Implementing and Documenting
Large-Scale Grammars –
German LFG

Contents

Acknowledgements

Abstract

Implementing and Documenting Large-Scale Grammars — German LFG

Introduction

This work addresses the implementation and documentation of large-scale grammars. The theoretical considerations are exemplified by a German LFG grammar.

Research in the field of grammar development focuses on grammar modularization, ambiguity management, robustness, testing and evaluation, maintainability and reusability. A point which has often been neglected is the detailed documentation of large-scale grammars. One of the aims of this work is to fill this gap.

We start by looking at the structure of grammars of natural languages. We show that the concept of modules, as known from software engineering, does not correspond directly to grammar modules. The grammar-specific properties put special constraints on documentation. We develop an XML-based documentation technique that allows us to accomodate these constraints.

In the second part of the dissertation, the documentation technique is exemplified by the detailed documentation of our implementation of the German DP (determiner phrase).

Basics

Lexical-Functional Grammar (LFG) is a constraint-based linguistic theory (Falk 2001, Bresnan 2001, Dalrymple 2001). It defines different levels of representation to encode syntactic, semantic and other information.

The levels that are relevant here are constituent structure (c-structure) and functional structure (f-structure). The level of c-structure represents the constituents of a sentence and the order of the terminals. The level of f-structure

encodes the functions of the constituents (e.g. subject, adjunct) and morpho-syntactic information, such as case, number, and tense.

The c-structure of a sentence is determined by a context-free phrase structure grammar and is represented by a tree. In contrast, the f-structure is represented by a matrix of attribute-value pairs. The analysis of the sentence in (1) illustrates both representation levels. (We assume a CP-analysis of German, cf. Berman 2001.)

(1) *Sie kommt.*
     she comes
     'She is coming.'

```
            CS 1:     ROOT
                     /\
              CProot[std] PERIOD
                  |          |
             DP[std]  Cbar   .
                |      |
           DPx[std]  V[v,fin]
                |      |
           PRON[std] Vx[v,fin]
                |      |
               sie    kommt
```

```
"Sie kommt."

     ┌PRED    'kommen<[37:pro]>'                                        ┐
     │        ┌PRED  'pro'                                            ┐ │
     │SUBJ    │NTYPE [NSYN pronoun]                                  │ │
     │     37 └CASE nom, GEND fem, NUM sg, PERS 3, PRON-FORM sie, PRON-TYPE pers┘ │
     │        ┌_PASSIVE -                              ┐              │
     │CHECK   │_VLEX     [_AUX-SELECT sein]            │              │
     │        └_VMORPH  [_PART-VERB -]                 ┘              │
     │TNS-ASP [MOOD indicative, TENSE present]                        │
     │TOPIC   [37:pro]                                                │
     │     69 CLAUSE-TYPE declarative, STMT-TYPE declarative, VTYPE main │
     └                                                                ┘
```

The implementation of the LFG grammar is based on the Xerox Linguistic Environment (XLE, cf. Butt *et al.* 1999, ch. 11). XLE is a grammar development platform for large-scale grammars. It supports the integration of external modules, such as tokenizers, finite-state morphologies, and subcategorization lexicons. Moreover, XLE provides various means of abstraction, e.g. macros or templates, which allow for a modular grammar implementation.

The following figure illustrates the preprocessing steps for the input string *Sie kommt.* ('She is coming.'), before it is input to the actual grammar component. The tokenizer splits the input string into a sequence of delimited tokens. For instance, the full stop is analyzed as a single token. In addition, the first word of the sentence is optionally decapitalized. The output of the tokenizer represents the input to the morphological analysis. The morphological analyzer maps each token to a sequence of morphological tags, encoding the lemma form, the part of speech, and inflectional information of the token. These tags are the input to the grammar. (The spaces in the output of the tokenizer and morphology should be read as token boundaries. The set members that figure in the successful analysis are set in italics.)

```
                    Sie kommt.
                        |
                 ┌─────────────┐
                 │  TOKENIZER  │
                 └─────────────┘
                        ↓
          { Sie | sie } kommt { . | , . }
                        |
                 ┌──────────────┐
                 │  MORPHOLOGY  │
                 └──────────────┘
                        ↓
   {  sie ^CAP +PPRO .Pers .3 .Pl .MFN .NA .None
   |  sie +PPRO .Pers .3 .Pl .MFN .NA .None
   |  sie +PPRO .Pers .3 .Sg .Fem .NA .None }

      {  kommen +V .Imp .Pl
      |  kommen +V .3 .Sg .Pres .Ind
      |  kommen +V .2 .Pl .Pres .Ind }

      {  . +PUNCT .Sent
      |  , +PUNCT .Comma . +PUNCT .Sent }
                        |
                 ┌─────────────┐
                 │  GRAMMAR    │
                 └─────────────┘
```
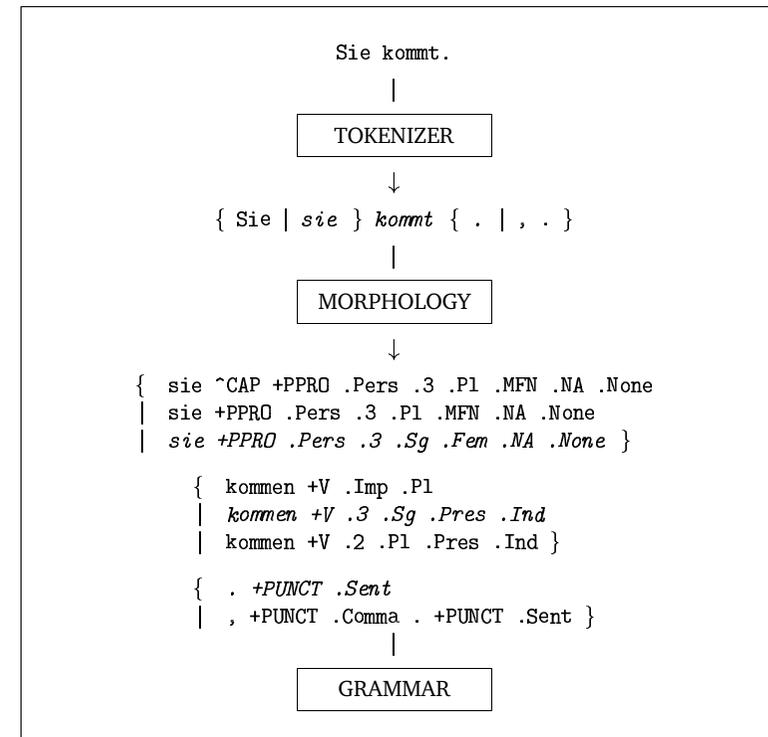
Grammar-Specific Properties

Grammar development can be seen as the development of a special kind of software, namely the grammar. As a large software project, grammar development should adhere to the techniques and design principles that are known from software engineering, such as modularity.

However, we argue that the implementation of a grammar differs from standard software development in two important aspects.

Relevance of the grammar code Usually, details of an implementation are only relevant to software developers. Users of the software are only interested in the input-output behaviour of the software, i.e. the software represents a "black box" to them.

In contrast, a grammar implementation represents important information in itself in that it encodes formalizations of linguistic phenomena. As a consequence, people other than grammar developers show interest in the actual code.

Modules and black boxes Both ordinary software and grammar implementations make use of modules as means of abstraction.

Modules in ordinary software assemble pieces of code that are functionally related. Such modules usually represent "black boxes". That is, the input and output of each module (i.e. the interfaces between the modules) are clearly defined, while the module-internal routines that map the input to the output are invisible to other modules. A modular design of the code supports the transparency and maintainability of the code. First, irrelevant details of the implementation can be hidden in a module, i.e. the code is not obscured by too many details. Second, if a certain functionality of the software is to be modified, the software developer ideally only has to modify the code within the module encoding that functionality.

Modules in grammar implementations similarly assemble pieces of code that are functionally related: they do this by encoding linguistic generalizations.

- For instance, each of the projection levels of LFG (c-structure, f-structure) represents a module.

- All X′-projections of a syntactic category (e.g. DP, Dbar, D) represent a module.

- Each macro and template represents a module by encoding common properties.

An example template is @PPfunc_desig, which is used by the closely related annotations of PPs in different positions, e.g. the annotations of PPs dominated by CP and by VP.

```
PPfunc_desig(_desig) = {      "non-semantic OBLtheta"
                              (_desig OBL) = ↓
                              (↓ PTYPE) = nosem

                       |      "semantic OBLtheta"
                       {      (_desig OBL-AG) = ↓
                       |      (_desig OBL-LOC) = ↓
                       |      (_desig OBL-DIR) = ↓
                       |      (_desig OBL-MANNER) = ↓
                       }
                              (↓ PTYPE) = sem
                   }.

CP →  ...
      PP: @(PPfunc_desig (↑{ COMP | XCOMP }*) )
      ...

VP →  ...
      PP: @(PPfunc_desig (↑XCOMP*) )
      ...
```

Code transparency The above example shows that templates can be used to encode common properties within a module. Put differently, templates encode generalizations.

In this way, the intentions of the grammar writer are encoded explicitly. It is not by accident that the PPs within the CP and VP are annotated by almost identical annotations. In this sense, the use of templates furthers code transparency. At the same time, templates help guarantee code maintainability. If the analysis of the PP functions is to be modified, only one template, @PPfunc_desig, has to be adjusted.

On the other hand, the functionality of the CP and VP rules cannot be understood without the definitions of the template @PPfunc_desig. In this sense, one might say that the use of templates hinders code transparency.

To sum up, templates and macros are modules since they encode functional units (linguistic generalizations). Contrary to software modules, they are not black boxes because their internal structure/content is relevant to the outside, i.e. the rule calling the template/macro.

In addition, grammar modules are often hierarchically structured, i.e. one module calls another module, which again calls another one, etc.

These properties of grammar modules have an impact on the documentation of grammars.

Restrictions on grammar documentation   The grammar-specific properties put constraints on (i) the content and (ii) the structure of the documentation.

(i) The grammar code by itself represents important information in that it encodes linguistic analyses. Therefore, large parts of grammar documentation consist of highly detailed code-level documentation. (Such a type of documentation usually serves the software developer rather than the software user.)

(ii) The content/definition of certain dependent modules (such as templates, macros) is relevant to the understanding of the functionality of the mother rule. Hence, the content of dependent modules must be accessible in some way within the documentation of the mother rule.

One way of encoding such dependencies is by means of links. Within the documentation of the mother rule, a pointer would point to the documentation of the macros/templates that are called by this rule. The reader of the documentation would simply follow these links (which might be realized by hyperlinks).

However, a typical grammar rule calls many macros and templates, and macros often call other macros and templates. This hierarchical structure makes the reading of link-based documentation troublesome since the reader has to follow all the links to understand the functionality of the top-most module.

We therefore conclude that the structure of the documentation should be independent of the structure of the grammar.

XML-Based Documentation

To fulfil the above constraints, we propose an XML-based documentation (XML: eXtensible Markup Language). That is, the source documentation and the grammar code are enriched by XML markup. XSLT stylesheets operate on this markup to generate the actual documentation (in our case, a LaTeX document, which is further processed to result in a postscript or PDF file) (XSLT: eXtensible Style Language for Transformations).

The XML tags are used to link and join the documentation text and the grammar code. In this way, the documentation is independent of the structure of the code.

The following figure presents the generation of the output documentation. The XML markup is added to the source grammar via Perl processing. Specific

XML tags within the documentation refer to tags within the grammar code. The XSLT processing copies the referenced parts of the code to the output documentation.
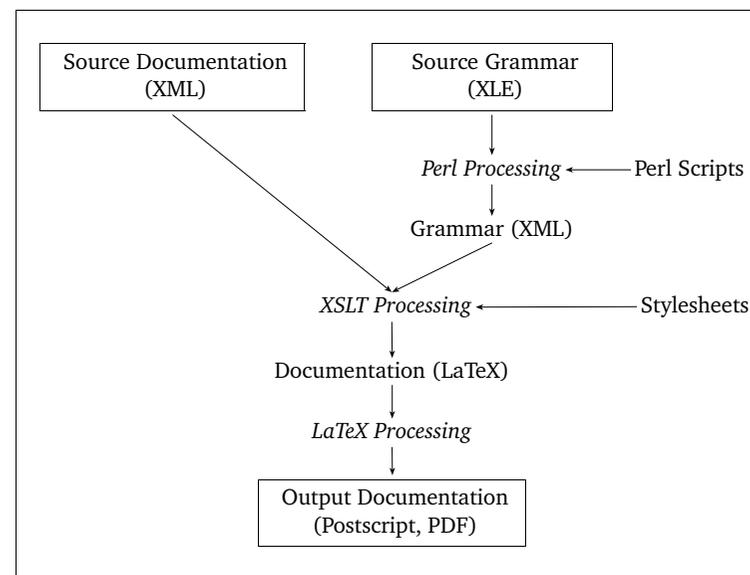
Besides providing a flexible method for documenting grammars, the XML tags can be used to support grammar development in various aspects. For instance, a testsuite can be generated on the basis of all the example sentences in the documentation. This allows the grammar writer to check all phenomena described in the documentation against the actual coverage of the grammar implementation.

Example of a Documentation and Implementation

The second part of the dissertation represents an application of the above described documentation technique: documentation of the implementation of the German DP in the German Pargram grammar. The documentation starts with a linguistic introduction, presenting the relevant phenomena. A detailed description of the implementation follows. The phenomena are illustrated by various examples, including snapshots of the analyses.

Finally, corpus data, indices of DP features, OT marks, etc., and the complete code of the documented DP grammar are presented by appendices.

Zusammenfassung (Abstract in German)

Implementation und Dokumentation großer Grammatiken — deutsche LFG

Einführung

Die vorliegende Arbeit handelt von der Implementierung und Dokumentation von Grammatiken mit großer Abdeckung. Die theoretischen Überlegungen werden am Beispiel einer deutschen LFG-Grammatik umgesetzt.

Das Forschungsinteresse im Bereich der Grammatik-Implementierung richtet sich v.a. auf Aspekte wie Grammatik-Modularisierung, Ambiguitätsmanagement, Robustheit, Testen und Evaluieren, Wartbarkeit und Wiederverwendbarkeit. Ein Aspekt, der oft vernachlässigt wird, ist das detaillierte Dokumentieren großer Grammatiken. Ein Ziel dieser Arbeit ist es, diese Lücke zu füllen.

Dazu gehen wir zunächst auf die Struktur formaler Grammatiken zur Verarbeitung natürlicher Sprache ein. Wir zeigen, dass sich das Konzept von Modulen nicht eins-zu-eins von "normaler" Software auf Grammatik-Implementationen übertragen lässt. Die Grammatik-spezifischen Eigenschaften der Module wiederum stellen besondere Anforderungen an eine Dokumentation. Wir entwickeln daher eine XML-basierte Dokumentationsmethodik, die diesen Anforderungen genügt.

Diese Methodik findet Anwendung im zweiten Teil der Arbeit, die aus einer detaillierten Dokumentation unserer Implementation der deutschen DP besteht.
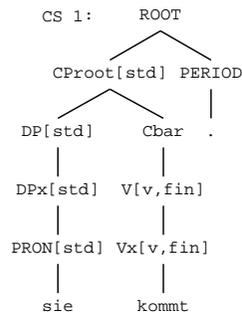
Grundlagen

Lexical-Functional Grammar (LFG) ist eine Constraint-basierte linguistische Theorie (Falk 2001, Bresnan 2001, Dalrymple 2001). Sie unterscheidet ver-

schiedene Repräsentationsebenen zur Kodierung von syntaktischer, semantischer und anderer Information.

Die für diese Arbeit relevanten Ebenen sind die Konstituentenstruktur (c-Struktur) und die funktionale Struktur (f-Struktur). Auf c-Struktur-Ebene werden die Konstituenten eines Satzes und Wortstellungseigenschaften repräsentiert. Die f-Struktur bildet die Funktionen der Konstituenten ab (z.B. Subjekt, Adjunkt) und kodiert außerdem morpho-syntaktische Informationen wie Kasus oder Numerus.

Die c-Struktur eines Satzes bestimmt sich durch eine kontextfreie Phrasenstrukturgrammatik und wird als Baum dargestellt. Dagegen wird die f-Struktur in Form einer Matrix von Attribut-Werte-Paaren dargestellt. Die Analyse des Satzes *Sie kommt.* illustriert die beiden syntaktischen Repräsentationsebenen. (Wir gehen von einer CP-Analyse des Deutschen aus, vgl. Berman 2001.)

```
CS 1:      ROOT
            |
         CProot[std] PERIOD
          |            |
      DP[std]   Cbar   .
        |        |
    DPx[std]  V[v,fin]
        |        |
   PRON[std]  Vx[v,fin]
        |        |
       sie     kommt
```

```
"Sie kommt."

 ┌PRED     'kommen<[37:pro]>'                                              ┐
 │         ┌PRED  'pro'                                                  ┐ │
 │SUBJ     │NTYPE [NSYN pronoun]                                        │ │
 │      37 │CASE nom, GEND fem, NUM sg, PERS 3, PRON-FORM sie, PRON-TYPE pers│ │
 │         ┌_PASSIVE -                         ┐                          │
 │CHECK    │_VLEX     [_AUX-SELECT sein]        │                          │
 │         └_VMORPH   [_PART-VERB -]            ┘                          │
 │TNS-ASP [MOOD indicative, TENSE present]                                │
 │TOPIC   [37:pro]                                                        │
 │69 CLAUSE-TYPE declarative, STMT-TYPE declarative, VTYPE main           │
 └                                                                        ┘
```

Die Implementation der LFG-Grammatik basiert auf dem Xerox Linguistic Environment (XLE, vgl. Butt *et al.* 1999, ch. 11). XLE ist eine Grammatikentwicklungsumgebung für große LFG-Grammatiken. XLE unterstützt die Einbindung externer Module wie Tokenizer, Morphologie und Subkategorisierunglexika. Außerdem stellt XLE Abstraktionsmittel verschiedener Art zur Verfügung (Makros, Templates), die eine modulare Implementation ermöglichen.

