Buchholz, S. and Marsi, E. (2006). CoNLL-X Shared Task on Multilingual Dependency Parsing. In *Proceedings of CoNLL-X*, pages 149–164, New York, USA. 105, 118, 130, 298

Burchardt, A., Erk, K., Frank, A., Kowalski, A., and Pado, S. (2006). SALTO – A Versatile Multi-Level Annotation Tool. In *5th edition of the International Conference on Language Resources and Evaluation*, pages 517–520, Genoa, Italy. 36

Cassidy, S. (2010). An RDF realisation of LAF in the DADA annotation server. In *Proceedings of ISA-5*, Hong Kong, China. 301

Celko, J. (2004). *Trees and Hierarchies in SQL*. Morgan Kaufmann. 149, 178

Chiarcos, C. (2012). A generic formalism to represent linguistic corpora in RDF and OWL/DL. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey. 120

Chiarcos, C., Dipper, S., Götze, M., Leser, U., Lüdeling, A., Ritz, J., and Stede, M. (2008). A Flexible Framework for Integrating Annotations from Different Tools and Tagsets. In *Proceeding of the Conference on Global Interoperability for Language Resources*, Hong Kong, China. 145

Chiarcos, C., Eckart, K., and Ritz, J. (2010). Creating and Exploiting a Resource of Parallel Parses. In *Proceedings of the Fourth Linguistic Annotation Workshop*, pages 166–171, Uppsala, Sweden. Association for Computational Linguistics. 146

Chiarcos, C., Hellmann, S., Nordhoff, S., Moran, S., Littauer, R., Eckle-Kohler, J., Gurevych, I., Hartmann, S., Matuschek, M., and Meyer, C. M. (2012). The Open Linguistics Working Group. In Nicoletta Calzolari (Conference Chair), Choukri, K., Declerck, T., Doğan, M. U., Maegaard, B., Mariani, J., Moreno, A., Odijk, J., and Piperidis, S., editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey. European Language Resources Association (ELRA). 118, 131

Chiarcos, C. and Sukhareva, M. (2015). OLiA – Ontologies of Linguistic Annotation. *Semantic Web*, 6(4):379–386. 136

Chomsky, N. (1957). *Syntactic structures*, volume 4 of *Janua linguarum: Minor*. Mouton & Co, 's-Gravenhage. 58

Crouch, D., Dalrymple, M., Kaplan, R., King, T., Maxwell, J., and Newman, P. (2011). XLE Documentation. Palo Alto Research Center. `http://www2.parc.com/isl/groups/nltt/xle/doc/xle_toc.html`. 92, 265, 267

Crysmann, B., Hansen-Schirra, S., Smith, G., and Ziegler-Eisele, D. (2005). *TIGER Morphologie-Annotationsschema*. Universität des Saarlandes, Universität Potsdam, Germany. 175

Dannenberg, A., Werner, S., and Vainio, M. (2016). Prosodic and syntactic structures in spontaneous English speech. In *Speech Prosody 2016*, pages 59–63, Boston, USA. 300

Davies, M. (2005). The advantage of using relational databases for large corpora: Speed, advanced queries, and unlimited annotation. *International Journal of Corpus Linguistics*, 10(3):307–334. 150

de la Clergerie, E. V., Hamon, O., Mostefa, D., Ayache, C., Paroubek, P., and Vilnat, A. (2008). PASSAGE: from French Parser Evaluation to Large Sized

Treebank. In Nicoletta Calzolari (Conference Chair), Choukri, K., Maegaard, B., Mariani, J., Odijk, J., Piperidis, S., and Tapias, D., editors, *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, pages 3570–3577, Marrakech, Morocco. European Language Resources Association (ELRA). 118, 129, 141

de Marneffe, M.-C., Dozat, T., Silveira, N., Haverinen, K., Ginter, F., Nivre, J., and Manning, C. D. (2014). Universal Stanford Dependencies: a Cross-Linguistic Typology. In Nicoletta Calzolari (Conference Chair), Choukri, K., Declerck, T., Loftsson, H., Maegaard, B., Mariani, J., Moreno, A., Odijk, J., and Piperidis, S., editors, *Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC'14)*, Reykjavik, Iceland. European Language Resources Association (ELRA). 138, 139

Duden, Informatik (2003). *Duden, Informatik : ein Fachlexikon für Studium und Praxis*. Bibliographisches Institut & F. A. Brockhaus AG, Mannheim, Germany, 3 edition. Herausgegeben von der Redaktion Studium und Beruf. Bearbeitet von Volker Claus und Andreas Schwill. 21

Earley, J. (1970). An Efficient Context-free Parsing Algorithm. *Commun. ACM*, 13(2):94–102. 64

Eberle, K., Eckart, K., Heid, U., and Haselbach, B. (2012). A Tool/Database Interface for Multi-Level Analyses. In *Proceedings of the eighth conference on International Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey. European Language Resources Association (ELRA). 42

Eberle, K., Faaß, G., and Heid, U. (2009). Proposition oder Temporalangabe? Disambiguierung von -ung-Nominalisierungen von verba dicendi in nach-PPs. In Chiarcos, C., de Castilho, R. E., and Stede, M., editors, *Von der Form zur Bedeutung: Texte automatisch verarbeiten / From Form to Meaning: Processing Texts Automatically. Proceedings of the Biennial GSCL Conference 2009, Potsdam, Germany*, pages 81–91, Tübingen, Germany. Narr. 50, 145