Das folgende Schaubild illustriert die Vorverarbeitung eines Inputstrings (*Sie kommt*), die die Eingabestruktur für die eigentliche Grammatikkomponente erzeugt. Die Tokenizerkomponente teilt den String in einzelne Tokens, z.B. wird der Punkt am Satzende als eigenes Token behandelt. Außerdem wird das erste Wort im Satz optional kleingeschrieben. Die Ausgabe des Tokenizers stellt die Eingabe für die Morphologie-Komponente dar. Diese bildet jedes Token auf eine Sequenz von Morphologie-Tags ab, die Information über die Zitierform, die Wortart und die Flexion des Tokens kodieren. Diese Tagsequenz bildet dann die Eingabe für die Grammatik. (Die Leerzeichen in der Ausgabe von Tokenizer und Morphologie stellen die Tokengrenzen dar. Die Disjunkte, die an der erfolgreichen Analyse teilhaben, sind kursiv gesetzt.)

```
                    Sie kommt.
                        |
                 ┌─────────────┐
                 │  TOKENIZER  │
                 └─────────────┘
                        ↓
        { Sie | sie } kommt { . | , . }
                        ↓
                ┌───────────────┐
                │  MORPHOLOGIE  │
                └───────────────┘
                        ↓
    { sie ^CAP +PPRO .Pers .3 .Pl .MFN .NA .None
    | sie +PPRO .Pers .3 .Pl .MFN .NA .None
    | sie +PPRO .Pers .3 .Sg .Fem .NA .None }

        { kommen +V .Imp .Pl
        | kommen +V .3 .Sg .Pres .Ind
        | kommen +V .2 .Pl .Pres .Ind }

        { . +PUNCT .Sent
        | , +PUNCT .Comma . +PUNCT .Sent }
                        |
                 ┌──────────────┐
                 │  GRAMMATIK   │
                 └──────────────┘
```
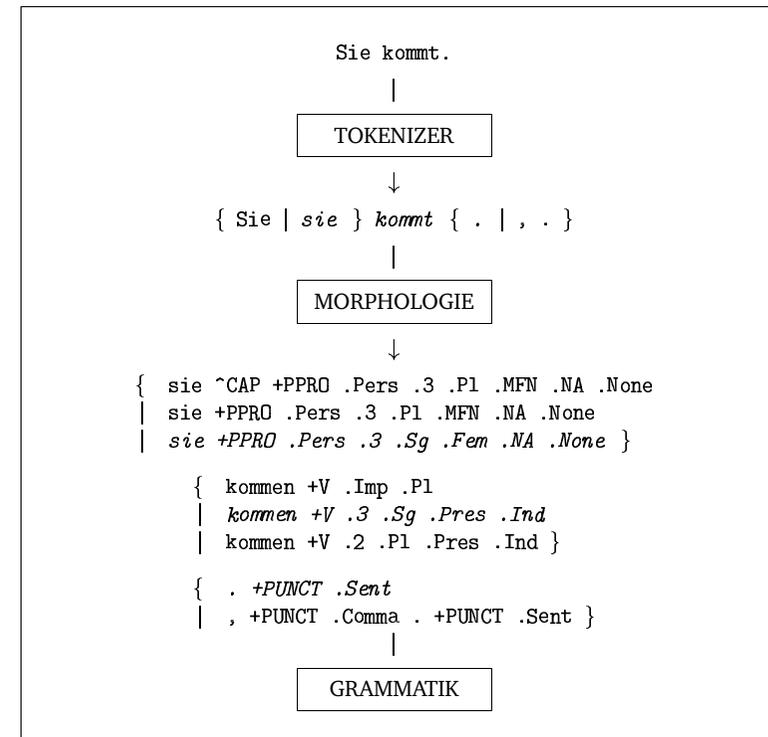
Grammatik-spezifische Eigenschaften

Grammatikentwicklung kann als die Entwicklung einer speziellen Art von Software (nämlich einer Grammatik) angesehen werden. Als großes Softwareprojekt unterliegt die Grammatikentwicklung damit den Prinzipien der Softwaretechnik, wie z.B. Modularität.

Wir argumentieren aber, dass sich die Implementation einer Grammatik in zwei wesentlichen Punkten von "typischer" Software unterscheidet.

Relevanz des Codes   Der Code normaler Software ist nur für den Entwickler relevant, nicht aber für den Nutzer dieser Software. Für den Nutzer genügt es, Input und Output der Software zu kennen, d.h. für ihn stellt die Software eine "Black Box" dar.

Im Gegensatz dazu stellt der Code einer Grammatik-Implementierung selbst wichtige Information dar. D.h. neben dem Grammatik-Entwickler interessieren sich z.B. Linguisten für die Implementierung und Formalisierung bestimmter sprachlicher Konstruktionen.

Module und Black Boxes   Normale Software wie auch Grammatik-Implementierungen benutzen Module als Abstraktionsmittel.

Module in normaler Software bündeln funktional zusammengehörigen Code. Sie stellen typischerweise "Black Boxes" dar. D.h. nach außen (für die anderen Module) sind nur Input und Output eines jeden Modules sichtbar, während Modul-interne Routinen verborgen bleiben. Dadurch wird der Code transparenter und leichter wartbar: Zum einen können so irrelevante Details der Implementierung versteckt werden, und zum zweiten braucht bei Modifikation einer bestimmten Funktionalität nur das betreffende Modul verändert werden, ohne dass Anpassungen in anderen Modulen nötig würden.

Module in Grammatik-Implementierungen bündeln ebenfalls funktional zusammengehörigen Code, indem sie nämlich linguistische Generalisierungen kodieren.

- Beispielsweise bilden die verschiedenen Projektionsebenen (c-Struktur, f-Struktur) jeweils ein Modul.

- Alle X′-Projektionen einer syntaktischen Kategorie (z.B. DP, Dbar, D) bilden ein Modul.

- Makros und Templates bilden jeweils Module, indem sie gemeinsame Eigenschaften gebündelt kodieren.

  Ein Beispiel-Template ist @PPfunc_desig, das für die weitgehend parallele Annotation von PPs in verschiedenen Funktionen benutzt wird, z.B. in der

Annotation von PPs im Vorfeld (dominiert von CP) wie auch im Mittelfeld (dominiert von VP).

```
PPfunc_desig(_desig) = {        "non-semantic OBLtheta"
                                (_desig OBL) = ↓
                                (↓ PTYPE) = nosem

                        |       "semantic OBLtheta"
                              { (_desig OBL-AG) = ↓
                              | (_desig OBL-LOC) = ↓
                              | (_desig OBL-DIR) = ↓
                              | (_desig OBL-MANNER) = ↓
                              }
                                (↓ PTYPE) = sem
                      }.

CP → ...
      PP: @(PPfunc_desig (↑ { COMP | XCOMP }*) )
      ...

VP → ...
      PP: @(PPfunc_desig (↑XCOMP*) )
      ...
```

Transparenz des Codes   Das obige Beispiel zeigt, dass Templates benutzt dazu werden können, gemeinsame Eigenschaften in einem Modul zu kodieren, in anderen Worten: Generalisierungen explizit zu kodieren.

Damit wird auch die Intention des Grammatik-Entwicklers explizit kodiert: Es kann dann kein Zufall sein, dass die PPs innerhalb der CP und der VP weitgehend gleich annotiert sind. In diesem Sinn wird der Code durch den Einsatz von Templates transparenter. Gleichzeitig garantiert die Verwendung von Templates, dass der Code wartbar bleibt: Soll die Analyse der PP-Funktionen geändert werden, ist davon nur das Template @PPfunc_desig betroffen.

Andererseits ist klar, dass die CP- und VP-Regeln ohne die Definition des Templates @PPfunc_desig nicht zu verstehen sind. In diesem Sinn kann man sagen, dass der Einsatz von Templates den Code weniger transparent macht.

Zusammenfassend kann man sagen, dass Templates und Makros Module darstellen, da sie funktionale Einheiten (Generalisierungen) kodieren. Allerdings stellen diese Module keine Black Boxes dar, da ihr "Innenleben" relevant ist, um die Funktionsweise der aufrufenden Regeln zu verstehen.

Zusätzlich gilt, dass Grammatik-Module sehr oft hierarchisch angeordnet

sind, d.h. ein Modul ruft das nächste auf, das wiederum ein weiteres Modul aufruft etc.

Diese Eigenschaften von Grammatik-Modulen haben Einfluss auf die Dokumentation von Grammatiken.

Anforderungen an eine Grammatikdokumentation Aus dem oben Genannten ergeben sich Anforderungen (i) an den Inhalt und (ii) an die Struktur der Dokumentation.

(i) Der Code einer Grammatik-Implementierung stellt selbst wichtige Information dar. Daher muss eine Grammatikdokumentation Code-nah und detailliert sein. (Ein solcher Typ von Dokumentation dient normalerweise dem Entwickler der Software, nicht dem Nutzer.)

(ii) Der Inhalt bestimmter abhängiger Module (wie Templates, Makros) ist wichtig, um die Funktionsweise der aufrufenden Regel zu verstehen. Daher muss bei der Dokumentation einer solchen Regel der Inhalt der abhängigen Module leicht zugänglich/verfügbar sein.

Eine Möglichkeit, solche Abhängingkeiten zu kodieren, ist mit Hilfe von Verweisen (Links). Innerhalb der Dokumentation einer Regel würden Zeiger auf die Dokumentation der Makros und Templates verweisen, die von dieser Regel aufgerufen werden. Der Leser der Dokumentation müsste dann nur diesen Verweisen folgen (die z.B. als Hyperlinks realisiert sein könnten).

Allerdings ruft eine typische Grammatikregel viele Makros und Templates auf, und Makros rufen weitere Makros und Templates auf. Diese hierarchische Struktur erschwert das Lesen einer Link-basierten Dokumentation. Der Leser müsste alle Links verfolgen, um die Funktionsweise der obersten Regel zu verstehen.

Wir schließen daraus, dass die Struktur der Dokumentation unabhängig von der Struktur der Grammatik sein sollte.

XML-basierte Dokumentation

Um die oben genannten Anforderungen zu erfüllen, schlagen wir eine XML-basierte Dokumentation vor (XML: eXtensible Markup Language). D.h. die Originaldokumentation und der Grammatikcode werden mit XML-Markup angereichert, auf dem XSLT-Stylesheets operieren, um die eigentliche Dokumentation zu generieren (in unserem Fall: ein LaTeX-Dokument, aus dem eine Postscript- oder PDF-Datei erzeugt wird) (XSLT: eXtensible Style Language for Transformations).

Die XML-Tags werden genutzt, um den Textteil und den Grammatikcode miteinander zu verlinken und zusammen zu führen. Auf diese Weise ist die Dokumentation unabhängig von der Struktur des zu dokumentierenden Codes.

Das folgende Schaubild stellt die Generierung der Dokumentation dar. Das XML-Markup für den Grammatikcode wird mit Hilfe von Perl-Scripts erstellt. Spezielle XML-Tags innerhalb der Dokumentation verweisen auf Tags im Grammatikcode. Bei der XSLT-Prozessierung werden entsprechende Fragmente des Grammatikcodes in die Output-Dokumentation kopiert.
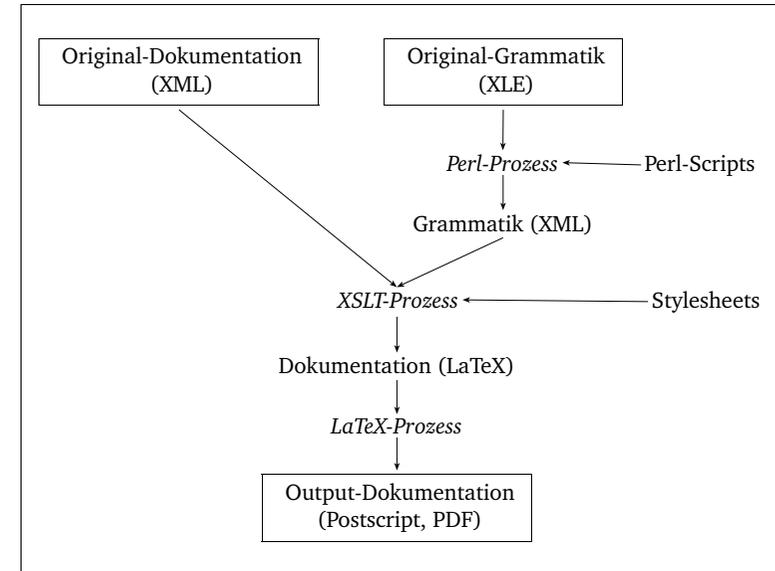
Neben einer flexiblen Art der Dokumentation können die XML-Tags die Grammatikentwicklung in vielerlei Hinsicht unterstützen. Z.B. kann aus allen Beispielsätzen der Dokumentation eine Testsuite generiert werden. Damit kann gewährleistet werden, dass die Grammatik die in der Dokumentation beschriebenen Phänomene abdeckt.

Beispiel einer Dokumentation und Implementation

Der zweite Teil der Dissertation besteht aus einer Anwendung der oben beschriebenen Dokumentationsmethode. Die Dokumentation der DP-Implementation der deutschen Pargram-Grammatik erstreckt sich über mehrere Kapitel. Nach einer linguistischen Einführung in die relevanten Phänomene folgt eine detaillierte Beschreibung der implementierten Analyse. Die Phänomene werden durch zahlreiche Beispiele inklusive Snapshots der Analysen illustriert.

In verschiedenen Anhängen finden sich Korpusbelege, Indizes der DP Features, OT Marks etc. sowie der vollständige Code der dokumentierten DP-Grammatik.

Part I

Basics

Chapter 1

Introduction

In computational linguistics, an important point of interest is grammar development, i.e. the implementation of grammars.

One line of research goes into the development of grammar-specific development platforms. Such grammar development environments define specific programming languages that mirror linguistic formalisms, such as LFG, HPSG. Moreover, they also contain various tools that support the task of grammar writing. A grammar which is formulated in the appropriate programming language can be used as a parser (or generator) within the development environment. That is, the user inputs a string, and the system outputs a linguistic analysis of this string, according to the rules defined by the grammar.

Examples of grammar development environments are:

- LKB (Linguistic Knowledge Building system, URL: `http://www-csli.stanford.edu/~aac/lkb.html`), described in Copestake (2002). LKB supports the formalism of HPSG (Head-Driven Phrase Structure Grammar);

- XTAG (URL: `http://www.cis.upenn.edu/~xtag/`, cf. Doran *et al.* 1994), supporting TAG (Tree Adjoining Grammar);

- XLE (Xerox Linguistic Environment, URL: `http://www.parc.com/istl/groups/nltt/xle/`), described in Butt *et al.* (1999, ch. 11). XLE supports LFG (Lexical-Functional Grammar).

A list of further systems can be found, e.g., in Copestake (2002, ch. 5.8).

A second line of research goes into the development of grammars. The above environments have been used in the development of large-scale grammars for English and other languages. Such grammars can be used to verify theoretic assumptions about natural language. Furthermore, many applications in computational linguistics depend on an exact syntactic analysis of language, e.g. machine translation or meaning construction.

Research topics in grammar development include grammar modularization, ambiguity management, robustness, testing and evaluation, maintainability and reusability.

A point which has often been neglected is the detailed documentation of large-scale grammars. One of the aims of this work is to fill this gap.

To illustrate the needs of documentation, we will refer to a large-scale LFG grammar for German. (Our argumentation, however, applies not only to grammars in the LFG formalism but to any grammar that is modularized to a certain extent, see below.) The grammar has been developed in the Pargram project at the IMS (Institut für maschinelle Sprachverarbeitung), University of Stuttgart.

Pargram is a joint project of researchers from PARC (Palo Alto Research Center, California), XRCE (Xerox Research Centre Europe, Grenoble; project partner until 2001), Fuji-Xerox (Tokyo, Japan), the University of Bergen (Norway), and the IMS (Butt *et al.* 2002, Butt *et al.* 1999, URL: `http://www.parc.com/istl/groups/nltt/pargram/`). Pargram aims at the implementation of large-scale LFG grammars of different languages that adhere to certain common assumptions, i.e. the grammars are implemented in a "parallel" way (hence the name Pargram: "parallel grammars"). The grammar implementations make use of the above mentioned Xerox Linguistic Environment (XLE).

The current German LFG grammar is the product of several years of project work, involving a number of people (in particular, Miriam Butt, Stefanie Dipper, Jonas Kuhn, Christian Rohrer). The specific work reported in this dissertation involved a major revision of large parts of the code.

We will argue that maintainability of a (manually encoded) large-scale grammar depends to a large extent on the modularization of the grammar rules. A large-scale grammar remains maintainable only if linguistic generalizations are encoded explicitly, e.g. by means of abstraction such as macros. Macros represent one type of modules in a grammar.

We will show that grammar modularization has an impact on grammar documentation. Grammar modules exhibit specific properties that put constraints on the form of the documentation. Similar to common grammar-specific tools that are provided by grammar development environments, we propose a grammar-specific documentation technique (which ought to be integrated into the grammar development environments, as also suggested by Erbach and Uszkoreit 1990, Erbach 1992). Moreover, we illustrate this documentation technique by highly detailed documentation chapters for the DP of the German LFG grammar.

The focus on highly detailed implementational aspects distinguishes our DP documentation from other work reporting on grammar implementations, such as Butt *et al.* (1999), Müller (1999). These authors focus on high-level design principles, such as the underlying concepts of the implemented analyses. In contrast, we focus on the actual implementation of these concepts. (Similarly, the

documentation presented by Berman and Frank 1996 includes implementation details of the documented fragments of German and French.)

Simultaneously, our DP analysis may serve as an example of a grammar that is modularized to a large extent. In addition, the DP analysis illustrates that a grammar implementation always represents a compromise between conflicting demands. Linguistic theory, coverage, overgeneration/ambiguity rate, and performance issues often require opposite strategies in grammar development, as visualized by the picture below. For instance, extending the coverage of the grammar typically results in an increased ambiguity rate and a decrease of efficiency.

Linguistic Theory ⟷ Coverage

Ambiguity Rate ⟷ Efficiency
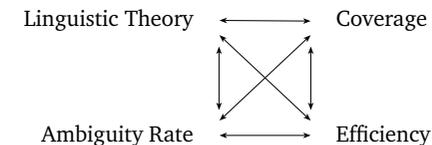
Our implementation of the German grammar is based on linguistic insights, corpus frequencies, and, to a minor degree, on performance considerations. In an evaluation as of January 2000, the grammar covered 35% of free newspaper text (Dipper 2000).

Overview   Part I of this dissertation contains an introduction to LFG and XLE, the grammar development platform we make use of. It presents all formal devices that are necessary for the understanding of the DP documentation in Part III.

Part II discusses aspects of grammar implementation that distinguish a grammar from ordinary software. We show that these aspects have an impact on the form of the grammar documentation. This concerns both the content and structure of the documentation. We propose a technique of grammar documentation that is based on the standardized document exchange format XML (eXtensible Markup Language). Our approach allows for a user-friendly documentation and, furthermore, can be exploited to support the process of grammar development.

Part III of the dissertation presents highly detailed documentation chapters for the German DP, generated by the proposed technique. The documentation comprises a linguistic introduction, followed by the documentation of the grammar code.

Part IV concludes the dissertation with a summary of the main results and directions for future research.

(Note that the source files of this dissertation consist of XML documents, which are automatically converted to LaTeX—in the same way as the documentation chapters are generated. As a consequence, the layout is restricted in certain aspects, e.g. tables and figures may not float.)

Chapter 2

Introduction to LFG

Contents

This chapter is a short introduction to Lexical-Functional Grammar (LFG). It is addressed to readers without prior knowledge of LFG. The goal of the introduction is to enable these people to "read" the parts of grammar code that figure in the documentation chapters. The focus therefore lies on conveying an intuitive access to the basic concepts of LFG rather than presenting formal definitions. Furthermore, LFG principles that are not applied in our implementation are left out in the introduction (e.g. binding theory, traces, semantic projection).

We illustrate the concepts and principles using examples of English. German examples are given in the following chapters. For profound introductions to LFG, the reader is referred to the following books: Falk (2001), Bresnan (2001), Dalrymple (2001), Dalrymple *et al.* (1995).

LFG is a grammar formalism which makes use of two representation levels to encode syntactic properties of sentences. The first section presents the basic properties of both levels (sec. 2.1). The topic of the second section is the link between these levels (sec. 2.2). Finally, additional restrictions are presented (sec. 2.3).

2.1   C-Structure and F-Structure

A prominent feature of LFG is the clear distinction between form and function of constituents within a sentence. An LFG analysis of a sentence usually consists of two parts, the constituent structure (c-structure) and the functional structure (f-structure).

C-structure   C-structure is a tree representation which encodes the constituents and word order properties of the sentence. It is generated by a generalized context-free phrase structure grammar. The right-hand side of a context-free rule in LFG consists of a regular expression. As an example, cf. the following rule.

```
NP → (D) N PP*
```

According to this rule, an NP can expand into an optional determiner (D), followed by an obligatory noun (N). Finally, arbitrarily many PPs (including none) can follow the noun (the so-called Kleene-star '*' denotes zero or more repetitions of the constituent marked by '*').

For a more complex example, cf. the c-structure analysis of the sentence *Mary never reads books*, which can be generated by the following rules.

```
S  → NP VP          V  → reads
VP → ADV VP         N  → Mary
VP → V (NP)         N  → books
NP → N              ADV → never
```
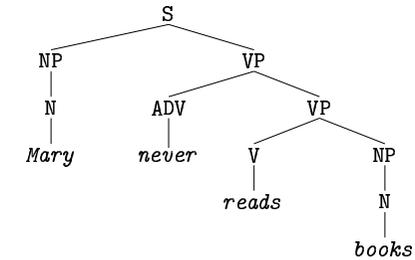

F-structure   F-structure encodes the syntactic/grammatical functions of the constituents, such as subject, object, adjunct (= modifier), etc. In addition, morpho-syntactic properties of the constituents, e.g. tense, mood, case, number, are represented in f-structure. In contrast to c-structure, f-structure is represented in terms of an attribute-value matrix. (The derivation of f-structures is described in the next section.)

A simple f-structure, containing the attribute (or "feature") CASE with the atomic value nom (nominative), is the following f-structure (in contrast to LFG tradition, we represent atomic values by lower case letters). An f-structure is always enclosed in brackets '[…]'.

$$\begin{bmatrix} \text{CASE} & \text{nom} \end{bmatrix}$$

F-structures typically contain more than one feature. (Note that we use the term "feature" either to refer to an attribute or to an attribute-value pair.) For instance, consider the following f-structure.

$$\begin{bmatrix} \text{PRED} & \text{'Mary'} \\ \text{CASE} & \text{nom} \end{bmatrix}$$

The feature PRED (predicate) is special in that its value is always quoted ('Mary'). The value is called a "semantic form". It can be thought to represent the semantic content of the corresponding word, and may serve as the input to a semantic analysis of the sentence. Subcategorization requirements are also encoded in the PRED feature. For instance, a verb like *read* subcategorizes for

two arguments, subject and object. The corresponding f-structure representation looks as follows.

$$\left[\begin{array}{ll} \text{PRED} & \text{`read<SUBJ,OBJ>'} \end{array}\right]$$

Semantic (so-called "thematic") arguments are enclosed in angle brackets '<...>' within the PRED value. Non-semantic arguments are listed outside the angle brackets, cf. the f-structure of a verb like *rain* which subcategorizes for an expletive subject.

$$\left[\begin{array}{ll} \text{PRED} & \text{`rain<>SUBJ'} \end{array}\right]$$

The value of a feature can be complex, i.e., it can consist of another f-structure. In the following example, the above f-structure featuring 'Mary' serves as the value of the feature SUBJ.

$$\left[\begin{array}{ll} \text{SUBJ} & \left[\begin{array}{ll} \text{PRED} & \text{`Mary'} \\ \text{CASE} & \text{nom} \end{array}\right] \end{array}\right]$$

Further examples of features with complex values are OBJ and ADJUNCT. Usually, these features are called "functions".

As a more complex example, consider the f-structure for *Mary never reads books* (cf. the c-structure example above).

$$\left[\begin{array}{ll} \text{PRED} & \text{`read<SUBJ,OBJ>'} \\ \text{SUBJ} & \left[\begin{array}{ll} \text{PRED} & \text{`Mary'} \\ \text{CASE} & \text{nom} \\ \text{NUM} & \text{sg} \\ \text{PERS} & \text{3} \end{array}\right] \\ \text{OBJ} & \left[\begin{array}{ll} \text{PRED} & \text{`book'} \\ \text{CASE} & \text{acc} \\ \text{NUM} & \text{pl} \\ \text{PERS} & \text{3} \end{array}\right] \\ \text{ADJUNCT} & \left\{\left[\begin{array}{ll} \text{PRED} & \text{`never'} \end{array}\right]\right\} \\ \text{TENSE} & \text{present} \end{array}\right]$$

The f-structure can be read as follows. The main predicate of the sentence is *read* which subcategorizes for two argument functions, subject and object. *Mary* functions as the subject, *book* as the object of the sentence. Finally, the adverb *never* functions as an adjunct/modifier of the sentence.

A feature must always have a unique value. However, there may be more than one adjunct modifying a sentence, e.g. as in *Mary read books yesterday in the office*. To represent multiple adjuncts, the value of the feature ADJUNCT is a

set that may contain arbitrarily many f-structures (see the above f-structure). In contrast, there is never more than one subject or object in a sentence, hence the value of SUBJ or OBJ is a single f-structure. (However, coordinated subjects and objects are analyzed by means of sets as well, cf. Kaplan and Maxwell 1988.)

We now proceed to the atomic-valued features in the above sample f-structure. NPs such as *Mary* and *books* correspond to f-structures that typically contain the features PRED, NUM (number), PERS (person), and CASE.

Note that the f-structure corresponding to the finite verb (i.e. the outermost f-structure) does not contain the features NUM and PERS even though the verb is inflected. Instead, the inflectional features of the finite verb will be required to correspond to (agree with) the corresponding features of the subject (see below).

What the above elaborations hint at is that the interaction between c-structure and f-structure determines what a (syntactically) grammatical sentence is. For instance, ungrammatical sentences such as **you hates books* or **Mary gives* are assigned a valid c-structure analysis. It is only by taking f-structure requirements into account that these sentences are excluded. The first sentence will be rejected due to incorrect verb inflection, the second due to unsatisfied subcategorization requirements.

To understand how the rejection of such ungrammatical sentences works, we now describe how the link between c-structure and f-structure is established.

2.2   Linking C-Structure and F-Structure

In order to set c-structure and f-structure into correspondence, the c-structure rules introduced above are enriched by so-called f-structure annotations. That means, each daughter category is associated with some f-structure information. When the c-structure of a sentence is derived, the corresponding f-structure(s) is/are built in parallel, according to the annotations on the categories.

We illustrate this by an example. Consider the first of the above c-structure rules, repeated here.

```
S  →  NP  VP
```

Now, each c-structure category is associated with some f-structure information. We represent this association by pointers that link (or map) each category to some f-structure (which is not specified any further at this point).
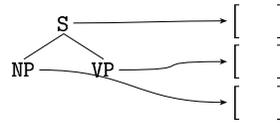
The relationship between the single f-structures is established by f-structure annotations in the rules, cf. the above c-structure rule, now annotated by f-structure constraints.

S →   NP   VP
        (↑SUBJ)=↓   ↑=↓

The arrows ↑ and ↓ refer to f-structures. The ↑-arrow refers to the f-structure of the mother node, the ↓-arrow to the f-structure of the node itself. (Whenever an arrow is followed by a feature, e.g. SUBJ, they are enclosed in parentheses, (↑SUBJ).)

The meaning of the arrows becomes more transparent if one considers the partial tree generated by this rule. In the tree, the arrows point to the node whose f-structure they refer to.
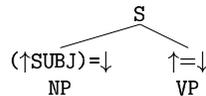
The annotations in the tree can be read as follows. We start with the annotation of the VP node: '↑=↓'. ↑ on the VP node denotes the f-structure corresponding to S, ↓ denotes the f-structure of the VP itself. Hence the annotation ↑=↓ defines that both f-structures are identical. (The technical term is "unification": the VP's f-structure and the f-structure of S unify/are unified.) We modify the above picture accordingly: the pointers of both VP and S now point to the same f-structure.

The annotation on the NP node is more complex: ' (↑SUBJ) = ↓'. It defines that the f-structure of S (denoted by ↑) contains a feature SUBJ whose value is the NP's f-structure (denoted by ↓). In other words: the NP's f-structure is embedded under the feature SUBJ which is part of the f-structure of S. We

again modify the above picture: we add the feature SUBJ to the f-structure of S (and VP) and insert the NP's f-structure as the value of SUBJ.
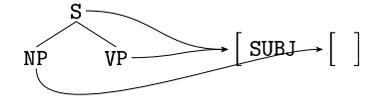
Put differently, the NP functions as the subject of S, and VP is the head of S (sharing all features).

The correspondence or mapping relation from c-structure to f-structure is called "ϕ-projection" (phi-projection). Therefore, the arrows are often labelled by ϕ in LFG presentations.

An alternative representation of the ϕ-projection makes use of f-structure variables (labels): f1, f2, etc. (this is also the representation used in XLE). Here, the c-structure categories are associated with the labels of their projected f-structures. For instance, a c-structure node may be marked by 'f1:S', which means that this node is of category S and projects the f-structure labelled by f1.

Since the f-structures of S and VP are identified, f1 and f3 label one and the same f-structure. Since the NP's f-structure (f2) is constrained to project as the SUBJ of the f-structure of S, f2 labels the f-structure embedded as the SUBJ of f1.

We now briefly present the enriched versions of the remaining c-structure rules from above (cf. p. 9). The c-structure rule deriving modifying adverbs is annotated in the following way.

VP →   ADV   VP
        ↓∈(↑ADJUNCT)   ↑=↓

That is, the ADV's f-structure will be embedded under the feature ADJUNCT which is part of the VP's f-structure. More precisely, the ADV's f-structure will be embedded under ADJUNCT as an element of a set, hence the ∈-relation.

The interpretation of the annotations of the following rules should now be obvious. (Note that in the VP rule, the annotation (↑OBJ CASE) = acc could be replaced by (↓CASE) = acc.)

```
VP  →    V           (NP)                    NP  →     N
        ↑=↓        (↑OBJ)=↓                           ↑=↓
                  (↑OBJ CASE)=acc
```

Finally, we turn to the terminal rules. Non-terminal and terminal rules differ in that non-terminal rules define structurally determined properties, whereas terminal rules define lexically determined properties.

The terminal rule for *reads* and the corresponding tree might look as follows. (Note that, since terminal nodes do not dominate any other node, the annotations associated with them typically contain ↑-arrows only.)

Terminal Rule	Tree Representation

```
                                                          V
                                                          |
                                                          |
V →        reads                             (↑PRED) =
    (↑PRED) =                                    'read<(↑SUBJ)(↑OBJ)>'
        'read<(↑SUBJ)(↑OBJ)>'                  (↑SUBJ CASE) = nom
    (↑SUBJ CASE) = nom                         (↑SUBJ NUM) = sg
    (↑SUBJ NUM) = sg                           (↑SUBJ PERS) = 3
    (↑SUBJ PERS) = 3                           (↑TENSE) = present
    (↑TENSE) = present                                 reads
```

reads imposes restrictions on its mother's f-structure (denoted by ↑), i.e. the f-structure of V. The restrictions partly concern the f-structure of V itself, partly the f-structure embedded under the function SUBJ.

- V's f-structure contains PRED and TENSE features with certain values. (Note that the subcategorized functions SUBJ and OBJ that occur in the PRED value are preceded by an ↑-arrow like all other f-structure annotations. In contrast, when the PRED feature is represented in an f-structure (rather than an f-structure annotation), the functions are no longer preceded by ↑: [PRED 'read<SUBJ,OBJ>'], since, by definition, the arrows map c-structure elements to f-structure elements. For more details, see below.)

- V's f-structure embeds the functions SUBJ and OBJ. SUBJ contains the features CASE, NUM, and PERS with specific values: nom, sg, and 3, respectively.

  Since V's f-structure is identified/unified with the f-structure of VP (according to the above VP rule), and moreover, the VP's f-structure is identified with the f-structure of S (due to the annotations in the S rule), the

restrictions introduced by *reads* are restrictions on the subject of the whole sentence. In other words, the equations associated with the inflected form *reads* determine subject-verb agreement.

This way, the ungrammatical sentence from above, **you hates books*, is excluded: The SUBJ PERS feature introduced by *hates* ([SUBJ PERS 3]) and the PERS feature introduced by *you* ([PERS 2]) are not compatible. Consider the abbreviated f-structure below, which illustrates the feature clash that marks the f-structure as invalid and the corresponding sentence as ungrammatical. (Pronouns in LFG introduce a PRED feature with the value 'pro' to indicate that their referent is to be determined by the context.)

$$
\begin{bmatrix}
\text{PRED} & \text{'hate<SUBJ,OBJ>'} \\
\text{SUBJ} & \begin{bmatrix} \text{PRED} & \text{'pro'} \\ \text{PERS} & \boxed{2/3} \end{bmatrix} \\
\text{OBJ} & \begin{bmatrix} \text{PRED} & \text{'book'} \end{bmatrix}
\end{bmatrix}
$$

Instead of terminal rules as shown above, the standard LFG notation uses lexicon entries (this is a purely notational variant). The lexicon entry of an (inflected) word such as *reads* defines the dominating c-structure category (V) and the associated f-structure constraints.

```
reads  V     (↑PRED) = 'read<(↑SUBJ)(↑OBJ)>'
             (↑SUBJ CASE) = nom
             (↑SUBJ NUM) = sg
             (↑SUBJ PERS) = 3
             (↑TENSE) = present

Mary   N     (↑PRED) = 'Mary'
             (↑NUM) = sg
             (↑PERS) = 3

books  N     (↑PRED) = 'book'
             (↑NUM) = pl
             (↑PERS) = 3

never  ADV   (↑PRED) = 'never'
```

To complete the picture, we present the full c-structure and f-structure of the example *Mary never reads books*, as they are derived by the rules and lexicon entries given in this section.

```
                    f1:S
              _____|_____
             |                 |
          f2:NP             f4:VP
             |          _____|_____
          f3:N         |             |
             |      f5:ADV         f6:VP
           Mary      never      ____|____
                       |        |       |
                     never    f7:V    f8:NP
                               |        |
                             reads    f9:N
                                        |
                                      books
```

$$
\text{f1,f4,f6,f7:}
\begin{bmatrix}
\text{PRED} & \text{‘read<(f1 SUBJ)(f1 OBJ)>’} \\
\text{SUBJ} & \text{f2,f3:}\begin{bmatrix} \text{PRED} & \text{‘Mary’} \\ \text{CASE} & \text{nom} \\ \text{NUM} & \text{sg} \\ \text{PERS} & \text{3} \end{bmatrix} \\
\text{OBJ} & \text{f8,f9:}\begin{bmatrix} \text{PRED} & \text{‘book’} \\ \text{CASE} & \text{acc} \\ \text{NUM} & \text{pl} \\ \text{PERS} & \text{3} \end{bmatrix} \\
\text{ADJUNCT} & \left\{\text{f5:}\begin{bmatrix} \text{PRED} & \text{‘never’} \end{bmatrix}\right\} \\
\text{TENSE} & \text{present}
\end{bmatrix}
$$
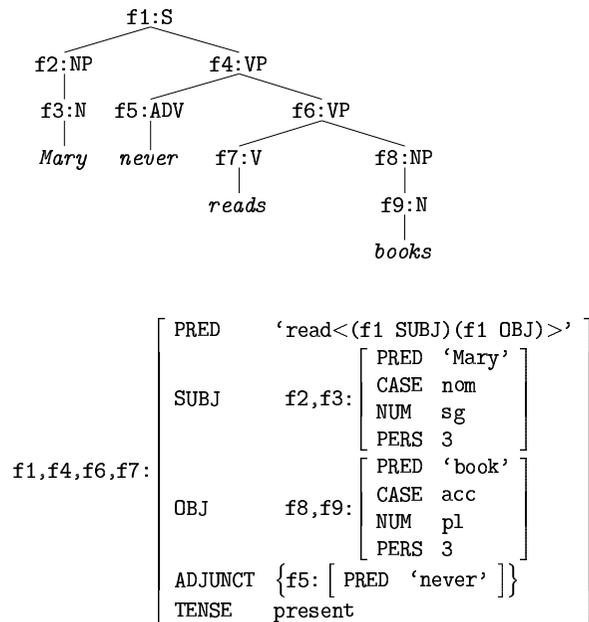
Note that the subcategorized functions, listed in the PRED value of the out-ermost f-structure, consist of a label plus a function, e.g. (f1 SUBJ). This expres-sion can be interpreted as a pointer to the f-structure denoted by the label plus function. For instance, the first argument slot of *read* points to the f-structure labelled by f2,f3, because this f-structure is embedded under the function SUBJ of the f-structure labelled by f1.

Usually, we omit the labels preceding the subcategorized functions within a PRED value, because the labels always refer to the f-structure containing the respective PRED feature.

Having established the link between the c-structure and f-structure repre-sentations, we now present further restrictions concerning the f-structure rep-resentation.

2.3   F-structure Restrictions

To complete the introduction, we want to clarify the notion of unification. Fur-thermore, we mention important principles applying to the f-structure repre-sentation.

Unification   In the previous section, it was said that two f-structures are "identical" or "are identified", e.g. by the annotation ↑=↓. The technical term is "unification": The f-structure that results from the identification of two f-structures is called the unification of the single f-structures.

Unification can be seen as the result of overlaying the single f-structures on each other. Hence, when two f-structure are unified, this means that they share all features. As an example, consider the two following f-structures.

$$
\begin{bmatrix} \text{SUBJ} & \begin{bmatrix} \text{NUM} & \text{sg} \\ \text{PERS} & \text{3} \end{bmatrix} \end{bmatrix}
\qquad
\begin{bmatrix} \text{SUBJ} & \begin{bmatrix} \text{NUM} & \text{sg} \\ \text{CASE} & \text{nom} \end{bmatrix} \end{bmatrix}
$$

The unification of both f-structures looks as follows.

$$
\begin{bmatrix} \text{SUBJ} & \begin{bmatrix} \text{NUM} & \text{sg} \\ \text{PERS} & \text{3} \\ \text{CASE} & \text{nom} \end{bmatrix} \end{bmatrix}
$$

That is, common features are merged (e.g. the feature SUBJ and the em-bedded feature NUM). Features that only occur in one of the f-structures are inherited by the resulting f-structure (PERS and CASE). Thus, formally, the re-sult of unifying f-structures is the union of their individual features, at all levels of embedding.