Eberle, K., Heid, U., Kountz, M., and Eckart, K. (2008). A tool for corpus analysis using partial disambiguation and bootstrapping of the lexicon. In Storrer, A.,

Geyken, A., Siebert, A., and Würzner, K.-M., editors, *Text Resources and Lexical Knowledge: Selected Papers from the 9th Conference on Natural Language Processing KONVENS 2008*, volume 8 of *Text, Translation, Computational Processing [TTCP]*, pages 145–158. Mouton de Gruyter, Berlin, Germany. 41, 145, 147, 299

Eckart, K. (2009). Repräsentation von Unterspezifikation in relationalen Datenbanksystemen. Diplomarbeit, Institut für Parallele und Verteilte Systeme, Universität Stuttgart, Germany. 28, 76, 144, 145, 150, 163, 178, 179

Eckart, K. (2012). A standardized general framework for encoding and exchange of corpus annotations: The Linguistic Annotation Framework, LAF. In Jancsary, J., editor, *Proceedings of KONVENS 2012*, pages 506–515. ÖGAI. SFLR 2012 workshop. 30, 161

Eckart, K. and Gärtner, M. (2016). Creating Silver Standard Annotations for a Corpus of Non-Standard Data. In Dipper, S., Neubarth, F., and Zinsmeister, H., editors, *Proceedings of the 13th Conference on Natural Language Processing (KONVENS 2016)*, volume 16 of *BLA: Bochumer Linguistische Arbeitsberichte*, pages 90–96, Bochum, Germany. 204, 291, 298, 300, 301

Eckart, K. and Heid, U. (2014). Resource interoperability revisited. In *Proceedings of the 12th edition of the KONVENS conference*, volume 1, pages 116–126, Hildesheim, Germany. 119

Eckart, K., Riester, A., and Schweitzer, K. (2012). A Discourse Information Radio News Database for Linguistic Analysis. In Chiarcos, C., Nordhoff, S., and Hellmann, S., editors, *Linked Data in Linguistics. Representing and Connecting Language Data and Language Metadata*, pages 65–75. Springer-Verlag Berlin Heidelberg, Germany. 31, 35, 163, 266

Eckart, K. and Seeker, W. (2013). Task-based Parser Output Combination. ESSLLI-13 Workshop on Extrinsic Parse Improvement (EPI). Düsseldorf, Germany. 206

Ehrich, V. and Rapp, I. (2000). Sortale Bedeutung und Argumentstruktur: ung-Nominalisierungen im Deutschen. *Zeitschrift für Sprachwissenschaft*, 19(2):245 – 303. 144

Elming, J., Johannsen, A., Klerke, S., Lapponi, E., Martinez Alonso, H., and Søgaard, A. (2013). Down-stream effects of tree-to-dependency conversions. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 617–626, Atlanta, Georgia, USA. Association for Computational Linguistics. 38

Faaß, G. and Eckart, K. (2013). SdeWaC – A Corpus of Parsable Sentences from the Web. In Gurevych, I., Biemann, C., and Zesch, T., editors, *Language Processing and Knowledge in the Web*, volume 8105 of *Lecture Notes in Computer Science*, pages 61–68. Springer-Verlag Berlin Heidelberg, Germany. 31, 77, 221, 223, 224

Farrar, S. and Langendoen, D. T. (2003). A linguistic ontology for the Semantic Web. *GLOT International*, 7(3):97–100. 136

Fiscus, J. G. (1997). A Post-Processing System To Yield Reduced Word Error Rates: Recognizer Output Voting Error Reduction (ROVER). In *IEEE Workshop on Automatic Speech Recognition and Understanding*, pages 347–354, Santa Barbara, California, USA. 49, 98, 99, 100, 102

Forst, M. and Kaplan, R. M. (2006). The importance of precise tokenizing for deep grammars. In *5th edition of the International Conference on Language Resources and Evaluation*, pages 369–372, Genoa, Italy. 92, 268, 269

Frederking, R. and Nirenburg, S. (1994). Three Heads are Better than One. In *Proceedings of the Fourth Conference on Applied Natural Language Processing*, pages 95–100, Stuttgart, Germany. Association for Computational Linguistics. 100, 101

Gärtner, M., Thiele, G., Seeker, W., Björkelund, A., and Kuhn, J. (2013). ICARUS – An Extensible Graphical Search Tool for Dependency Treebanks. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, Sofia, Bulgaria. Association for Computational Linguistics. 201

Hahn, U., Hermann, S., Enderle, P., Fritze, F., Gärtner, M., and Kushnarenko, V. (2017). RePlay-DH - Realisierung einer Plattform und begleitender Dienste

zum Forschungsdatenmanagement für die Fachcommunity - Digital Humanities. Poster, http://archiv.ub.uni-heidelberg.de/volltextserver/22886/. 296

Hajič et al., J. (2009). The CoNLL-2009 Shared Task: Syntactic and Semantic Dependencies in Multiple Languages. In *Proceedings of CoNLL 2009: Shared Task*, pages 1–18, Boulder, Colorado, USA. 71, 105, 130, 149, 159, 298