Obviously, the features that each f-structure contains must be compatible in order for unification to succeed. For instance, the f-structures [NUM sg] and [NUM pl] cannot be unified. (Remember that a feature must have a unique value. This is called the uniqueness/consistency condition.)

By definition, semantic forms (i.e. features whose values are quoted) can never unify, e.g. the f-structures [PRED 'book'] and [PRED 'book'] cannot be uni-fied. This mirrors the fact that each occurrence of the word *book* in a sentence adds to the meaning of the sentence.

Coherence, completeness   Features such as SUBJ and ADJUNCT are of-ten called syntactic or grammatical functions (GF). A subclass thereof consti-tutes functions that can be subcategorized, e.g., by a verb (SUBJ, OBJ, etc.). These functions are called argument functions or governable functions. Argu-ment functions underlie special restrictions.

(i) The coherence condition requires that any governable function that ap-pears in an f-structure be subcategorized by a PRED feature in the same f-structure, i.e. it must be listed as one of the functions that appear in the seman-tic form of a PRED feature, as in [PRED 'read<SUBJ,OBJ>']. Here, the functions SUBJ and OBJ are subcategorized.

(ii) The completeness condition requires that each function that appears in the semantic form of a PRED feature be realized in the same f-structure. (For additional requirements concerning semantic arguments, see below.)

As an example, consider the (simplified) f-structure of *Mary never reads books*.

$$
\begin{bmatrix}
\text{PRED} & \text{`read<SUBJ,OBJ>'} \\
\text{SUBJ} & \begin{bmatrix} \text{PRED} & \text{`Mary'} \end{bmatrix} \\
\text{OBJ} & \begin{bmatrix} \text{PRED} & \text{`book'} \end{bmatrix} \\
\text{ADJUNCT} & \{ \begin{bmatrix} \text{PRED} & \text{`never'} \end{bmatrix} \}
\end{bmatrix}
$$

The functions SUBJ and OBJ are governable/argument functions, hence they must figure in the PRED feature of the same f-structure (which they do in the example at hand). Conversely, all functions listed in the PRED feature must appear in the same f-structure (which they do as well). Note that the grammatical function ADJUNCT—not being an argument function—does not need to be subcategorized.

The sentence *Mary sleeps books* violates the coherence condition, cf. the (incoherent) f-structure below. The function OBJ that appears in the f-structure is not listed in the verb's PRED feature.

$$
\begin{bmatrix}
\text{PRED} & \text{`sleep<SUBJ>'} \\
\text{SUBJ} & \begin{bmatrix} \text{PRED} & \text{`Mary'} \end{bmatrix} \\
\boxed{\text{OBJ}} & \begin{bmatrix} \text{PRED} & \text{`book'} \end{bmatrix}
\end{bmatrix}
$$

Conversely, the sentence *Mary gives* violates the completeness condition, cf. the following (incomplete) f-structure. While the functions OBJ and OBJ2 (indirect object) are subcategorized, there are no such functions represented in the same f-structure.

$$
\begin{bmatrix}
\text{PRED} & \text{`give<SUBJ,}\boxed{\text{OBJ}}\text{,}\boxed{\text{OBJ2}}\text{>'} \\
\text{SUBJ} & \begin{bmatrix} \text{PRED} & \text{`Mary'} \end{bmatrix}
\end{bmatrix}
$$

The argument functions that are usually assumed in LFG (Bresnan 2001, p. 97f, Dalrymple 2001, p. 9) comprise SUBJ, OBJ, OBJ2 (or OBJtheta, indirect object), OBL (oblique argument, realized by PPs), OBLtheta (semantic oblique argument), COMP (complement, realized by sentential constituents), and XCOMP (similar to COMP, but without an overt SUBJ, see below).

Non-argument functions are ADJUNCT, XADJUNCT (similar to ADJUNCT, but without an overt SUBJ), and the discourse functions TOPIC and FOCUS.

Semantic vs. non-semantic arguments   As mentioned above (cf. p. 10), there are non-semantic arguments, e.g. the expletive subject in *It rains*.

Non-semantic arguments are treated differently from semantic arguments. The main difference is that non-semantic arguments do not (necessarily) have a PRED feature (remember that the PRED feature can be thought to represent the semantic content of a word).

The conditions on coherence and completeness impose additional restrictions on semantic arguments (i.e., PRED-bearing f-structures), namely that they must receive a semantic/thematic role by the PRED feature which subcategorizes them. For instance, a sentence such as *Mary rains* is ungrammatical since *Mary*'s f-structure contains a PRED feature but *rains* does not provide a thematic role for its subject.

In the notation of the PRED feature, the difference between semantic and non-semantic arguments is marked by the position of the argument functions within respectively outside the angle brackets. Arguments within the brackets receive a thematic role, arguments outside the brackets do not.

As an example, compare the f-structures of *Mary sleeps* and *It rains*. (Since *rains* inflects like any other verb in third person singular, the subject's f-structure of *It rains* contains inflectional information.)

$$
\begin{bmatrix}
\text{PRED} & \text{`sleep<SUBJ>'} \\
\text{SUBJ} & \begin{bmatrix} \text{PRED} & \text{`Mary'} \\ \text{CASE} & \text{nom} \\ \text{NUM} & \text{sg} \\ \text{PERS} & 3 \end{bmatrix}
\end{bmatrix}
\qquad
\begin{bmatrix}
\text{PRED} & \text{`rain<>SUBJ'} \\
\text{SUBJ} & \begin{bmatrix} \text{CASE} & \text{nom} \\ \text{NUM} & \text{sg} \\ \text{PERS} & 3 \end{bmatrix}
\end{bmatrix}
$$

Functional control   In certain cases, a constituent may fulfil the subcategorization requirements of two verbs simultaneously, e.g. with modals or raising verbs as in *Mary seems to come*. On the one hand, *Mary* functions as the syntactic (non-semantic/non-thematic) subject of *seems*, on the other hand, *Mary* is the logical (semantic) subject of *come*.

The conflict is solved by the embedding verb, i.e. the modal or raising verb. It subcategorizes for a non-semantic SUBJ argument and an XCOMP argument. Furthermore, it specifies that its own subject also functions as the subject of the XCOMP argument, see the lexicon entry below. (XCOMP is called an "open function", because the subject of the XCOMP argument is specified externally, i.e. the constituent which projects the XCOMP function does not dominate the subject of XCOMP. The 'X' within the name 'XCOMP' stems from the fact that different categories may realize the XCOMP argument in English, e.g. VPs, APs, PPs.)

```
seems  V  (↑PRED) = 'seem<(↑XCOMP)>(↑SUBJ)'
          (↑SUBJ) = (↑XCOMP SUBJ)
```

The relation defined between the two identified (unified) functions is called "functional control". In an f-structure, the relation is indicated by a line linking the two involved f-structures.

$$\begin{bmatrix} \text{PRED} & \text{'seem<XCOMP>SUBJ'} \\ \text{SUBJ} & \begin{bmatrix} \text{PRED} & \text{'Mary'} \end{bmatrix} \\ \text{XCOMP} & \begin{bmatrix} \text{PRED} & \text{'come<SUBJ>'} \\ \text{SUBJ} & [\ ] \end{bmatrix} \end{bmatrix}$$
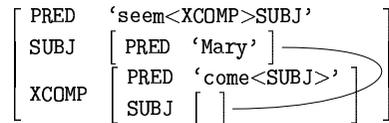
Note that the coherence condition on the semantic argument *Mary* is fulfilled by the embedded verb *come* in that *Mary* receives a thematic role from *come*.

Constraining equations, existential constraints All feature-value equations we have seen so far are so-called "defining equations". That means that they introduce a feature-value pair to a certain f-structure. Besides defining equations, there are other types of equation in LFG: constraining equations and existential constraints.

Constraining equations (notation: =c) impose the restriction that a specific feature-value pair be part of the respective f-structure. That is, some other rule or lexicon entry must introduce the feature-value pair to the f-structure. For instance, a verb like *rains* requires the expletive *it* as its subject. This constraint can be stated in the lexicon entry of *rains* (the expletive *it* does not introduce a semantic form since it has no meaning).

```
rains  V     (↑PRED) = 'rain<>(↑SUBJ)'
             (↑SUBJ FORM) =c it

it     PRON  (↑FORM) = it
```

Constraining equations can also express negation (notation: ˜=), requiring that a certain f-structure must not contain a certain feature-value pair. For instance, the lexicon entry of the verb form *read* could state that its subject must not be third person singular, rather than enumerate all other possibilities. (Disjunctive annotations are marked by curly brackets '{ ... | ... }'.)

```
read V  (↑PRED) = 'read<(↑SUBJ)(↑OBJ)>'
        { (↑SUBJ NUM) = sg (↑SUBJ PERS) ˜= 3
        | (↑SUBJ NUM) = pl
        }
        (↑TENSE) = present
```

Existential constraints only require the presence of a feature without specifying its value. For instance, a singular count noun such as *book* must be preceded by a specifier. This restriction can be encoded by (↑SPEC) in the lexicon entry of *book*. Both determiners, *the* and *a* could fulfil this restriction.

```
book  N  (↑PRED) = 'book'
         (↑SPEC)

the   D  (↑SPEC) = def

a     D  (↑SPEC) = indef
```

Finally, existential constraints can be negated, e.g. proper nouns may prohibit the presence of a specifier by the constraint ˜(↑SPEC), cf. the lexicon entry of *Mary*.

```
Mary N  (↑PRED) = 'Mary'
        ˜(↑SPEC)
```

Functional uncertainty In English, a topicalized wh-phrase such as *who* can realize various kinds of functions (relative to the outermost f-structure), e.g. OBJ (2), COMP OBJ (3), XCOMP OBJ (4), XCOMP COMP OBJ (5), COMP XCOMP OBJ (6), XCOMP COMP XCOMP OBJ (7), etc.

(2) *Who did you see?*
(3) *Who does Mary think that John saw?*
(4) *Who does Mary seem to see?*
(5) *Who does Mary seem to think that John saw?*
(6) *Who does Mary think that John seems to see?*
(7) *Who does Mary seem to think that John wants to see?*

In LFG, such long-distance (unbounded) dependencies are captured by so-called functional uncertainty equations. The annotation of the topicalized wh-phrase is built of a regular expression which can be instantiated in different ways, cf. the CP rule.

```
CP →            NP              S
       (↑{COMP|XCOMP}* OBJ)=↓  ↑=↓
```

The regular expression {COMP|XCOMP}* allows for all of the above mentioned functions: {COMP|XCOMP} denotes either one COMP or one XCOMP. The Kleene star (*) following the brackets denotes zero or more repetitions of the bracketed expression. Hence {COMP|XCOMP}* denotes any sequence built of COMP and/or XCOMP features, including the empty string.

Inside-out equations   The equations we have seen so far all start with ↑ or ↓, optionally followed by one or more features, e.g. (↑OBJ CASE) = acc. In this example, the f-structure designated by ↑ is said to embed the function OBJ which contains the feature [CASE acc]. This type of equation is called "outside-in equation" since the feature path starts from an outer f-structure and proceeds to inner (i.e., more embedded) f-structures.

The converse also exists, "inside-out equation". As an example, consider an example entry of the reflexive pronoun *himself*.

```
himself PRON   (↑PRED) = 'pro'
               ((OBJ ↑) SUBJ GEND) = masc
               ((OBJ ↑) SUBJ NUM) = sg
```

In the expression (OBJ ↑), the ↑-arrow has the same meaning as usual, i.e. it denotes the f-structure of the mother node. This f-structure is said to be embedded under the function OBJ, in other words, the reflexive pronoun functions as the OBJ. (OBJ ↑), therefore, denotes the outer f-structure, which contains the OBJ function.

The remaining parts of the expression are outside-in again. The outer f-structure furthermore has to embed a function SUBJ whose GEND and NUM features are restricted to the values masc and sg, respectively. That is, the equation enforces agreement between the subject and the reflexive pronoun that represents the OBJ function, cf. the f-structure projected by the lexicon entry of the reflexive pronoun.

$$
\begin{bmatrix}
\text{SUBJ} & \begin{bmatrix} \text{GEND} & \text{masc} \\ \text{NUM} & \text{sg} \end{bmatrix} \\
\text{OBJ} & \begin{bmatrix} \text{PRED} & \text{'pro'} \end{bmatrix}
\end{bmatrix}
$$

C-structure vs. f-structure heads   Finally, we want to introduce the notions "c-structure head", "f-structure head", and "co-head".

The distinction between c-structure and f-structure implies two different notions of "head". C-structure heads are heads in the sense of X′-theory. For instance, in the following annotated tree, D and Dbar are c-structure heads of the DP, whereas NP is the c-structure complement of D.
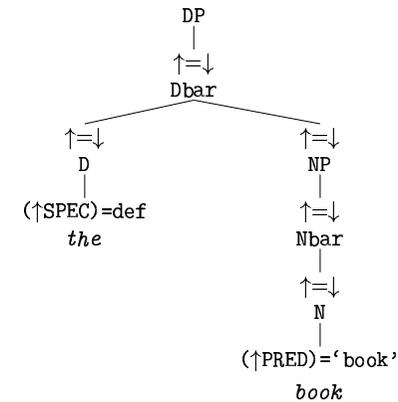
In f-structure, however, any node whose f-structure is unified with the f-structure of its mother, is an (f-structure) head. Hence in the tree above, both D and NP are f-structure heads, being annotated by ↑=↓. D and NP are said to be (f-structure) co-heads.

In our presentation, we also make use of the term "semantic head". This term refers to the terminal that introduces the PRED value of an f-structure. In the annotated tree above, N is a semantic head.

2.4   Summary

In this chapter, we presented an introduction to the formalism of LFG, Lexical-Functional Grammar.

LFG distinguishes two representation levels to encode syntactic properties of sentences, the constituent structure (c-structure) and functional structure (f-structure). The structures are linked by a correspondence function (or mapping relation), called "ϕ-projection".

The c-structure representation of a sentence is constrained by the context-free grammar rules. The f-structure representation of a sentence is constrained (i) by the f-structure constraints associated with the c-structure rules and lexicon entries that are applied in the analysis of this sentence and (ii) by the principles of functional uniqueness, coherence, and completeness.

A sentence is grammatical if and only if it satisfies all grammar constraints in c-structure and f-structure.

Chapter 3

Introduction to XLE

Contents

This chapter contains an introduction to XLE, the Xerox Linguistic Environment. XLE is a grammar development platform which allows the user to define a grammar in a notation similar to that of LFG.

The grammar writer enters rules and lexicon entries in an editor (Emacs). On the basis of these rules and lexicon entries, XLE parses input sentences and displays their c-structure and f-structure analyses. XLE can also generate strings from a given f-structure.

XLE allows for the integration of external modules, such as a tokenizer, a morphological analyzer, and a lexicon (XLE can handle large lexicons as required in a realistic application context). Moreover, XLE provides various means of abstraction, e.g. macros, which are prerequisites for the development of a large-scale grammar. Finally, XLE can parse complex sentences in a reasonable amount of time. This makes XLE a suitable software for the development of large-scale grammars.

This introduction is not a user manual for XLE. For instance, it does not describe how to parse sentences with XLE. Instead, this introduction presents the notation of grammar code within XLE (knowledge of this notation is assumed in the documentation chapters in Part III). In addition, the introduction presents the preprocessing steps that an input string has to undergo before it is processed by the grammar. (For an XLE introduction including a description of the user interface, see Butt *et al.* 1999, ch. 11ff.)

We start by presenting the external modules performing the preprocessing steps (sec. 3.2), followed by the lexicon module (sec. 3.2.5). Subsequently, the XLE-specific notation and the means of abstraction are described (sec. 3.3).

First of all, however, we shortly describe how XLE presents the c-structure and f-structure analyses to the user (sec. 3.1).

3.1   C-Structure and F-Structure Display in XLE

First, consider the example c-structure and f-structure of (8), as displayed by XLE (shown below the example). (As can be seen by the tree representation, our analysis implements a DP analysis. The internal structure of the DP is addressed below (cf. p. 64), where we give reasons for omitting the Dbar projection and adding an intermediate projection DPx.)

(8)   *Hans ist gekommen.*
      H.   is   come
      'Hans came.'

```
CS 1:                ROOT:1017

                CProot[std]:843        PERIOD:74

         DP[std]:378              Cbar:841  .:73

    DPx[std]:377 Vaux[sein,fin]:1039 VP[v,+h,part]:1027

         NP:269           ist:18    VPx[v,+h,part]:1026

      NAMEP:268                     VC[v,+h,part]:727

       Hans:0                        V[v,part]:725

                                   Vx[v,part]:724

                                    gekommen:38
```
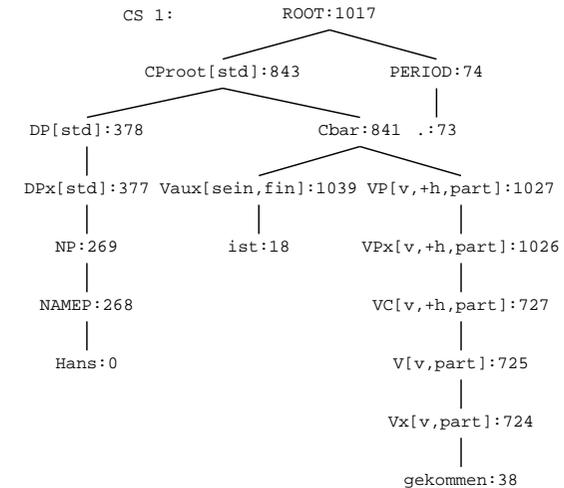
"Hans ist gekommen."

```
        PRED      'kommen<[0:Hans]>'

               0 PRED   'Hans'
             268 NSEM   [PROPER first_name]
        73 SUBJ 269
        74       377 NTYPE [GRAIN proper]
        38       378 CASE nom, GEND masc, NUM sg, PERS 3
       724
       725              [ AUX-FORM  [sein-perf] ]
       727 CHECK        [ VLEX      [AUX-SELECT sein] ]
      1026              [ VMORPH    [PARTICIPLE perfect] ]
      1027
        18
      1039 TNS-ASP  ASPECT [PERF +]
       841           MOOD indicative, TENSE present
       843 TOPIC    [0:Hans]
      1017 CLAUSE-TYPE declarative, PASSIVE -, STMT-TYPE declarative, VTYPE main
```

Each c-structure node is labelled by a number, e.g. the terminal *Hans* by the number 0 ('Hans:0'), the preterminal NAMEP by 268, etc. F-structures are labelled as well. The numbers attached to the left bracket of an f-structure list all c-structure nodes that project this f-structure (i.e. the nodes that are mapped to this f-structure). For instance, the f-structure embedded under the feature SUBJ is projected from the nodes labelled 0, 268, 269, 377, and 378, i.e. from the terminal node *Hans* (labelled 0), the nodes NAMEP (268), NP (269), DPx[std] (377), and DP[std] (378).

The f-structures embedded under the features TNS-ASP and CHECK are not labelled by any number. This means that they are not projected directly from

any c-structure node. Instead, the features of the embedded f-structures, such as [MOOD indicative], which is embedded by TNS-ASP, are introduced by complex feature equations, e.g. by (↑TNS-ASP MOOD) = indicative. In such cases, only the outer f-structure is labelled by (the number of) the projecting c-structure node.

Coreferences between f-structures are encoded by numbers inside the brackets of an f-structure. For instance, in the above example one and the same f-structure is the value of the function SUBJ and TOPIC simultaneously. In traditional LFG presentations, only one of the coreferent functions embeds the full f-structure as its value, whereas the other function embeds an empty f-structure, that is linked to the coreferent f-structure by a line.

$$
\begin{bmatrix}
\text{PRED} & \text{'kommen<SUBJ>'} \\
\text{SUBJ} & \begin{bmatrix} \text{PRED 'Hans'} \end{bmatrix} \\
\text{TOPIC} & \begin{bmatrix} \ \end{bmatrix}
\end{bmatrix}
$$
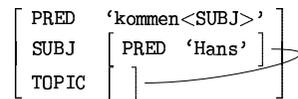
Instead of using a line, XLE fills the embedded empty f-structure by a number label, followed by a colon plus a "word": [0:Hans], compare the XLE display of the above f-structure.

```
"Hans ist gekommen"

  37 PRED      'kommen<[0:Hans]>'
 639
 640          0 PRED 'Hans'
 642 SUBJ   258
 766        259
 767        368
  17        370
 778 TOPIC    [0:Hans]
 754
 756
 758
```

The label is the smallest number from the set of labels marking the coreferring f-structure (= the value of SUBJ). The "word" corresponds to the value of the PRED feature of the coreferring f-structure. This way, the user can easily see if the TOPIC function is linked as desired.

The same notation is used for the subcategorized functions that are listed within the PRED value of the verb: 'kommen<[0:Hans]>'.

Note that in the remaining part of this work, the display of all but the smallest label number is suppressed for better readability, compare the example below.

```
CS 1:          ROOT
              /    \
      CProot[std]   PERIOD
        /   \          |
   DP[std]  Cbar        .
      |      /  \
 DPx[std] Vaux[sein,fin] VP[v,+h,part]
     |         |            |
    NP        ist       VPx[v,+h,part]
     |                      |
   NAMEP                VC[v,+h,part]
     |                      |
   Hans                 V[v,part]
                            |
                        Vx[v,part]
                            |
                        gekommen
```
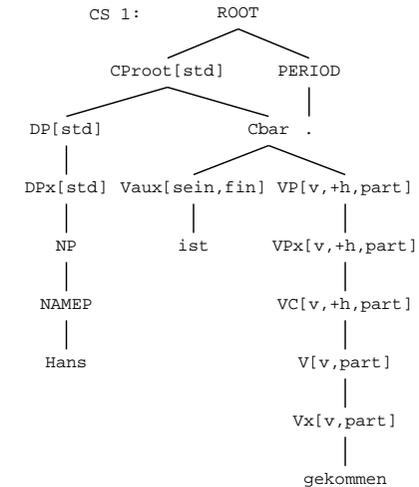
```
"Hans ist gekommen."

PRED     'kommen<[0:Hans]>'
         PRED  'Hans'
         NSEM  [PROPER first_name]
SUBJ     NTYPE [GRAIN proper]
       0 CASE nom, GEND masc, NUM sg, PERS 3

         AUX-FORM <sein-perf>
CHECK    VLEX     [AUX-SELECT sein]
         VMORPH   [PARTICIPLE perfect]

TNS-ASP  ASPECT [PERF +]
         MOOD indicative, TENSE present
TOPIC    [0:Hans]
      18 CLAUSE-TYPE declarative, PASSIVE -, STMT-TYPE declarative, VTYPE main
```

(Trees and f-structures displayed in our documentation are always associated with rules, whose effects are to be illustrated by the displayed trees and f-structures. These rules allow the reader to infer the correspondence relation ϕ.)

3.2 External Modules

A sentence that is parsed by a grammar first has to be preprocessed. In our implementation, a tokenizer (sec. 3.2.1), a morphological analyzer (sec. 3.2.2),

and a guesser (sec. 3.2.3) modify the input string before it is finally analyzed by the grammar. These components consist of finite-state transducers (Beesley and Karttunen 2003).

The interface between the output of the preprocessors and the grammar rules is realized by so-called "sublexical rules" (sec. 3.2.4).

Another external module is a lexicon containing information such as subcategorization properties (sec. 3.2.5). For reasons that will become clear below we call the external lexicon "stem lexicon".

3.2.1 Tokenizer

The main task of a tokenizer is to split the input string into a sequence of delimited tokens. The single tokens then represent the input to the next preprocessing step, the morphological analysis.

Usually, spaces indicate token boundaries, i.e. two words separated by a space are two tokens. However, most punctuation marks follow the preceding word immediately, e.g. the full stop in example (9).

(9)   *Hans kommt.*
       H.   comes
       'Hans is coming.'

The tokenizer inserts token boundaries, marked by '@', between words and punctuation marks. That is, the example sentence above is transformed by the tokenizer into (10) (note the token boundary between *kommt* and the final full stop).

(10)   *Hans@kommt@.*

The output tokens of the tokenizer are the input to the subsequent steps. In the final representation, i.e. in c-structure, these tokens represent the terminal nodes. As an example, compare the input sentence in (11) and the terminal nodes in the c-structure analysis below. Note that the full stop realizes a terminal node of its own (the marker of token boundaries, '@', is not displayed).

(11)   *Hans kommt.*
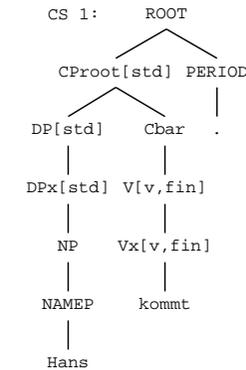
Abbreviated words ending with a full stop must be treated differently since the full stop is part of the word in these cases, cf. the representation of the abbreviated word *o.* (*oder*) 'or' in the c-structure analysis of (12).

(12)   *Hans o. Maria*
        H.   or M.
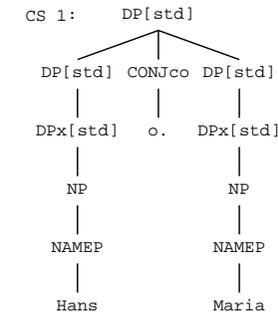
Besides insertion of the token boundaries, the tokenizer in our implementation performs further tasks, which involve case conversion ((de)capitalization) and punctuation doubling.

Optional decapitalization   The first word of a sentence is optionally decapitalized by the tokenizer. For instance, if the input string is *Er kommt* ('He is coming'), the tokenizer outputs two strings, the original string *Er kommt* and

an additional one, *er kommt*. Only the second one will be analyzed successfully by the morphological component. (The first variant receives a so-called "suboptimal" analysis (cf. sec. 3.3.5), since the non-German word *Er* will be guessed to be a proper noun (cf. sec. 3.2.3).)

In certain cases, the decapitalization process results in a real ambiguity. For instance, the input string *Sie kommen.* ('They/you come.') allows for two readings which differ with respect to the case of the first letter. Lower-case *sie* 'they' is the third person plural form of the personal pronoun. Upper-case *Sie* 'you' is the polite form to address someone. The following c-structures represent the two analyses that our grammar outputs for the string *Sie kommen.*.

Capitalized Version   Decapitalized Version
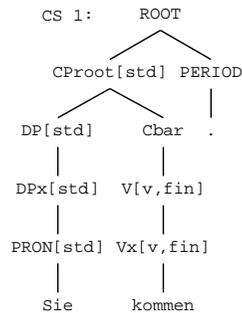
```
  CS 1:    ROOT                    CS 2:    ROOT
           / \                              / \
  CProot[std] PERIOD              CProot[std] PERIOD
      / \       |                     / \       |
  DP[std] Cbar  .                 DP[std] Cbar  .
    |      |                        |      |
  DPx[std] V[v,fin]              DPx[std] V[v,fin]
    |      |                        |      |
  PRON[std] Vx[v,fin]            PRON[std] Vx[v,fin]
    |      |                        |      |
   Sie   kommen                   sie    kommen
```
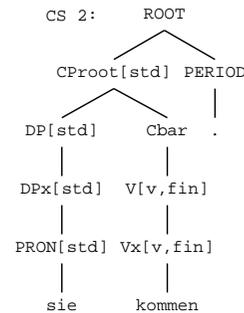
Optional capitalization   The tokenizer also optionally capitalizes certain tokens, namely elliptical compounds that start with a hyphen, e.g. as in *Folkloretänze und -lieder* 'folk dances and (folk) songs'. The hyphen indicates that the second compound is elliptical, its full form being *Folklorelieder*.

The morphological component we make use of can only analyze nouns written in capital letters, hence the elliptic noun *-lieder* must be capitalized. As an example, compare the input sentence in (13) and its c-structure analysis, featuring the capitalized terminal *-Lieder*.

(13)   *Folkloretänze und -lieder*
       folk dances   and songs
       'folk dances and (folk) songs'

```
  CS 1:      DP[std]
              / | \
     DP[std] CONJco DP[std]
        |      |       |
    DPx[std]  und   DPx[std]
        |              |
       NP             NP
        |              |
     N[comm]        N[comm]
        |              |
  Folkloretänze      -Lieder
```
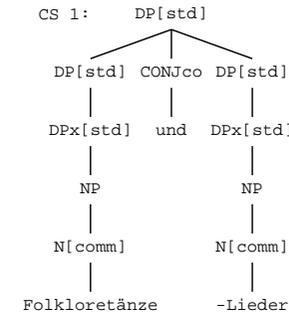
Punctuation doubling   In German, subordinate sentences are enclosed by commas, cf. the relative clause in (14).

(14)   *Jeder,   der lacht,   gewinnt.*
       Everybody who laughes wins

However, if the relative clause occurs in sentence-final position, the second comma is omitted in front of the full stop, cf. (15).

(15)   *Jeder   gewinnt, der lacht.*
       Everybody wins   who laughes

Rather than implementing different rules for relative clauses (with or without closing punctuation), the tokenizer provides additional commas in front of the full stop (and in front of any punctuation mark). As an example, cf. the terminals in the c-structure analysis of the sentence in (16), where the comma following the relative clause (CPdep[rel]) is not part of the original input.

(16)   *Jeder   gewinnt, der lacht.*
       Everybody wins   who laughes

```
CS 1:         ROOT
          CProot[std]  PERIOD
                          |
       DP[std]      Cbar  .
          |
    DPx[std] V[v,fin]     NACHFELD
       |
    D[std]  Vx[v,fin] COMMA  CPdep[rel]  COMMA
       |        |        |                  |
     jeder   gewinnt    ,  DP[rel]  VP[v,fin]  ,
                              |
                          DPx[rel] VPx[v,fin]
                              |
                          PRON[rel] VC[v,fin]
                              |
                             der    V[v,fin]
                                       |
                                    Vx[v,fin]
                                       |
                                     lacht
```
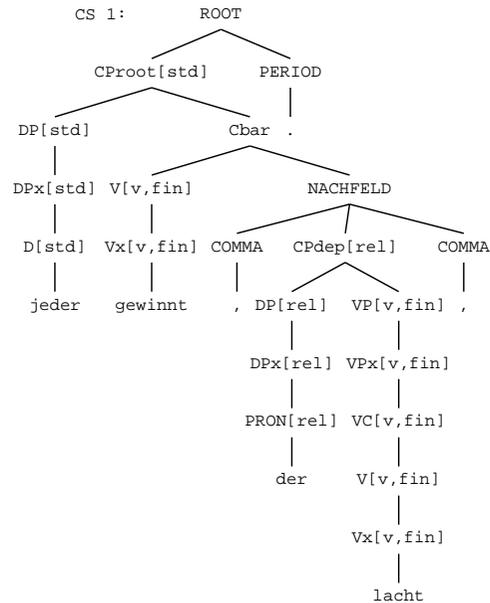
An example where a comma has to be inserted in front of another comma is (17). The appositive *siegessicher* 'sure to win' and the complement clause are separated by two commas, as can be seen in the c-structure analysis.

(17) *Jeder   glaubt, siegessicher, daß er gewinnt.*
      everybody believes victory‗sure that he wins
      'Everybody, sure to win, believes that he wins.'

```
CS 1:            ROOT
             CProot[std]  PERIOD
                             |
       DP[std]        Cbar  .
          |
    DPx[std] V[v,fin]              NACHFELD
       |
    D[std] Vx[v,fin] COMMA AP[std,-infl] COMMA COMMA CPdep[std] COMMA
       |       |       |        |          |     |                |
     jeder   glaubt   ,    APx[std,-infl]  ,     ,  Cbar-comp[std]  ,
                               |
                            A[-infl]                 C[std] VP[v,fin]
                               |                       |
                          siegessicher               daß   VPx[v,fin]
                                                              |
                                                     DP[std]  VPx[v,fin]
                                                        |         |
                                                    DPx[std]  VC[v,fin]
                                                        |         |
                                                    PRON[std]  V[v,fin]
                                                        |         |
                                                       er      Vx[v,fin]
                                                                  |
                                                               gewinnt
```
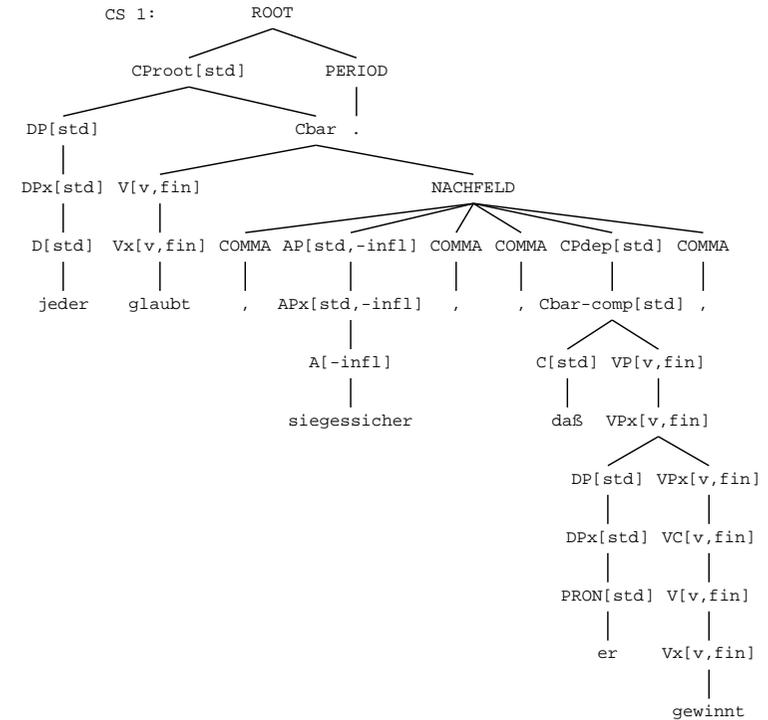
3.2.2 Morphological Analyzer

As mentioned above, the tokens marked by the tokenizer represent the input to the morphological analyzer. The morphological analyzer maps each token to a sequence of morphological tags. These tags are the actual input to the grammar.

The morphological component we make use of is a variant of DMOR1 (Becker 2001). It outputs four types of tags, encoding (i) the lemma, (ii) derivational information (optional), (iii) the part of speech, (iv) inflectional information. For instance, the morphological tags for the token *Karls* (the genitive form of the proper name *Karl*) include the lemma: Karl, part of speech: +NPROP (proper noun), and inflectional tags: .Masc (masculine), .Gen (genitive), .Sg (singular). (M-Input/M-Output means input/output of the morphological analyzer.)

```
M-Input   Karls
M-Output  Karl +NPROP .Masc .Gen .Sg
```

As a consequence of this approach, the terminals that our syntactic grammar has to deal with are not surface word forms such as *Karls* 'Karl's' or *kommt* 'comes'. Instead, the terminals are the tag tokens output by the morphological analyzer, such as +NPROP, .Sg, or lemma forms like Karl. Consequently, the (internal) lexicon of our grammar does not comprise entries for word forms like *Karls* but entries for +NPROP, .Sg, etc. (cf. sec. 3.2.4). In the following, we call this type of lexicon the "tag lexicon".

The advantage of this approach is that information can be bundled, e.g., idiosyncratic lexical information can be associated with the lemma form, hence matching all inflected forms automatically (otherwise the information would have to be added to each single inflected form). Furthermore, the morphological analyzer can analyze words that do not occur in any lexicon, e.g. compounds that are not lexicalized, cf. (18).

(18)   *Schwachstrominstallationsarbeiten*
        weak_power_installation_work[FEM,PL]
        'work for the installation of weak power'

Here, the morphological analysis supplies the information that the compound is a feminine plural noun. Based on this information, the grammar can handle such compounds, whereas a grammar relying on a full-form lexicon could only guess the part of speech and inflection of the compound.

We now describe each tag type in turn, starting with the lemma tag.

Lemma   The first tag of the morphological output encodes the lemma. Usually, the lemma of a word corresponds to the uninflected nominative singular (for nouns, adjectives) or the infinitive (for verbs). For instance, the inflected (proper) noun *Karls* 'Karl's' is assigned the lemma form 'Karl', see below.

```
M-Input   Karls
M-Output  Karl +NPROP .Masc .Gen .Sg
```

The inflected verbs *kommt* 'come' and *anklagst* 'accuse' are associated with the forms of the infinitive, 'kommen' and 'an#klagen', respectively (*anklagen* is a so-called particle verb, hence the hash sign following the particle *an*). The form *kommt* is morphologically ambiguous, therefore the morphological analyzer outputs several options.

```
M-Input   kommt   ('come')
M-Output  kommen +V .3 .Sg .Pres .Ind
          kommen +V .2 .Pl .Pres .Ind
          kommen +V .Imp .Pl
```

```
M-Input   anklagst   ('accuse')
M-Output  an#klagen +V .2 .Sg .Pres .Ind
```

In case there is no uninflected or unambiguous form in nominative singular (e.g. with pronouns and quantifying expressions), the feminine form is chosen. As an example, cf. the morphological analyses of *ihn* 'he[ACC]' and *jedem* 'each[MASC/NEUT,DAT]' with the lemmas *sie* 'she' and *jede* 'each[FEM]', respectively.

```
M-Input   ihn   ('him[ACC]')
M-Output  sie +PPRO .Pers .3 .Sg .Masc .Acc .None
```

```
M-Input   jedem   ('each[MASC/NEUT,DAT]')
M-Output  jede +INDEF .Pro .MN .Dat .Sg .St
```

The lemma form is relevant to the (stem) lexicon lookup during parsing. Lexical information, e.g subcategorization, is associated with lemma forms. (Without lemma forms, lexical information would have to be multiplied for each single inflected form.)