Halle, M. and Marantz, A. (1993). Distributed Morphology and the Pieces of Inflection. In Hale, K. and Keyser, S. J., editors, *The View from Building 20: Essays in Linguistics in Honor of Sylvain Bromberger*, volume 24 of *Current Studies in Linguistics*, pages 111–176. The MIT Press, Cambridge, MA, USA. 208

Haselbach, B. (2011). Deconstructing the German verb particle *nach* at the syntax-semantics interface. *Generative Grammar in Geneva (GG@G)*, 7:71–92. 185, 207, 208, 209, 210, 211, 212, 213, 214, 215, 218, 219, 220, 221, 234, 235, 237, 247, 260

Haselbach, B. (2017). Ps At The Interfaces: On The Syntax, Semantics, And Morphology Of Spatial Prepositions In German. Unpublished manuscript of submitted PhD thesis, Universität Stuttgart, Germany. 208

Haselbach, B. (n.d.). Annotation guidelines for nach-verbs. Internal Manual, Universität Stuttgart, Germany. 235

Haselbach, B., Eckart, K., Seeker, W., Eberle, K., and Heid, U. (2012a). Approximating theoretical linguistics classification in real data: the case of German *nach* particle verbs. In *Proceedings of COLING 2012*, Mumbai, India. 50, 206, 208, 211, 212, 216, 217, 219, 221, 235, 237, 238, 249, 250, 258, 262

Haselbach, B., Seeker, W., and Eckart, K. (2012b). German "nach"-Particle Verbs in Semantic Theory and Corpus Data. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey. 206, 207, 208, 210, 212

Heid, U., Schmid, H., Eckart, K., and Hinrichs, E. (2010). A Corpus Representation Format for Linguistic Web Services: The D-SPIN Text Corpus Format and its Relationship with ISO Standards. In Nicoletta Calzolari (Conference Chair), Choukri, K., Maegaard, B., Mariani, J., Odijk, J., Piperidis, S., Rosner, M., and Tapias, D., editors, *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta. European Language Resources Association (ELRA). 121, 131, 141

Henderson, J. C. and Brill, E. (1999). Exploiting Diversity in Natural Language Processing: Combining Parsers. In *Proceedings of the 1999 Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora (EMNLP 1999)*, pages 187–194, College Park, Maryland, USA. Association for Computational Linguistics. 25, 49, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 276

Hinrichs, E., Hinrichs, M., and Zastrow, T. (2010). WebLicht: Web-Based LRT Services for German. In *Proceedings of the ACL 2010 System Demonstrations*, pages 25–29, Uppsala, Sweden. Association for Computational Linguistics. 118, 130, 296

Ide, N. and Pustejovsky, J. (2010). What Does Interoperability Mean, anyway? Toward an Operational Definition of Interoperability for Language Technology. In *Proceedings of the Second International Conference on Global Interoperability for Language Resources (ICGL 2010)*, Hong Kong, China. 118, 119, 120, 140

Ide, N. and Romary, L. (2006). Representing linguistic corpora and their annotations. In *Proceedings of the Fifth International Conference on Language Resources and Evaluation (LREC 2006)*, Genoa, Italy. 161

Ide, N. and Suderman, K. (2007). GrAF: A Graph-based Format for Linguistic Annotations. In *Proceedings of the Linguistic Annotation Workshop*, pages 1–8, Prague, Czech Republic. Association for Computational Linguistics. 132, 141, 161

Ide, N. and Suderman, K. (2012). A Model for Linguistic Resource Description. In *Proceedings of the Sixth Linguistic Annotation Workshop*, pages 57–66, Jeju, Republic of Korea. Association for Computational Linguistics. 30, 301

Ide, N. and Suderman, K. (2014). The Linguistic Annotation Framework: a standard for annotation interchange and merging. *Language Resources and Evaluation*, 48(3):395–418. 132, 161, 164, 175, 176

ISO 12620:2009 (2009). ISO 12620:2009 Terminology and other language and content resources – Specification of data categories and management of a Data Category Registry for language resources. 118, 133, 152, 163

ISO 24610-1:2006 (2006). ISO 24610-1:2006 Language resource management – Feature structures – Part 1: Feature structure representation. 174, 175, 176, 177, 178

ISO 24612:2012 (2012). ISO 24612:2012 Language resource management – Linguistic annotation framework (LAF). 118, 132, 141, 144, 145, 161, 164, 289, 295

ISO 24615-1:2014 (2014). ISO 24615-1:2014 Language resource management – Syntactic annotation framework (SynAF) – Part 1: Syntactic model. 69

ISO/IEC 8859-1:1998 (1998). ISO/IEC 8859-1:1998 Information technology – 8-bit single-byte coded graphic character sets – Part 1: Latin alphabet No. 1. 225

ISO/PRF 24615-2 (2017). ISO/PRF 24615-2 Language resource management – Syntactic annotation framework (SynAF) – Part 2: XML serialization (ISOTiger). 69

Ivanova, K. (2010). Integritätsbedingungen für die Datenkonsistenz der B3-Datenbank. Diplomarbeit, Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart, Germany. 144

Kamp, H. and Reyle, U. (1993). *From Discourse to Logic. Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and Discourse Representation Theory*. Kluwer, Dordrecht, the Netherlands. 208

Kaplan, R. M. and Bresnan, J. (1982). Lexical-Functional Grammar: A formal system for grammatical representation. In Bresnan, J., editor, *The Mental Representation of Grammatical Relations*, pages 173–281. The MIT Press, Cambridge, MA, USA. Reprinted in Mary Dalrymple, Ronald M. Kaplan, John Maxwell, and Annie Zaenen, eds., Formal Issues in Lexical-Functional Grammar, 29–130. Stanford: Center for the Study of Language and Information. 1995. 92

Karlsson, F. (1990). Constraint Grammar as a Framework for Parsing Unrestricted Text. In *COLING-90 Papers presented to the 13th International Conference on Computational Linguistics*, volume 3, Helsinki, Finland. 68

Katz-Brown, J., Petrov, S., McDonald, R., Och, F., Talbot, D., Ichikawa, H., Seno, M., and Kazawa, H. (2011). Training a Parser for Machine Translation Reordering. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 183–192, Edinburgh, Scotland, UK. Association for Computational Linguistics. 113, 115

König, E., Lezius, W., and Voormann, H. (2003). *TIGERSearch 2.1 User's Manual. Chapter V - The TIGER-XML treebank encoding format*. Universität Stuttgart, Germany. 69

Kountz, M., Heid, U., and Eckart, K. (2008). A LAF/GrAF based Encoding Scheme for underspecified Representations of syntactic Annotations. In Nicoletta Calzolari (Conference Chair), Choukri, K., Maegaard, B., Mariani, J., Odijk, J., Piperidis, S., and Tapias, D., editors, *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, pages 2262–2269, Marrakech, Morocco. European Language Resources Association (ELRA). 144, 161, 164, 174, 178, 180

Kountz, M., Heid, U., and Spranger, K. (2007). Automatic Sortal Interpretation of German Nominalisations with *-ung*. Towards using Underspecified Representations in Corpora. In *Proceedings of Corpus Linguistics 2007*, Birmingham, England, UK. University of Birmingham. 50

Krause, T., Lüdeling, A., Odebrecht, C., Romary, L., Schirmbacher, P., and Zielke, D. (2014). LAUDATIO-Repository: Accessing a heterogeneous field of linguistic corpora with the help of an open access repository. Digital Humanities 2014 Conference. Poster Session. `https://www.linguistik.hu-berlin.de/de/institut/professuren/korpuslinguistik/mitarbeiter-innen/carolin/dh2014_krause_et._al.pdf`. Lausanne, Switzerland. 145, 296

Krause, T. and Zeldes, A. (2016). ANNIS3: A new architecture for generic corpus query and visualization. *Digital Scholarship in the Humanities*, 31(1):118–139. 145

Kübler, S., McDonald, R., and Nivre, J. (2009). *Dependency Parsing*. Synthesis Lectures on Human Language Technologies. Morgan & Claypool. 54, 65, 66, 188, 189

Landis, J. R. and Koch, G. G. (1977). The measurement of observer agreement for categorical data. *Biometrics*, 33(1):159–174. 239

Langer, H. (2004). Syntax und Parsing. In Carstensen, K.-U., Ebert, C., Endriss, C., Jekat, S., Klabunde, R., and Langer, H., editors, *Computerlinguistik und Sprachtechnologie*, chapter 3.4, pages 232–275. Elsevier, München, Germany, second edition. 54, 57, 60, 61, 63, 64, 65, 68, 83, 92

Le Roux, J., Foster, J., Wagner, J., Kaljahi, R. S. Z., and Bry, A. (2012). DCU-Paris13 Systems for the SANCL 2012 Shared Task. In *Working Notes of SANCL 2012*, Montreal, Canada. 300

Leech, G. (1993). Corpus annotation schemes. *Literary and Linguistic Computing*, 8(4):275–281. 33

Lemnitzer, L. and Zinsmeister, H. (2015). *Korpuslinguistik – Eine Einführung*. narr Studienbücher. Narr Francke Attempto Verlag, Tübingen, Germany, 3 edition. 195

Levin, B. (1993). *English Verb Classes and Alternations: A Preliminary Investigation*. University of Chicago Press, Chicago, IL, USA. 210

Lezius, W. (2002). *Ein Suchwerkzeug für syntaktisch annotierte Textkorpora*. PhD thesis, Universität Stuttgart, Germany. Arbeitspapiere des Instituts für Maschinelle Sprachverarbeitung (AIMS), volume 8, number 4. 149, 201

Lippman, D. (2013). *Math in Society*. 2.4 edition. 280

Lu, W. (2013). Funktionen für die Tool-Versionierung in der B3-Datenbank. Studienarbeit, Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart, Germany. 144

Ludewig, J. and Lichter, H. (2013). *Software Engineering*. dpunkt.verlag, Heidelberg, Germany. 35, 41, 146

Marcus, M. P., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2). Special Issue on Using Large Corpora: II. 104, 121, 228

Mayer, J. (1995). Transcription of German Intonation. The Stuttgart System. ms. Universität Stuttgart. `http://www.ims.uni-stuttgart.de/institut/arbeitsgruppen/phonetik/papers/STGTsystem.pdf`. 265

McDonald, R., Lerman, K., and Pereira, F. (2006). Multilingual Dependency Analysis with a Two-Stage Discriminative Parser. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 216–220, New York City, USA. Association for Computational Linguistics. 105