Derivational information   Tags starting with ^ usually encode some sort of derivational information. Examples are:

- Nominalized adjectives, marked by ^ADJ (and ^VPAST/^VPRES). These are assigned the (non-capitalized) lemma of the base word, i.e. the adjective lemma, cf. the analysis of *Wesentliches* 'essential (parts)'.

```
M-Input   Wesentliches   ('essential parts')
M-Output  wesentlich ^ADJ .Pos +NN .Neut .NA .Sg .St
```

Similarly, nominalized deverbal adjectives are assigned the corresponding verb lemma, cf. the analysis of *Angeklagter* 'defendant', which is derived from the past participle of *anklagen* 'accuse'.

```
M-Input    Angeklagter  ('defendant')
M-Output   an#klagen ^VPAST ^ADJ +NN .Fem .GD .Sg .St
           an#klagen ^VPAST ^ADJ +NN .Masc .Nom .Sg .St
           an#klagen ^VPAST ^ADJ +NN .MFN .Gen .Pl .St
```

- Nominalized infinitives, marked by ^VINF, e.g. *Laufen* 'running'.

```
M-Input    Laufen  ('running')
M-Output   laufen ^VINF +NN .Neut .NDA .Sg
```

- Capitalized pronouns, marked by ^CAP, e.g. *Sie* 'you' (polite form).

```
M-Input    Sie  ('you')
M-Output   sie ^CAP +PPRO .Pers .3 .Pl .MFN .None
```

- Feminine nouns ending with -*in*, marked by ^FEM, e.g. *Bürgermeisterin* 'mayoress' (compared to *Bürgermeister* 'mayor').

```
M-Input    Bürgermeisterin  ('mayoress')
M-Output   Bürger +CMPD Meister ^FEM +NN .Fem .NGDA .Sg
```

Note that the noun *Bürgermeisterin* is a compound which is split into its components *Bürger* 'citizen' and *Meister* 'master'. Since this noun is clearly lexicalized, it should preferably be output as a complex form and not be split into its components. However, the morphological component and the external lexicon resource (the stem lexicon) are usually being developed independently from the grammar. That is, for the grammar writer the morphological component and the stem lexicon are black boxes which more or less have to be dealt with as they are.

Part of speech   Following the lemma tag (and optional derivation tags) is the part-of-speech tag. Part-of-speech tags are marked with a plus sign. In many cases, the tag does not only specify the actual part of speech (such as noun, pronoun, verb, adjective). It rather specifies subtypes of the respective parts of speech, e.g. common noun (+NN) vs. proper noun (+NPROP), personal pronoun (+PPRO) vs. demonstrative pronoun (+DEM) vs. interrogative pronoun (+WPRO), etc.

As examples, consider the analyses of the common noun *Mann* 'man', which can also be used as a proper noun (as in *Thomas Mann*).

```
M-Input    Mann  ('man/Mann')
M-Output   Mann +NN .Masc .NDA .Sg
           Mann +NPROP .NoGend .NGDA .Sg
```

(In some rare cases, more fine-grained subtype information is encoded by tags starting with a full stop. For instance, personal pronouns (+PPRO) are further subclassified by .Pers (ordinary persononal pronouns) vs. .Refl (reflexive pronouns) vs. .Rec (reciprocal pronouns).)

Note that even punctuation marks receive a "morphological" analysis, cf. the analysis of a full stop.

```
M-Input    .
M-Output   . +PUNCT .Sent
```

Inflectional information   The remaining tags encode inflectional information, e.g. gender, case, number. They all start with a full stop. In the variant of DMOR1 we use, ambiguous forms are often indicated by underspecified tags. For instance, nouns that do not inflect for case are marked by .NGDA (nominative/genitive/dative/accusative).

Interaction of the components   The interaction between the tokenizer, morphological analyzer, and grammar works as follows. The tokenizer operates on the whole string, inserting token boundaries between words (= tokens). If the tokenizer introduces ambiguities, e.g. by optionally decapitalizing the first token in a sentence, the analyses of a token are joined to build a disjunctive set.

The morphological analyzer is applied one by one to the tokens (which might be members of a disjunctive set). The analyses that the morphological component supplies for a token again build a disjunctive set. Hence, the analysis of the single input string yields a sequence of disjunctive sets (which may contain one member only).

The grammar component takes into account all members of all sets. C-structure and f-structure constraints determine which member of a set is to be combined with which members of the other sets in order to yield a successful analysis.

The following figure illustrates the preprocessing steps for the input string *Sie kommt.* ('She is coming.'). (The spaces in the output of the tokenizer and morphology should be read as token boundaries. The set members that figure in the successful analysis are set in italics.)

```
                        Sie kommt.
                            |
                    +---------------+
                    |   TOKENIZER   |
                    +---------------+
                            ↓
                { Sie | sie } kommt { . | , . }
                            |
                    +---------------+
                    |  MORPHOLOGY   |
                    +---------------+
                            ↓
    {   sie ^CAP +PPRO .Pers .3 .Pl .MFN .NA .None
    |   sie +PPRO .Pers .3 .Pl .MFN .NA .None
    |   sie +PPRO .Pers .3 .Sg .Fem .NA .None }

        {   kommen +V .Imp .Pl
        |   kommen +V .3 .Sg .Pres .Ind
        |   kommen +V .2 .Pl .Pres .Ind }

        {   . +PUNCT .Sent
        |   , +PUNCT .Comma . +PUNCT .Sent }
                            |
                    +---------------+
                    |   GRAMMAR     |
                    +---------------+
```
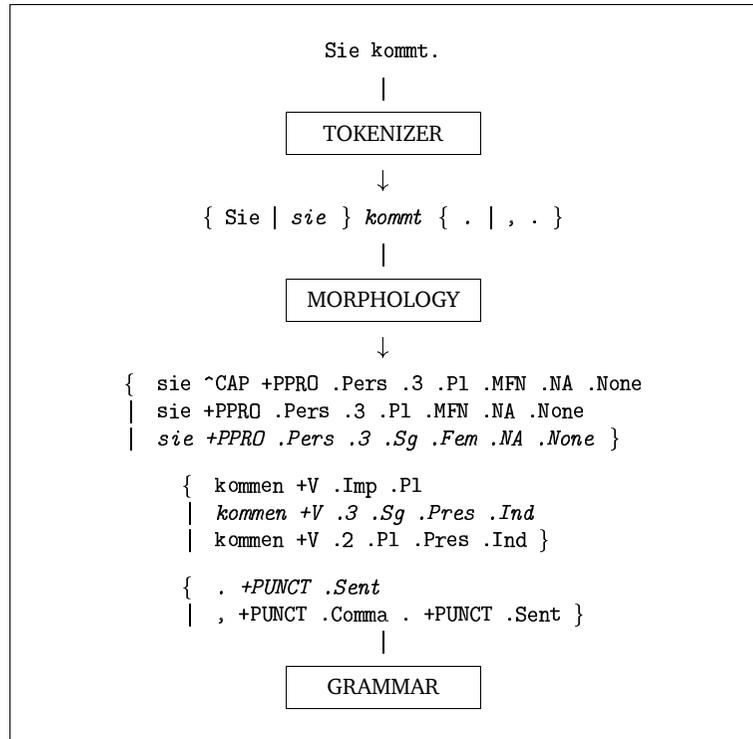
First the tokenizer optionally decapitalizes the first token and inserts additional commas (one comma shown in the example above). The morphological analysis supplies one analysis for capitalized *Sie* (marked by ^CAP), and two analyses for decapitalized *sie*. It furthermore introduces a three-way ambiguity for *kommt*.

That is, in the example above, the input to the grammar component consists of 3x3x2 = 18 different possible morphological token sequences. Due to grammar constraints, only one optimal analysis represents the final output in this example. This analysis chooses the decapitalized *sie* in third person singular from the first set, *kommt* as third person singular (second set), and no additional comma (third set).

The grammar may also make use of the tokenizer's tokens directly, skipping the morphological analysis. This is what we do in the case of (lexicalized) multi-word expressions, such as *in der Regel* 'usually'. That is, the grammar ignores

the corresponding morphological tag sequences and instead operates on the original input tokens 'in', 'der', and 'Regel' jointly, cf. the terminal nodes in the c-structure analysis of (19). (Such cases are handled by so-called "full-form lexicon entries" (cf. p. 50).)

(19) *In der Regel kocht Hans.*
      in the rule   cooks H.
      'Usually, Hans cooks.'

```
CS 1:          ROOT
              /    \
        CProot[std]  PERIOD
         /    \         |
    ADVP[std]  Cbar     .
       |      /   \
    ADV[std] V[v,fin] DP[std]
       |       |       |
  in der Regel Vx[v,fin] DPx[std]
              |         |
            kocht       NP
                        |
                      NAMEP
                        |
                       Hans
```
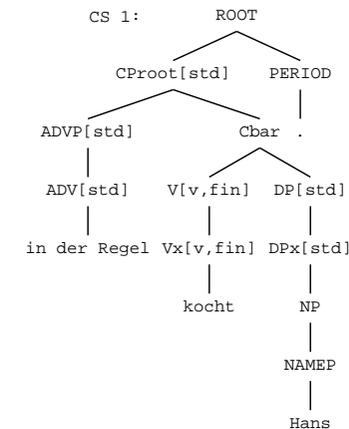
3.2.3   Guesser

A large-scale grammar is supposed to parse large, and varied, text, e.g. newspaper text. Obviously, many of the words that occur in such texts are unknown to the morphological analyzer, e.g. proper nouns, neologisms, foreign words, or words that are misspelt.

In these cases, so-called guessers can be exploited that categorize words according to superficial properties, e.g. capitalization or alphanumeric characters. The guesser we are using marks such words by tags that mimick morphological tags in that they start with a plus sign or full stop. These tags also represent an input to the grammar. The special tag +MUnknown marks the input as being provided by the guesser.

Examples of guessed words are shown below (M-Input/M-Output should be read as "guesser input/output" here). Note that our guesser operates successfully only on capitalized words and alphanumeric strings.

```
M-Input    Exr
M-Output   Exr +MUnknown .Capitalized


M-Input    H20
M-Output   H20 +MUnknown .Alphanum
```

The morphological analyzer and the guesser operate simultaneously on the same token. The analyses of both components are joined to build a disjunctive set as described above. As an example, cf. the outline of the analysis of the sentence in (20), which contains the non-word *Exr* (*Exr* could, e.g., be a misspelt variant of the personal pronoun *Er* 'he'). (Again, the set members that figure in the successful analysis are set in italics.)
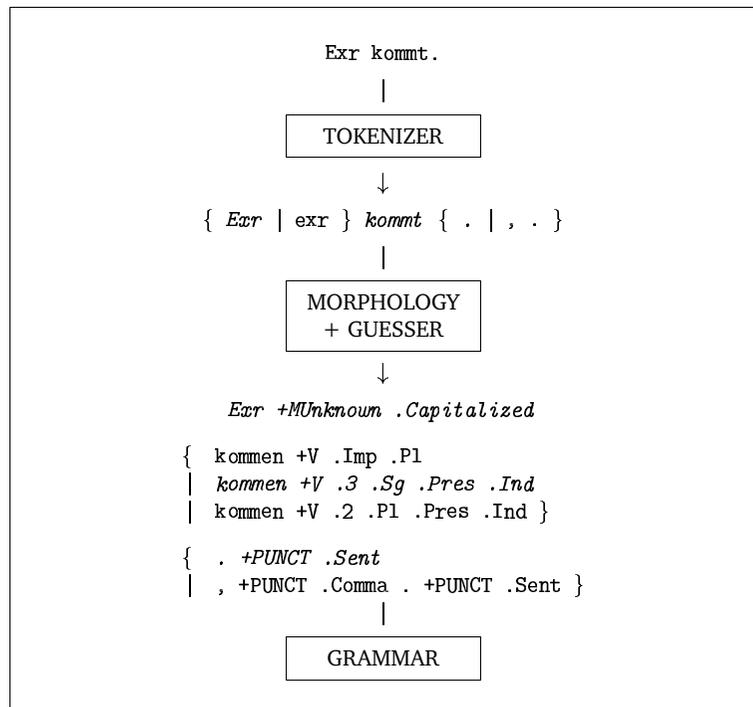
(20)   *Exr kommt.*
        ??   comes

```
                    Exr kommt.
                        |
                ┌───────────────┐
                │   TOKENIZER    │
                └───────────────┘
                        ↓
            { Exr | exr } kommt { . | , . }
                        |
                ┌───────────────┐
                │  MORPHOLOGY    │
                │  + GUESSER     │
                └───────────────┘
                        ↓
              Exr +MUnknown .Capitalized

        {  kommen +V .Imp .Pl
        |  kommen +V .3 .Sg .Pres .Ind
        |  kommen +V .2 .Pl .Pres .Ind }

        {  . +PUNCT .Sent
        |  , +PUNCT .Comma . +PUNCT .Sent }
                        |
                ┌───────────────┐
                │   GRAMMAR      │
                └───────────────┘
```

The token sequence *Exr +MUnknown .Capitalized* in the morphology/guesser output corresponds to the disjunctive input { Exr | exr }. The morphological analyzer does not supply an analysis for either one of the disjuncts, whereas the guesser in general only supplies analyses for capitalized words (e.g. 'Exr'). Hence, the decapitalized version *exr* neither figures in the output of the morphology nor the guesser.

3.2.4   Sublexical Rules

Traditional phrase structure grammars expect a sequence of words as input, e.g. *Karl kommt* 'Karl is coming'. Lexicon entries represent the interface between the individual words of the input string (e.g. *Karl*) and the preterminal categories of the grammar (e.g. N). A lexicon entry assigns each word a c-structure category, e.g. *Karl* is assigned the category N. The interaction of rules and lexicon entries allows the derivation of a tree containing the terminal *Karl*, which is dominated by the category N.

```
                NP
                |
                N
                |
              Karl
```

Categories such as N, V, P, etc. represent the actual preterminals in our grammar as well. However, as mentioned above, the input to the grammar are the tags from the morphological analyzer and guesser. This means that a more complex interface is needed, building a bridge between tags such as *+NPROP* and preterminals such as N. This interface is realized by so-called sublexical rules and tag lexicon entries for morphological tags (Kaplan and Newman 1997).

Tag entries   First, consider a traditional LFG lexicon entry. The entry for the terminal *Karls* 'Karl's' defines the dominating c-structure category (N) and the associated f-structure constraints.

```
Karls  N  (↑PRED) = 'Karl'
          (↑GEND) = masc
          (↑CASE) = gen
          (↑NUM) = sg
```

The morphological tags of *Karls* are:

```
M-Input    Karls
M-Output   Karl +NPROP .Masc .Gen .Sg
```

As can be seen, there is a close relation between the tags and the c-structure and f-structure constraints in the above entry. For instance, the lemma form 'Karl' corresponds to the PRED value; +NPROP predicts the category to be N; the inflectional features .Masc, .Gen, .Sg correspond to the respective f-structure features.

The tag lexicon entries below encode these relationships. Similar to ordinary entries, these entries list the terminals, i.e. the tags, and associate them with c-structure categories (N-S, N-T, etc.) and f-structure constraints. (+NPROP is different in that it plays a role only in c-structure via its category N-T, see below). We call the categories N-S etc. sublexical categories.

```
Karl    N-S     (↑PRED) = 'Karl'
+NPROP  N-T
.Masc   GEND-F  (↑GEND) = masc
.Gen    CASE-F  (↑CASE) = gen
.Sg     NUM-F   (↑NUM) = sg
```

The names of the sublexical categories hint at the "function" of the different types of tags: N-S is the category of **n**ominal **s**tems (= lemmas), N-T is the category of the **n**ominal part-of-speech **t**ag, the category GEND-F corresponds to the general **gend**er **f**eature (which is not restricted to nouns), etc.

Sublexical rules   The sublexical categories (N-S, N-T, etc.) represent the input to a special type of rules, the sublexical rules. These rules are similar to ordinary context-free rules in LFG, but usually, they only expand to sublexical categories and do not make use of recursion (i.e. the resulting "sublexical trees" are flat).

As an example, we present the N rule and the resulting (annotated) tree for the input tag sequence *Karl +NPROP .Masc .Gen .Sg* (corresponding to the original input *Karls*).

```
N →  N-S   N-T   GEND-F  CASE-F  NUM-F
     ↑=↓   ↑=↓    ↑=↓     ↑=↓     ↑=↓
```
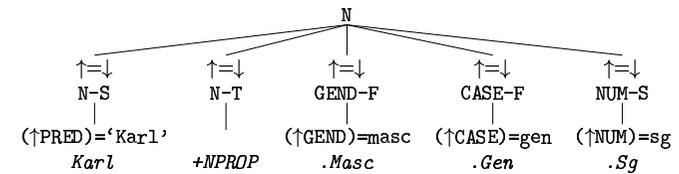
The following f-structure is the f-structure projected by the sublexical rules and tag lexicon entries above. Note that the full-form lexicon entry of *Karls* 'Karl's' (shown above) projects the same f-structure, which is the desired result.

$$\begin{bmatrix} \text{PRED} & \text{'Karl'} \\ \text{GEND} & \text{masc} \\ \text{CASE} & \text{gen} \\ \text{NUM} & \text{sg} \end{bmatrix}$$

Furthermore, the category N, the root of the sublexical tree built on the morphological input, is available in the c-structure representation, and can be used to build complex trees, derived by genuine c-structure grammar rules. Again, this is the same configuration as in the case of the (full form) lexicon entry for *Karls*. To sum up, our tag entries and sublexical rules, taking as their input the analyses of an external morphological component, fully compensate for a full-fledged lexicon of inflected words.

Note that morphological structure is commonly assumed to be less complex than syntactic structure. Hence it might seem inappropriate to apply LFG-like rules and annotations in the analysis of morphological structure. However, this approach has the advantage that the grammar writer can easily create and modify the morphology-syntax interface, without the need for learning a new formalism.

A note on XLE notation   Tag entries and sublexical rules in XLE require a special notation. Tag entries contain the keyword 'xle' (or 'XLE') following the sublexical category. The entry ends with a full stop, cf the entries of *Karl* 'Karl' and *Hund* 'dog'.

```
Karl N-S xle (↑PRED) = 'Karl'.
Hund N-S xle (↑PRED) = 'Hund'.
```

Sublexical rules append the suffix '_BASE' to the sublexical categories. (The trivial (default) annotation ↑=↓ can be omitted in XLE notation.)

```
N  →   N-S_BASE
       N-T_BASE
       GEND-F_BASE
       CASE-F_BASE
       NUM-F_BASE.
```

As a first example, we show the sublexical tree that our grammar constructs for the input in (21).

(21)   *Hund*
        dog

```
        CS 1:       N[comm]


N-S_BASE N-T_BASE GEND-F_BASE CASE-F_BASE NUM-F_BASE
   |        |         |           |           |
  Hund     +NN       .Masc       .NDA        .Sg
```
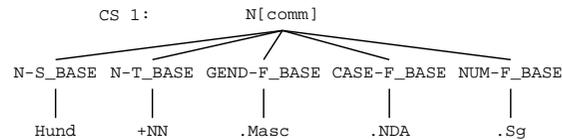
In the default display mode of the c-structure, XLE displays only the output tokens of the tokenizer as terminal nodes, cf. the analysis of (22). On closer inspection, the N[comm] node is expanded to the so-called sublexical tree above, where the actual morphological input is shown in the leaves of the sublexical tree.