McDonald, R. and Nivre, J. (2011). Analyzing and Integrating Dependency Parsers. *Computational Linguistics*, 37(1):197–230. 25, 47, 49, 50, 103, 104, 105, 107, 111

McDonald, R., Nivre, J., Quirmbach-Brundage, Y., Goldberg, Y., Das, D., Ganchev, K., Hall, K., Petrov, S., Zhang, H., Täckström, O., Bedini, C., Bertomeu Castelló, N., and Lee, J. (2013). Universal Dependency Annotation for Multilingual Parsing. In *Proceedings of the 51st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 92–97, Sofia, Bulgaria. Association for Computational Linguistics. 137, 139

McEnery, T. and Wilson, A. (2001). *Corpus linguistics. An introduction.* Edinburgh textbooks in empirical linguistics. Edinburgh University Press, Edinburgh, Scotland, UK, 2nd edition. 33

McIntyre, A. (2006). The interpretation of German datives and English *have*. In Hole, D., Meinunger, A., and Abraham, W., editors, *Datives and Other Cases: Between Argument Structure and Event Structure*, pages 185–211. John Benjamins, Amsterdam, Netherlands. 216

Nivre, J., Agić, Ž., Ahrenberg, L., Aranzabe, M. J., Asahara, M., Atutxa, A., Ballesteros, M., Bauer, J., Bengoetxea, K., Bhat, R. A., Bick, E., Bosco, C., Bouma, G., Bowman, S., Candito, M., Cebiroğlu Eryiğit, G., Celano, G. G. A., Chalub, F., Choi, J., Çöltekin, Ç., Connor, M., Davidson, E., de Marneffe, M.-C., de Paiva, V., Diaz de Ilarraza, A., Dobrovoljc, K., Dozat, T., Droganova, K., Dwivedi, P., Eli, M., Erjavec, T., Farkas, R., Foster, J., Freitas, C., Gajdošová, K., Galbraith, D., Garcia, M., Ginter, F., Goenaga, I., Gojenola, K., Gökırmak, M., Goldberg, Y., Gómez Guinovart, X., Gonzáles Saavedra, B., Grioni, M., Grūzītis, N., Guillaume, B., Habash, N., Hajič, J., Hà Mỹ, L., Haug, D., Hladká, B., Hohle, P., Ion, R., Irimia, E., Johannsen, A., Jørgensen, F., Kaşıkara, H., Kanayama, H., Kanerva, J., Kotsyba, N., Krek, S., Laippala, V., Lê Hồng, P., Lenci, A., Ljubešić, N., Lyashevskaya, O., Lynn, T., Makazhanov, A., Manning, C., Mărănduc, C., Mareček, D., Martínez Alonso, H., Martins, A., Mašek, J., Matsumoto, Y., McDonald, R., Missilä, A., Mititelu, V., Miyao, Y., Montemagni, S., More, A., Mori, S., Moskalevskyi, B., Muischnek, K., Mustafina, N., Müürisep, K., Nguyễn Thị, L., Nguyễn Thị Minh, H., Nikolaev, V., Nurmi, H., Ojala, S., Osenova, P., Øvrelid, L., Pascual, E., Passarotti, M., Perez, C.-A., Perrier, G., Petrov, S., Piitulainen, J., Plank, B., Popel, M., Pretkalniņa, L., Prokopidis, P., Puolakainen, T., Pyysalo, S., Rademaker, A., Ramasamy, L., Real, L., Rituma, L., Rosa, R., Saleh, S., Sanguinetti, M., Saulīte, B., Schuster, S., Seddah, D., Seeker, W., Seraji, M., Shakurova, L., Shen, M., Sichinava, D., Silveira, N., Simi, M., Simionescu, R., Simkó, K., Šimková, M., Simov, K., Smith, A., Suhr, A., Sulubacak, U., Szántó, Z., Taji, D., Tanaka, T., Tsarfaty, R., Tyers, F., Uematsu, S., Uria, L., van Noord, G., Varga, V., Vincze, V., Washington, J. N., Žabokrtský, Z., Zeldes, A., Zeman, D., and Zhu, H.

(2017). Universal Dependencies 2.0. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University. 138

Nivre, J., de Marneffe, M.-C., Ginter, F., Goldberg, Y., Hajic, J., Manning, C. D., McDonald, R., Petrov, S., Pyysalo, S., Silveira, N., Tsarfaty, R., and Zeman, D. (2016). Universal Dependencies v1: A Multilingual Treebank Collection. In Nicoletta Calzolari (Conference Chair), Choukri, K., Declerck, T., Goggi, S., Grobelnik, M., Maegaard, B., Mariani, J., Mazo, H., Moreno, A., Odijk, J., and Piperidis, S., editors, *Proceedings of the Tenth International Conference on Language Resources and Evaluation (LREC 2016)*, Portorož, Slovenia. European Language Resources Association (ELRA). 118, 138

Nivre, J., Hall, J., Kübler, S., McDonald, R., Nilsson, J., Riedel, S., and Yuret, D. (2007). The CoNLL 2007 Shared Task on Dependency Parsing. In *Proceedings of the CoNLL Shared Task Session of EMNLP-CoNLL 2007*, pages 915–932, Prague, Czech Republic. Association for Computational Linguistics. 105

Nivre, J., Hall, J., Nilsson, J., Eryiğit, G., and Marinov, S. (2006). Labeled Pseudo-Projective Dependency Parsing with Support Vector Machines. In *Proceedings of the Tenth Conference on Computational Natural Language Learning (CoNLL-X)*, pages 221–225, New York City, USA. Association for Computational Linguistics. 105

Noha, A. (2016). Mapping of text and speech tokenizations: the DIRNDL case. Masterarbeit, Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart, Germany. 266

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics. 39, 114

PASSAGE-L1 (2009). Définition du formalisme d'annotation. ANR-PASSAGE Livrable: L1. version 1.14, `http://atoll.inria.fr/passage/docs/TAGMATICA_LIMSI_L1.pdf`. 129

Petrov, S., Das, D., and McDonald, R. (2012). A Universal Part-of-Speech Tagset. In Nicoletta Calzolari (Conference Chair), Choukri, K., Declerck, T., Doğan, M. U., Maegaard, B., Mariani, J., Moreno, A., Odijk, J., and Piperidis, S., editors, *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey. European Language Resources Association (ELRA). 136, 137, 138, 139

Petrov, S. and McDonald, R. (2012). Overview of the 2012 Shared Task on Parsing the Web. In *Notes of SANCL 2012*, Montreal, Canada. 105, 298

Quasthoff, U., Richter, M., and Biemann, C. (2006). Corpus Portal for Search in Monolingual Corpora. In *5th edition of the International Conference on Language Resources and Evaluation*, pages 1799–1802, Genoa, Italy. 223

Rapp, S. (1995). Automatic Phonemic Transcription and Linguistic Annotation from Known Text with Hidden Markov Models – An Aligner for German. In *Proceedings of ELSNET Goes East and IMACS Workshop "Integration of Language and Speech in Academia and Industry"*, Moscow, Russia. 265

Raubal, M. (2011). Entwicklung und Anwendung eines Templates-Mechanismus zur Datenextraktion aus Graphstrukturen linguistischer Datenbanken: TIGER vs. B3-Datenbank. Diplomarbeit, Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart, Germany. 144

Riester, A. and Baumann, S. (2013). Focus triggers and focus types from a corpus perspective. *Dialogue and Discourse*, 4(2):215–248. 263, 264, 265

Riester, A. and Baumann, S. (2017). *The RefLex Scheme – Annotation Guidelines*, volume 14 of *SinSpeC. Working Papers of the SFB 732*. University of Stuttgart, Stuttgart, Germany. 185, 263, 265

Riester, A. and Piontek, J. (2015). Anarchy in the NP. When new nouns get deaccented and given nouns don't. *Lingua*, 165:230–253. Prosody and Information Status in Typological Perspective. 143

Rohrer, C. and Forst, M. (2006). Improving coverage and parsing quality of a large-scale LFG for German. In *5th edition of the International Conference on*

Language Resources and Evaluation, pages 2206–2211, Genoa, Italy. 92, 265, 267, 311

Rosén, V., Smedt, K. D., Meurer, P., and Dyvik, H. (2012). An Open Infrastructure for Advanced Treebanking. In Hajič, J., Smedt, K. D., Tadić, M., and Branco, A., editors, *META-RESEARCH Workshop on Advanced Treebanking*, pages 22–29, Istanbul, Turkey. LREC 2012 Workshop. 22

Rosenberg, A. and Hirschberg, J. (2007). V-Measure : A conditional entropy-based external cluster evaluation measure. In *EMNLP 2007*, pages 410–420, Prague, Czech Republic. 247

Roßdeutscher, A. and Kamp, H. (2010). Syntactic and Semantic Constraints in the Formation and Interpretation of *ung*-Nouns. In Alexiadou, A. and Rathert, M., editors, *The Semantics of Nominalisations across Languages and Frameworks*, volume 22 of *Interface Explorations*, pages 169–214. De Gruyter Mouton, Berlin, Germany. 208

Sagae, K. and Lavie, A. (2006). Parser Combination by Reparsing. In *Proceedings of the Human Language Technology Conference of the NAACL, Companion Volume: Short Papers*, pages 129–132, New York City, USA. Association for Computational Linguistics. 103, 104, 105, 106, 107, 108, 110

Schiehlen, M. (2003). A Cascaded Finite-State Parser for German. In *Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistics (EACL 2003)*, pages 163–166, Budapest, Hungary. 76, 147

Schiller, A., Teufel, S., Stöckert, C., and Thielen, C. (1999). Guidelines für das Tagging deutscher Textcorpora mit STTS. Universität Stuttgart and Universität Tübingen, Germany. `http://www.ims.uni-stuttgart.de/forschung/ressourcen/lexika/TagSets/stts-1999.pdf`. 76, 192, 303

Schmid, H. (1994). Probabilistic Part-of-Speech Tagging Using Decision Trees. In *Proceedings of International Conference on New Methods in Language Processing*, pages 44–49, Manchester, England, UK. 76, 147, 222

Schmid, H. (2004). Efficient Parsing of Highly Ambiguous Context-Free Grammars with Bit Vectors. In *Proceedings of the 20th International Conference on Computational Linguistics (COLING 2004)*, Geneva, Switzerland. 22, 64, 83, 147, 269