(22)   *Ein Hund kommt.*
        a   dog   comes
        'A dog is coming.'

```
        CS 1:       ROOT


            CProot[std] PERIOD
                          |
        DP[std]    Cbar    .


        DPx[std]  V[v,fin]


        D[std]  NP  Vx[v,fin]
          |      |     |
        ein N[comm] kommt
                |
              Hund
```
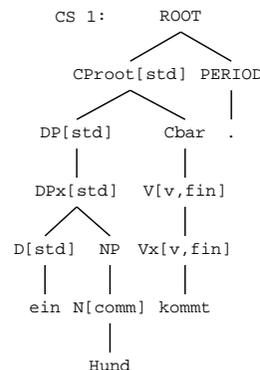
In contrast to the above presentation, we make use of special sublexical categories for names, NAME-S, NAME-T, in our implementation. The mother node is NAMEP (name phrase), cf. the example in (23). (The examples in the following section also assume NAME-S and NAME-T as the sublexical categories of proper nouns.)

(23)   *Karls*
        Karl[GEN]

```
        CS 1:              NAMEP


NAME-S_BASE NAME-T_BASE GEND-F_BASE CASE-F_BASE NUM-F_BASE
     |          |           |           |          |
   Karl       +NPROP      .Masc        .Gen       .Sg
```
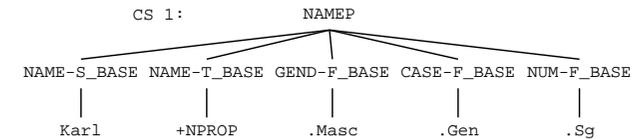
Finally, see the sublexical tree for guessed proper nouns, as in (24).

(24)   *Exr*
        ??

```
        CS 1:          NAMEP


    DEFAULT-S_BASE MUNKN-T_BASE CAP-F_BASE
         |             |            |
        Exr         +MUnknown   .Capitalized
```
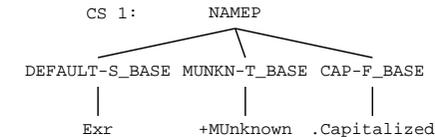
Default lexicon entries for lemma tags   As described, the tag lexicon specifies entries for morphological tags, e.g. for the lemma tag 'Karl', the part-of-speech tag +NPROP, and inflectional tags such as .Masc.

```
Karl    xle NAME-S    (↑PRED) = 'Karl'.
+NPROP  xle NAME-T .
.Masc   xle GEND-F    (↑GEND) = masc.
```

However, there is a fundamental difference between the tags encoding the lemma vs. the part of speech or inflection: our morphological analyzer specifies 34 different part-of-speech tags and 70 different inflectional tags. In contrast, there are several thousands of different lemma tags that are provided by our morphology (e.g., about 6,800 lemma tags for verbs, 23,850 lemma tags for nouns, etc.).

In the approach presented so far, thousands of lexicon entries would have to be written.

```
Karl   NAME-S xle  (↑PRED) = 'Karl'.
Otto   NAME-S xle  (↑PRED) = 'Otto'.
Maria  NAME-S xle  (↑PRED) = 'Maria'.
 ...
```

Instead of writing lexicon entries for each single lemma tag, default entries are provided by XLE that interface with the morphololgical output. One entry covers capitalized tags (e.g. for lemma tags of common and proper nouns), the other lower-case tags (e.g. for lemma tags of verbs and adjectives).

We first show the entry for capitalized words. The keyword '–LUnknown' matches any capitalized tag that is "unknown to the lexicon", i.e. that is not listed in any lexicon resource. The entry then simply says that this can be a noun stem (N-S) or a name stem (NAME-S) (the semicolon here denotes a disjunction). In both cases, a feature PRED is introduced. The keyword '%stem' is a variable whose value is determined by the token matching the keyword –LUnknown.

```
-LUnknown  N-S     xle (↑PRED) = '%stem';
           NAME-S  xle (↑PRED) = '%stem'.
```

For instance, suppose the lemma tag *Karl* is unknown to the lexicon. It then automatically matches –LUnknown, and '%stem' is instantiated as 'Karl', resulting, e.g., in the following entry (which can be imagined as a virtual lexicon entry).

```
Karl  NAME-S  xle (↑PRED) = 'Karl'.
```

The reader may wonder how the lemma tag of the name *Karl* is prevented from making use of the N-S entry. In fact, this cannot be prevented. However, as soon as the sublexical rule for the category N tries to match the tag sequence provided by the morphological analyzer, the part-of-speech tag +NPROP and its sublexical category NAME-T will be incompatible with the expansion of the category N. Only the rule of category NAMEP will succeed, cf. the rules of N and NAMEP below.

```
      N  →  N-S_BASE              NAMEP  →  NAME-S_BASE
            N-T_BASE                        NAME-T_BASE
            GEND-F_BASE                     GEND-F_BASE
            CASE-F_BASE                     CASE-F_BASE
            NUM-F_BASE.                     NUM-F_BASE.
```

Compare the sublexical trees that our grammar constructs for a common noun (25) and a proper noun (26).

(25) *Hunds*
      dog[GEN]
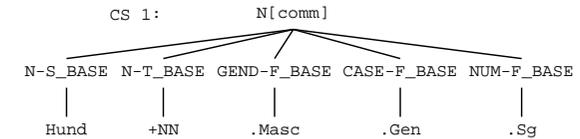
(26) *Karls*
      Karl[GEN]
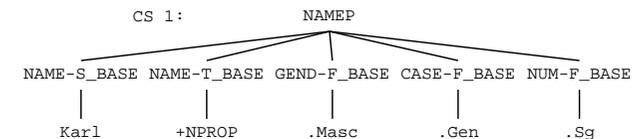
The second default entry, –Lunknown (note the lower case 'u'), matches lower-case tags, e.g. lemmas of adjectives, adverbs, or verbs. In this example, verbs that are not listed in the lexicon are guessed to be intransitive ((↑PRED) = '%stem<(↑SUBJ)>') or transitive ((↑PRED) = '%stem<(↑SUBJ)(↑OBJ)>').

```
-Lunknown  A-S    xle  (↑PRED) = '%stem';
           ADV-S  xle  (↑PRED) = '%stem';
           V-S    xle  { (↑PRED) = '%stem<(↑SUBJ)>'
                       | (↑PRED) = '%stem<(↑SUBJ)(↑OBJ)>'
                       (↑OBJ CASE) = acc }.
```

3.2.5   Lexicon

Up to now, we have discussed lexicon entries for tags (cf. sec. 3.2.4). The task of these entries is to "translate" the morphological tags into c-structure and f-structure constraints, thus implementing a general interface between the morphological analysis and the grammar.

Besides these tag entries, there are lexicon entries in the traditional sense, i.e. entries that specify idiosyncratic lexical information such as subcategorization properties. In our approach, however, this type of information is not associated with full-form entries as in traditional LFG but rather with lemma tags, cf. the example entry for *lesen* 'read' (the keyword %stem again copies the value of the matching lemma, i.e. 'lesen' in this case).

```
lesen V-S xle (↑PRED) = '%stem<(↑SUBJ)(↑OBJ)>'.
```

Other idiosyncratic information concerns nouns. Nouns can, e.g., be marked as count nouns vs. mass nouns (this information is used by the grammar rules to determine the use of specifiers). Proper nouns can specify the name type, first vs. last name, geographical name, etc. Example entries are given for the count noun *Hund* 'dog', the mass noun *Wasser* 'water', the first name *Hans*, and the geographical name *Stuttgart*.

```
Hund        N-S xle  (↑PRED) = '%stem'
                     (↑NOUN-TYPE) = count.

Wasser      N-S xle  (↑PRED) = '%stem'
                     (↑NOUN-TYPE) = mass.

Hans        N-S xle  (↑PRED) = '%stem'
                     (↑NSEM PROPER) = first_name.

Stuttgart  N-S xle  (↑PRED) = '%stem'
                     (↑NSEM PROPER) = city.
```

Usually, such lexicon entries are being developed independently from the grammar and are stored in an external module. We call this resource the "stem lexicon", since it associates stem entries with idiosyncratic information.

Full-form entries   XLE also allows for traditional, full-form lexicon entries. For instance, the German morphology does not handle multi-word expressions. Hence they must be listed as full forms in a lexicon. Full-form entries contain an asterisk '*' instead of the keyword 'xle'. Since full-form entries "skip" the morphological interface, they assign syntactic categories, such as N or ADV, to the full form, rather than sublecixal categories like N-S or ADV-S. As an example, cf. the entry for *in der Regel* 'usually' (spaces within multi words must be quoted by ').

```
in' der' Regel ADV * (↑PRED) = '%stem'.
```

A note on XLE notation   The default entries –LUnknown and –Lunknown do, in fact, match any tag, not only the ones that are unknown to the lexicon. This can be prevented by adding the keyword 'ONLY' to a lexicon entry (Kaplan and Newman 1997). For instance, if the verb *geben* 'give' is listed in the (stem) lexicon as ditransitive as shown in the following entry, marked with ONLY, the "guessed" intransitive and transitive readings from the –Lunknown entry will be ignored/overwritten.

```
geben  V-S xle  (↑PRED) = '%stem<(↑SUBJ)(↑OBJ)(↑OBJ2)>'
                (↑OBJ CASE) = acc
                (↑OBJ2 CASE) = dat; ONLY.
```

In contrast, the keyword 'ETC' allows for additional matches. Note that there may be several lexicon entries for one head word simultaneously (e.g. spread over different files), so 'ETC' must be used whenever other entries must not be overwritten ('ONLY' is the default).

Another way of overwriting or allowing for information from other lexicon entries is the following. The sublexical categories in an entry, e.g. V-S, can be prefixed by +, –, !, or =. For instance, in the following (nonsense) entry '+V-S' means: add the following f-structure annotation as another alternative to the entries of *geben* of category V-S (if there are any). '–ADV-S' means: delete all other entries for *geben* of category ADV-S. '!A-S' means: replace all other entries for *geben* of category A-S by the current one. Finally, '=N-S' means: keep all entries for *geben* of category N-S.

```
geben  +V-S xle     (↑PRED) = '%stem<(↑SUBJ)(↑OBJ)(↑OBJ2)>';
       -ADV-S xle;
       !A-S xle     (↑PRED) = '%stem<(↑SUBJ)>';
       =N-S xle.
```

3.3   XLE Notation

This section gives an overview of XLE-specific notation in grammar rules, including means of abstraction such as macros. The overview is restricted to XLE features that are made use of in the implementation of the German grammar. In the presentation, we follow the XLE documentation very closely, which is delivered together with the XLE system (URL: `http://www.parc.com/istl/groups/nltt/xle/`).

3.3.1   Rule Notation (Basics)

Grammar rules in XLE are encoded in pure ASCII code. As an example, cf. the following rule.

```
CP --> DP: (^SUBJ) = !;
        Cbar: ^=!.
```

The f-structure annotation follows the c-structure node, separated by a colon. '^' denotes the ↑-arrow, '!' the ↓-arrow. The f-structure annotations pertaining to one category end with a semicolon, a rule ends with a full stop. (In certain cases, the semicolon can be omitted, e.g. in front of the full stop.)

The following rule is equivalent to the above rule. That is, the layout of the rule does not play a role.

```
CP --> DP: (^SUBJ)=!; Cbar: ^=!.
```

If no f-structure annotation is stated (or if the f-structure annotation does not refer to ↓), the default annotation ↑=↓ is assumed, i.e. the following rule is equivalent to the above as well.

```
CP --> DP: (^SUBJ) = !;
       Cbar.
```

The dollar sign '$' denotes set membership, i.e. the f-structure denoted by ↓ is an element of the set-valued feature ADJUNCT.

```
VP --> PP: ! $ (^ADJUNCT);
       VP.
```

Comments are enclosed in double-quotes.

```
VP --> PP: ! $ (^ADJUNCT) "adjunct PP";
       VP "head".
```

Regular expressions in c-structure rules   The right hand side of a rule consists of a regular expression. The examples from above represent the simplest case, namely concatenation of categories. Other possible regular expressions can be constructed as follows (examples of the more unusual operations are given below).

Operation	Notation	Comment
optionality	(X)	
Kleene star	X*	arbitrarily many X (including none)
Kleene plus	X+	at least one occurrence of X
iteration	X#0#2	between zero and two X
disjunction	{ X | Y }	X or Y
intersection	X & Y	X and Y
complementation	X – Y	X and not Y
shuffle	X , Y	X and Y in any order
empty string	e	
any category appearing anywhere else in the rule	?	
grouping	[X]	

Regular expressions: grouping, shuffle, iteration   The following toy rule illustrates the use of grouping, shuffle, and iteration.

```
DP -->  (D)
        NP
        [ [ PP: ! $ (^ADJUNCT); ]#0#2
        ,
          [ DP: ! $ (^ADJUNCT); ]#0#1
        ].
```

The DP rule expands to an optional determiner (D) and a category NP (for the omitted Dbar projection (cf. p. 64)). In addition, it can generate up to two PPs ('#0#2') and up to one DP ('#0#1'). The PPs and the DP may be permutated (shuffled). Example DPs are given in (27), (28).

(27) *eine Anfrage   der   Eltern   im   Landratsamt   nach*
      a   request [the   parents DP]   [in the   district office PP1] [for
      Meßwerten
      measure values PP2]
      'a request for measured values of the parents in the District Office'

(28) *das Ministerium für Arbeit   des Landes NRW*
      the ministry   [for labour PP] [the state   NRW DP]
      'the Ministry of Labour of the state NRW'

The above rule is equivalent to the following.

```
DP -->  (D)
        NP
        { e                     "empty string"

        | PP: ! $ (^ADJUNCT);   "PP"
        | DP: ! $ (^ADJUNCT);   "DP"

        | PP: ! $ (^ADJUNCT);   "PP PP"
          PP: ! $ (^ADJUNCT);
        | PP: ! $ (^ADJUNCT);   "PP DP"
          DP: ! $ (^ADJUNCT);
        | DP: ! $ (^ADJUNCT);   "DP PP"
          PP: ! $ (^ADJUNCT);

        | PP: ! $ (^ADJUNCT);   "PP PP DP"
          PP: ! $ (^ADJUNCT);
          DP: ! $ (^ADJUNCT);
```

```
  | PP: ! $ (^ADJUNCT);   "PP DP PP"
    DP: ! $ (^ADJUNCT);
    PP: ! $ (^ADJUNCT);
  | DP: ! $ (^ADJUNCT);   "DP PP PP"
    PP: ! $ (^ADJUNCT);
    PP: ! $ (^ADJUNCT);
  }.
```

Regular expressions: complementation   The use of complementation is
demonstrated by the next (nonsense) rule. The expression '{ PP | DP }*' allows
for arbitrarily many PP and DP adjuncts (including none), in any order. The
expression '[...] - e', however, substracts the empty string, i.e. it enforces the
presence of at least one adjunct.

```
DP --> (D)
       NP
       [ { PP: ! $ (^ADJUNCT);
         | DP: ! $ (^ADJUNCT); }*
       ] - e.
```

Regular expressions: the empty string   The empty string 'e' is also often
used to encode f-structure constraints that do not emerge from a specific cat-
egory but belong to the rule as a whole, cf. the DP rule below. The constraint
attached to the empty string expresses the generalization that all German DPs
are third person singular.

```
DP --> e: (^PERS) = 3;
       (D)
       NP.
```

Regular expressions: intersection   Suppose we want to write a rule for
extraposed constituents, which captures the following (toy) data.

- Constituents that can be extraposed are: relative clauses (of category
  CPrel), complement VPs (VP), complement clauses (CP), adverbial claus-
  es (CPconj).

- Extraposed constituents are enclosed by commas, e.g. 'COMMA CPrel
  COMMA' (cf. p. 33).

- The relative order of the constituents is fixed: CPrel > VP > CP > CPconj.

- Any extraposed constituent is optional. At most two constituents occur
  simultaneously (this restriction improves performance).

These data can be captured in an elegant way by means of the intersec-
tion operator ('&'). The effect of the intersection is to allow for at most two
extraposed constituents.

```
CP -->
    DP: (^SUBJ) = !;
    Cbar
    [ [ "first intersected expression"
        ( COMMA CPrel: ! $ (^ADJUNCT);   COMMA ) "relative clause"
        ( COMMA VP: (^VCOMP) = !;        COMMA ) "complement VP"
        ( COMMA CP: (^COMP) = !;         COMMA ) "complement clause"
        ( COMMA CPconj: ! $ (^ADJUNCT); COMMA ) "adverbial clause"
      ]
      &
      [ "second intersected expression"
        COMMA ? COMMA
      ]#0#2
    ].
```

We start with the second of the intersected expressions. '?' within the expres-
sion '[COMMA ? COMMA]' matches any category appearing anywhere else in
the rule. That is, '?' can be replaced by a disjunction of all categories appearing
in that rule: { DP | Cbar | COMMA | CPrel | VP | ... }.

That is, the expression '[COMMA ? COMMA]' matches any of the triples
<comma, extraposed constituent, comma>. (Due to the intersection with the
first expression, the instantions of '?' by DP, Cbar, or COMMA are ruled out.)

Now, the second expression is restricted by '#0#2' to at most two repeti-
tions. The intersection operator transfers this restriction to the first expression,
i.e. at most two extraposed constituents are effectively allowed.

(Note that the relative order of the extraposed constituents is fixed since
the constituents are concatenated. Hence, a relative clause must precede an
argument clause, and an argument clause must precede an adverbial clause.
In contrast, the above DP rules allow for any order of postnominal PP and DP
adjuncts, by the shuffle operator (cf. p. 53) or a disjunction (cf. p. 54).)

F-structure designators   Elements of f-structures are denoted/referred
by so-called f-structure designators. Simple designators are attributes (CASE,
SUBJ), values (nom), and the arrows ↑ and ↓. Complex designators are built on
the base of simple designators, e.g. (↓CASE) or (↑XCOMP* OBJ CASE).

Note that there are also a few designators referring to c-structure ele-
ments. '*' denotes the current c-structure node. 'M*' and the equivalent '(*
MOTHER)' denote the mother node of *. Finally, '(* RIGHT_SISTER)' and '(*
LEFT_SISTER)' denote the right/left sisters of *.

The c-structure designators can be preceded by terms referring to other projections such as the f-structure projection. 'f::' denotes the f-structure projection, 'o::' the projection of optimality marks (cf. sec. 3.3.5). 'f::*' (the combination of 'f::' and '*') then denotes the f-structure projected by the current node; 'f::M*' denotes the f-structure of the mother node. That is, '↑' and '↓' are actually abbreviations of 'f::M*' and 'f::*'. Similarly, 'o::*' denotes the o-structure of the current node.