Schmid, H. (2006). Trace Prediction and Recovery with Unlexicalized PCFGs and Slash Features. In *Proceedings of the 21st International Conference on Computational Linguistics and 44th Annual Meeting of the Association for Computational Linguistics*, pages 177–184, Sydney, Australia. Association for Computational Linguistics. 22, 83

Schmid, H., Fitschen, A., and Heid, U. (2004). SMOR: A German Computational Morphology Covering Derivation, Composition, and Inflection. In Lino, M. T., Xavier, M. F., Ferreira, F., Costa, R., Silva, R., with the collaboration of Pereira, C., Carvalho, F., Lopes, M., Catarino, M., and Barros, S., editors, *LREC 2004 Fourth International Conference on Language Resources and Evaluation*, pages 1263–1266, Lisboa, Portugal. European Language Resources Association (ELRA). 225

Schweitzer, K., Eckart, K., Dogil, G., and Augurzky, P. (2012). Prosodic Balance. Poster presentation at the 13th Conference on Laboratory Phonology. Stuttgart, Germany. 143

Seddah, D., Kübler, S., and Tsarfaty, R. (2014). Introducing the SPMRL 2014 Shared Task on Parsing Morphologically-rich Languages. In *Proceedings of the First Joint Workshop on Statistical Parsing of Morphologically Rich Languages and Syntactic Analysis of Non-Canonical Languages*, pages 103–109, Dublin, Ireland. Dublin City University. 88, 106

Seddah, D., Tsarfaty, R., Kübler, S., Candito, M., Choi, J. D., Farkas, R., Foster, J., Goenaga, I., Gojenola Galletebeitia, K., Goldberg, Y., Green, S., Habash, N., Kuhlmann, M., Maier, W., Nivre, J., Przepiórkowski, A., Roth, R., Seeker, W., Versley, Y., Vincze, V., Woliński, M., and Wróblewska, A. (2013). Overview of the SPMRL 2013 Shared Task: A Cross-Framework Evaluation of Parsing Morphologically Rich Languages. In *Proceedings of the Fourth Workshop on*

Statistical Parsing of Morphologically-Rich Languages, pages 146–182, Seattle, Washington, USA. Association for Computational Linguistics. 88, 106, 269, 298

Seeker, W., Bohnet, B., Øvrelid, L., and Kuhn, J. (2010). Informed ways of improving data-driven dependency parsing for German. In *Coling 2010: Posters*, pages 1122–1130, Beijing, China. Coling 2010 Organizing Committee. 69

Seeker, W. and Çetinoğlu, Ö. (2015). A Graph-based Lattice Dependency Parser for Joint Morphological Segmentation and Syntactic Analysis. *Transactions of the Association for Computational Linguistics*, 3:359–373. 54, 61

Seeker, W. and Kuhn, J. (2012). Making Ellipses Explicit in Dependency Conversion for a German Treebank. In *Proceedings of the Eight International Conference on Language Resources and Evaluation (LREC'12)*, Istanbul, Turkey. 71, 225

Smith, G. (2003). *A Brief Introduction to the TIGER Treebank, Version 1*. Universität Potsdam, Germany. 303, 305, 306

Spranger, K. and Heid, U. (2007). Applying Constraints derived from the Context in the process of Incremental Sortal Specification of German *-ung*-Nominalizations. In Christiansen, H. and Villadsen, J., editors, *Proceedings of the 4th International Workshop on Constraints and Language Processing (CSLP@Context 07)*, pages 65–77, Roskilde, Denmark. 50

Stede, M. and Huang, C.-R. (2012). Inter-operability and reusability: the science of annotation. *Language Resources and Evaluation*, 46(1):91–94. 118, 119, 120, 132

Surdeanu, M., Johansson, R., Meyers, A., Màrquez, L., and Nivre, J. (2008). The CoNLL 2008 Shared Task on Joint Parsing of Syntactic and Semantic Dependencies. In *CoNLL 2008: Proceedings of the Twelfth Conference on Computational Natural Language Learning*, pages 159–177, Manchester, England, UK. Coling 2008 Organizing Committee. 105

Surdeanu, M. and Manning, C. D. (2010). Ensemble Models for Dependency Parsing: Cheap and Good? In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 649–652, Los Angeles, California, USA. Association for Computational Linguistics. 49, 103, 104, 105, 107, 108, 109, 110, 111, 112, 113, 287

Tapanainen, P. and Voutilainen, A. (1994). Tagging accurately - Don't guess if you know. In *Proceedings of the Fourth Conference on Applied Natural Language Processing*, pages 47–52, Stuttgart, Germany. Association for Computational Linguistics. 49, 100, 101, 102

Tsarfaty, R., Nivre, J., and Andersson, E. (2011). Evaluating Dependency Parsing: Robust and Heuristics-Free Cross-Annotation Evaluation. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 385–396, Edinburgh, Scotland, UK. Association for Computational Linguistics. 135, 189, 190

Tsarfaty, R., Nivre, J., and Andersson, E. (2012). Cross-Framework Evaluation for Statistical Parsing. In *Proceedings of the 13th Conference of the European Chapter of the Association for Computational Linguistics*, pages 44–54, Avignon, France. Association for Computational Linguistics. 118, 135

Turing, A. M. (1950). Computing Machinery and Intelligence. *Mind*, LIX(236):433–460. 24

Ulusoy, G. (2014). Workflowvisualisierung in der B3-Datenbank. Diplomarbeit, Institut für Maschinelle Sprachverarbeitung, Universität Stuttgart, Germany. 40, 144

Utt, J., Springorum, S., Köper, M., and Schulte im Walde, S. (2014). Fuzzy V-Measure – An Evaluation Method for Cluster Analyses of Ambiguous Data. In *Proceedings of the 9th International Conference on Language Resources and Evaluation*, pages 581–587, Reykjavik, Iceland. 262

Vilnat, A., Paroubek, P., de la Clergerie, E. V., Francopoulo, G., and Guénot, M.-L. (2010). PASSAGE Syntactic Representation: a Minimal Common Ground

for Evaluation. In Nicoletta Calzolari (Conference Chair), Choukri, K., Maegaard, B., Mariani, J., Odijk, J., Piperidis, S., Rosner, M., and Tapias, D., editors, *Proceedings of the Seventh International Conference on Language Resources and Evaluation (LREC'10)*, Valletta, Malta. European Language Resources Association (ELRA). 129

Ward, J. H. (1963). Hierarchical grouping to optimize an objective function. *JASA*, 58(301):236–244. 221, 246

Witt, A., Heid, U., Sasaki, F., and Sérasset, G. (2009). Multilingual language resources and interoperability. *Language Resources and Evaluation*, 43(1):1–14. 33, 34, 118, 119, 120

Xu, F., Li, H., Zhang, Y., Uszkoreit, H., and Krause, S. (2011). Minimally Supervised Domain-Adaptive Parse Reranking for Relation Extraction. In *Proceedings of the 12th International Conference on Parsing Technologies*, pages 118–128, Dublin, Ireland. Association for Computational Linguistics. 114

Zarrieß, S., Schäfer, F., and Schulte im Walde, S. (2013). Passives of Reflexives: a Corpus Study. Abstract and Talk at LinguisticEvidence – Berlin Special. Berlin, Germany. 50

Zeman, D. (2008). Reusable Tagset Conversion Using Tagset Drivers. In Nicoletta Calzolari (Conference Chair), Choukri, K., Maegaard, B., Mariani, J., Odijk, J., Piperidis, S., and Tapias, D., editors, *Proceedings of the Sixth International Conference on Language Resources and Evaluation (LREC'08)*, pages 213–218, Marrakech, Morocco. European Language Resources Association (ELRA). 138

Zeman, D. and Žabokrtský, Z. (2005). Improving Parsing Accuracy by Combining Diverse Dependency Parsers. In *Proceedings of the Ninth International Workshop on Parsing Technologies (IWPT 2005)*, pages 171–178, Vancouver, British Columbia, Canada. 25, 49, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 296

Zinsmeister, H., Kuhn, J., and Dipper, S. (2002). TIGER Transfer – Utilizing LFG Parses for Treebank Annotation. In Butt, M. and King, T. H., editors, *The*

Proceedings of the LFG '02 Conference, Stanford, CA, USA. CSLI Publications. 92

Zipser, F. and Romary, L. (2010). A model oriented approach to the mapping of annotation formats using standards. In *Language Resource and Language Technology Standards – state of the art, emerging needs, and future developments*, Valletta, Malta. LREC 2010 Workshop. 118, 135, 295

All links were last followed on 4th August 2017.

Aknowledgements

I would like to thank Jonas Kuhn for taking me on as a PhD student, for always having the right balance between asking questions and granting freedom and for his constant support. I want to thank Ulrich Heid for making me part of the SFB, for his support ever since, for getting this thesis started and running and for all the discussions on ambiguity, databases, standards and interoperability. Thanks to Andreas Witt for agreeing to be the third examiner and joining the committee for this thesis.

This work comes to life with its case studies. I am grateful to Arndt Riester for the collaboration on his information status data and to Boris Haselbach and Wolfgang Seeker for the collaboration on nach-particle verbs, for the inspiring discussions from our three points of view and for being a wonderful support to get into writing.

I am thankful to my project colleagues Katrin Schweitzer and Markus Gärtner for their support and understanding in the last part of this thesis, to Özlem Çetinoğlu for her support regarding the LFG grammar, to all annotators involved in the case studies, to the Mensa-group for providing an anchor point in bustling days and to all people at the IMS for making it this great place to work and study.

I wish to thank all the people inside and outside of the IMS, sharing and discussing the PhD experience with me, Gertrud, Carolin, Michael, Marie, Benjamin, to name just a few. Thanks to Jürn and Taylor for answering my questions, thanks to Henrik for being the best companion in finishing this thesis and thanks to Dominik for keeping me balanced.

I want to thank my parents for their constant support, for the joy of learning they gave me and for the confidence that with some good literature and the right amount of time, you can master nearly every task.

The made-up sentence examples relating to football are a tribute to the not at all made-up story of the football team SV Darmstadt 98. In the years between 2013 and 2015 they managed to ascend from the team facing relegation into the 4th league to a team playing in the 1st league of the German Football Association (DFB).

This work was conducted in the context of Sonderforschungsbereich 732 supported by the Deutsche Forschungsgemeinschaft (DFG).