Constraints on f-structure designators   A basic f-structure annotation consists of an expression that asserts certain properties of a designator, e.g. that a designator is present in a certain f-structure (existential constraint, e.g. (↑CASE)) or that a designator is equal to another designator (e.g. ↑=↓).

Important properties that can be asserted of a designator are shown in the following table.

Constraint	Notation	Example
defining equality	=	^=!
constraining equality	=c	(^CASE) =c nom
set membership (∈)	$	! $ (^ADJUNCT)
existence		(!CASE)

In addition, there are truth-value constants 'TRUE' and 'FALSE'. For instance, a disjunct can be deactivated in the following way (for the notations of disjuncts, see below):

```
DP --> { D
      | e: FALSE }
      NP.
```

In this example, the option of determinerless DPs is ruled out due to the annotation via FALSE. (The constants TRUE and FALSE are very useful in debugging.)

More complex constraints are presented in the next section (sec. 3.3.2).

Boolean combinations of f-structure constraints   F-structure annotations often consist of more than one designator constraint. Complex annotations can be constructed by boolean combinations of constraints.

Combination	Notation	Example
conjunction		DP: (^OBJ) = ! (!CASE) = acc
disjunction	{ ... | ... }	DP: { (^SUBJ) = ! | (^OBJ) = ! }
optionality	{ ... }	VP: ^=! { "optional 'pro'-SUBJ" (!SUBJ PRED) = 'pro' } which is equivalent to: ^=! { (!SUBJ PRED) = 'pro' | TRUE }
negation	~	DP: (^OBJ) = ! (!CASE) ~= nom "no nom." or equivalently: ~(!CASE) = nom "no nom."
grouping	[...]	DP: (^SUBJ) = ! ~ [(!CASE) = gen "no genitive" (!CASE) = dat "or dative"]

Note that a double-negated equation is equivalent to the corresponding constraining equality (=c), i.e. the following equations are equivalent.

$$(\text{^CASE}) =c \text{ gen} \qquad (\text{^CASE}) \sim\sim= \text{ gen}$$

3.3.2   Rule Notation (Advanced)

Having described the basic features of the XLE notation, we now present more complex constraints on designators, as provided by XLE.

Instantiated feature values   Remember that by definition, semantic forms (i.e. features whose values are quoted) can never unify (cf. p. 17). XLE provides a special notation for additional simple, non-semantic features that the grammar writer does not want to unify. Such features receive values that end with an underscore, e.g. [PRON-FORM es_] (for non-semantic expletives, e.g. *es* 'it'). Such values are called "instantiated feature values". (Note that features often allow for ordinary and instantiated values.)

Head precedence   The head precedence relation is encoded as '<h' and '>h' (f1 <h f2 is true if the semantic head of f1 precedes the semantic head of f2). For instance, a relative clause (functioning as ADJ-REL) can be annotated as follows:

```
... CPrel: (^ADJ-REL) = !
             ^ <h !
```

The head-precedence constraint requires that (the semantic head of) the modified noun (denoted by ↑) precede the relative clause (denoted by ↓).

Surface scope   The surface scope relation is encoded as '$<h>s' (f1 $<h>s f2 is true if f1 ∈ f2 and f1 "has scope over" all other set members which follow f1). For instance, in coordinations, as in (29), the conjuncts can be annotated as follows:

```
DP --> DP: ! $<h>s ^;
       CONJ
       DP: ! $<h>s ^.
```

(29)   *Hans und Maria lachen.*
         H.   and M.   laugh

```
                    "Hans und Maria lachen."

              ⎡PRED     'lachen<[17]>'              ⎤
              ⎢                  ⎧ ⎡PRED 'Hans'   ⎤ ⎫⎥
              ⎢SUBJ           0⎢ >s   ([40:Maria])⎥  ⎥
              ⎢               ⎨ ⎣               ⎦  ⎬⎥
              ⎢            17⎩40⎡PRED 'Maria']    ⎭⎥
              ⎢56 TOPIC      [17]                   ⎥
              ⎣                                     ⎦
```

That is, the f-structures of the conjuncts are analyzed as the members of a set (which in turn may represent the value of a feature such as SUBJ).

Since *Hans* precedes *Maria* (<h), *Hans* "has scope over" *Maria* ('>s'). (What >s effectively does in this case is to encode the relative surface order of all set members.) In the f-structure of *Hans*, this is indicated by the relation '>s [Maria]', cf. the f-structure of the above example.

The relation $<h>s is especially useful in generation. Without the relation '>s', the f-structures of (30) and (31) would be identical. Hence, if the generator were applied to f-structures of a coordinated phrase, it would always generate all permutations of the conjunctions.

(30)   *Hans und Maria*
         H.   and M.

(31)   *Maria und Hans*
         M.   and H.

Membership   The symbol of membership has been presented above: '$'. Usually, the designator to the left of $ is said to be a member of the set-valued designator to the right of $, e.g. as in the following example.

```
VP --> PP: ! $ (^ADJUNCT)
       VP.
```

However, in certain cases, the relation can be switched, namely whenever the $ sign appears within a designator, i.e. within a feature path. That is, $ then stands for the relation ∋ rather than the usual ∈. For instance, an extraposed relative clause can be annotated as follows.

```
... CPrel: (^ {SUBJ|OBJ} ADJUNCT $ OBJ ADJ-REL) = !
```

With this annotation, a member of the set embedded by ADJUNCT is chosen non-deterministically. This member is said to contain a feature OBJ ADJ-REL whose value is provided by the f-structure of the relative clause.

As an example, cf. the f-structure of (32).

(32)   *Das Reisen in die Länder   wird   bevorzugt, die   beitreten.*
         the travel [in the countries PP] becomes preferred [that enter CP]
         'Trips to those countries that will join (the union) are preferred.'

```
"Das Reisen in die Länder wird bevorzugt, die beitreten."

⎡PRED     'bevorzugen<NULL, [0:reisen]>'                                          ⎤
⎢         ⎡PRED     'reisen'                                                    ⎤ ⎥
⎢         ⎢        ⎧ ⎡PRED     'in<[132:Land]>'                            ⎤ ⎫ ⎥
⎢         ⎢        ⎪ ⎢        ⎡PRED      'Land'                          ⎤ ⎪ ⎥
⎢         ⎢        ⎪ ⎢        ⎢          ⎡PRED      'bei#treten<[300:pro]>'⎤⎥ ⎪ ⎥
⎢SUBJ     ⎢ADJUNCT ⎨ ⎢        ⎢ADJ-REL   ⎢SUBJ    300⎡PRED 'pro']          ⎥⎥ ⎬ ⎥
⎢         ⎢        ⎪ ⎢OBJ     ⎢          ⎢PRON-REL    [300:pro]            ⎥⎥ ⎪ ⎥
⎢         ⎢        ⎪ ⎢        ⎢      367⎣TOPIC-REL    [300:pro]            ⎦⎥ ⎪ ⎥
⎢         ⎢        ⎪ ⎢        ⎣SPEC     ⎡DET ⎡PRED 'die']⎤                 ⎦ ⎪ ⎥
⎢         ⎢        ⎩ ⎣109  132⎣         ⎣   ⎣          ⎦⎦                 ⎭ ⎥
⎢         ⎢ 0 SPEC  ⎡DET ⎡PRED 'die']⎤                                        ⎥
⎢215 TOPIC [0:reisen]                                                            ⎥
⎣                                                                                ⎦
```

The relative clause in the example functions as a modifier (ADJ-REL) of a noun (*Land* 'country'), which is the complement (OBJ) of a preposition (*in* 'in'). The PP *in die Länder* 'to those countries' functions as a modifier, i.e. as a member of the ADJUNCT set. That is, if there are several members in the ADJUNCT set, the relative clause can in principle choose either one as its antecedent.

Local variables %VAR   In the previous example, a relative clause non-deterministically chooses its antecedent from an ADJUNCT set. There are, however, additional restrictions on relative clauses in German: the relative pronoun and its antecedent must agree in number and gender.

For instance, the modifier *per Zug* 'by train[MASC,SG]' in (33) cannot serve as the antecedent of the relative pronoun *die* 'that[PL]'.

(33)   *Das Reisen per Zug   in die Länder   wird   bevorzugt, die*
       the travel [via train PP] [in the countries PP] becomes preferred [that
       beitreten.
       enter CP]
       'Trips by train to those countries that will join (the union) are pre-
       ferred.'

Hence, agreement equations have to be added to the above rule.

```
... CPrel: (^ {SUBJ|OBJ} ADJUNCT $ OBJ ADJ-REL) = !
           (^ {SUBJ|OBJ} ADJUNCT $ OBJ NUM) = (!NUM)
           (^ {SUBJ|OBJ} ADJUNCT $ OBJ GEND) = (!GEND)
```

The two new equations, however, do not impose the desired restrictions. The problem is that in each of the three annotation equations, the chosen set member can be a different one (since it is chosen non-deterministically). The intention of the equations, though, is to find one antecedent and check for the values of the features NUM and GEND of that one antecedent.

The solution implemented in XLE is the following. First, a set member is chosen non-deterministically. The label of the chosen f-structure is stored by a so-called "local variable" (or "local name"), which is prefixed by '%', e.g. %AN-TECEDENT. The variable keeps record of the current chosen path, so that the same path can be reused in different equations. The variable then can be used like an ordinary, deterministic designator.

The final version of the relative clause then looks as follows.

```
... CPrel: (^ {SUBJ|OBJ} ADJUNCT $ OBJ) = %ANTECEDENT
           (%ANTECEDENT ADJ-REL) = !
           (%ANTECEDENT NUM) = (!NUM)
           (%ANTECEDENT GEND) = (!GEND)
```

C-structure constraints   As mentioned above, there are a few designators referring to c-structure elements, e.g. * and (* RIGHT_SISTER), which we make use of in our implementation. For instance, we allow determiners in general to project full DPs on their own, i.e. the NP category is optional as shown in the Dbar rule.

```
Dbar -->
   D
   (NP).
```

However, certain so-called "attributive" determiners always need an NP sister. The f-structures of these determiners are marked by a feature [CHECK _SPEC-TYPE _DET attr]. The restriction can then be encoded as follows.

```
Dbar -->
   D: { ~(^CHECK _SPEC-TYPE _DET)
      | (^CHECK _SPEC-TYPE _DET) =c attr
        (* RIGHT_SISTER) );
   (NP).
```

The disjunct checks for the presence of the feature that marks "attributive" determiners. If it is contained in the f-structure of Dbar, then the category D (denoted by '*') is required to have a right sister node, in this case, an NP node.

3.3.3   Means of Abbreviation in C-Structure

XLE provides various means of abbreviation, e.g. macros. Such means of abbreviation have advantages from a technical and a linguistic point of view. They can be used to increase code transparency by bundling groups of constraints. On the other hand, they can be used to express linguistic generalizations.

In this section, abbreviations applying to the c-structure representation are presented. The next section deals with f-structure abbreviations (sec. 3.3.4).

3.3.3.1   Macros

Code transparency   Macros can be used to bundle a number of categories. For instance, we make use of a macro called DPpost to encode all kinds of postnominal modifiers. A simplified version of the macro is given below.

```
DPpost =
    ( DP: ! $ (^ADJUNCT)      "optional genitive DP"
         (!CASE) = gen )
    ( PP: ! $ (ADJUNCT)       "optional PP" )
    [ COMMA
    { CPrel: ! $ (^ADJ-REL)   "relative clause"
    | VP: (^VCOMP) = !         "complement VP"
    | CP: (^COMP) = !          "complement clause"
    }
      COMMA ]#0#2.
```

According to the macro, postnominal modifiers include an optional geni-
tive DP, which is followed by an optional PP. Finally, up to two sentential con-
stituents, enclosed by commas, may occur.

The macro is called in the DP rule. (Macro calls are marked by '@'. When
we refer to macros and templates in the text, we also mark them by '@'.)

```
DP -->
    (D)
    NP
    @(DPpost).
```

Whenever a macro is called, the content of the macro is copied to the place
of the macro call. That is, the above rule which makes use of the macro is
identical to the following one.

```
DP -->
    (D)
    NP
    ( DP: ! $ (^ADJUNCT) "optional genitive DP"
          (!CASE) = gen )
    ( PP: ! $ (^ADJUNCT) "optional PP" )
    [ COMMA
    { CPrel: ! $ (^ADJ-REL)  "relative clause"
    | VP: (^VCOMP) = !        "complement VP"
    | CP: (^COMP) = !         "complement clause"
    }
      COMMA ]#0#2.
```

This example illustrates that the structure of the DP rule is much easier to
grasp if the postnominal modifiers are "hidden" by a macro.

Linguistic generalizations   A further advantage is that macros can be
"reused" in different places. This is illustrated by the coordination of equal cat-
egories, e.g. the coordination of DPs or the coordination of PPs ("same-category
coordination"). The respective DP and PP rules may look as follows.

```
DP --> { (D)                    PP --> { PREP
        NP                              DP: (^OBJ) = !;
      | "coordination"                | "coordination"
        DP: ! $ ^;                      PP: ! $ ^;
        CONJ                            CONJ
        DP: ! $ ^                       PP: ! $ ^
      }.                              }.
```

That is, the basic coordination structure is always the same: XP CONJ XP
(with XP = DP, PP, . . .). This linguistic generalization can be captured by means
of a parametrized macro @COORD. The macro definition contains a parameter
_cat whose value is determined by the macro call.

```
COORD(_cat) =
    _cat: ! $ ^;
    CONJ
    _cat: ! $ ^.
```

The modified DP and PP rules are shown below. The macro call @(COORD
DP) specifies the value of the parameter _cat as DP. That is, when the content
of the macro is copied to the place of the macro call, each occurrence of _cat
within the macro definition is replaced by 'DP' (resulting in the original DP rule
from above).

```
DP --> { (D)                    PP --> { PREP
        NP                              DP: (^OBJ) = !;
      | "coordination"                | "coordination"
        @(COORD DP)                     @(COORD PP)
      }.                              }.
```

(Note that XLE provides a so-called meta-rulemacro which helps to express
the generalization about coordination even more adequatly. Rather than calling
the macro @COORD in each rule separately, the meta-rulemacro can be used,
which enumerates all rules that allow for the coordination macro.)

Maintainability   Macros also ease the maintenance of the grammar code.
For instance, the grammar writer might decide to allow for multiple coordinat-
ed conjuncts as in (34).

(34)   *Hans, Maria und Otto*
       H.   M.   and O.
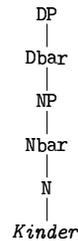   'Hans, Maria, and Otto'

The only part of grammar code that needs to be changed is the macro definition.
The modification automatically carries over to the PP coordination, allowing for
multiple PP coordination as in (35).

```
COORD(_cat) =
  [ _cat: ! $ ^;
    COMMA ]*
    _cat: ! $ ^;
    CONJ
    _cat: ! $ ^.
```

(35)   *in Stuttgart, in Frankfurt und in Berlin*
        in Stuttgart in Frankfurt and in Berlin
        'in Stuttgart, in Frankfurt, and in Berlin'

Flat c-structures   In our implementation, we also make use of macros to
flatten the c-structure representation in certain cases. For instance, our analysis implements a DP analysis. A full c-structure analysis of the noun *Kinder*
'children' as in (36) may look as follows.

(36)   *Kinder   lachen.*
        children laugh
        'Children are laughing.'

```
            DP
            |
           Dbar
            |
            NP
            |
           Nbar
            |
            N
            |
          Kinder
```

Rather than producing such long chains of categories, we employ macros
(which do not project c-structure nodes). For instance, the DP rule calls a macro
@Dbar in our implementation, rather than introducing a category Dbar.

```
DP --> { ...
        | @(Dbar) }.

Dbar = D
       NP.
```
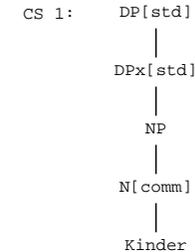
This analysis is a compromise between linguistic assumptions about (the
category) Dbar and the request for flatter structures (to increase readability).

Note that in our implementation, we assume additional, intermediate projections, e.g. DPx, for technical reasons (the main reason for introducing the
category DPx is to encode the special template @COMPLETE (cf. p. 71)).
Hence, the use of macros, such as @Dbar, may partially compensate for the
increased depth of the trees which is due to these additional projections.

As an example tree of our implementation, cf. the c-structure analysis of
(37).

(37)   *Kinder*
        children

```
        CS 1:     DP[std]
                     |
                  DPx[std]
                     |
                    NP
                     |
                  N[comm]
                     |
                   Kinder
```

To sum up, the examples show that the use of macros may not only improve
code transparency and maintainability but also allows for the encoding of linguistic generalizations. Moreover, macros can be used to flatten the c-structure
representation.

3.3.3.2   Meta-Categories

In contrast to macros, meta-categories typically do not replace bundles of categories, but just a single one. We make use of meta-categories in the following
case only. For efficiency reasons, we distinguish different c-structure types of
VPs, e.g. VP[v], VP[cop], and VP[coh] (so-called "complex categories", see below). VP[v] encodes VPs headed by a full verb, VP[cop] are VPs headed by a
copula verb, and VP[coh] is headed by a coherent verb.

Now, whenever a VP is introduced in a rule, the three VP types have to be
enumerated. As an example, see the following toy rule introducing topicalized
and extraposed VPs. (Topicalized constituents occupy the specifier position of
the CP projection in German.)

```
CP --> "topicalized constituents"
       { DP: { (^SUBJ) = !;
              | (^XCOMP* OBJ) = !; }
       | ...
       | VP[v]: (^XCOMP* VCOMP) = !;
       | VP[cop]: (^XCOMP* VCOMP) = !;
       | VP[coh]: (^XCOMP* VCOMP) = !;
       }

       Cbar
```

```
"extraposed constituents"
[ COMMA
{ ...
| VP[v]: (^XCOMP* (OBJ) VCOMP) = !
| VP[cop]: (^XCOMP* (OBJ) VCOMP) = !
| VP[coh]: (^XCOMP* (OBJ) VCOMP) = !
| ...
}
COMMA ]#0#2.
```

The example rule shows that the three VP types obscure the structure of the rule. Moreover, the generalization that the three VP types behave alike (e.g. they are annotated by the same f-structure constraints) is not encoded explicitly. It could be sheer coincidence.

Meta-categories allow for the encoding of the generalization. The VP meta-category is defined as follows.

```
VP = { VP[v]
     | VP[cop]
     | VP[coh] }.
```

The modified CP rule is shown below. Whenever the meta-category VP occurs in a rule, the above disjunction is copied to the place of the meta-category (the f-structure annotations are distributed across the disjuncts). That is, the resulting CP rule is identical to the above one.

```
CP --> "topicalized constituents"
     { DP: { (^SUBJ) = !;
           | (^XCOMP* OBJ) = !; }
     | ...
     | VP: (^XCOMP* VCOMP) = !;
     }

     Cbar

     "extraposed constituents"
     [ COMMA
     { ...
     | VP: (^XCOMP* (OBJ) VCOMP) = !
     }
     COMMA ]#0#2.
```

3.3.3.3 Parametrized Rules (Complex Categories)

Rules can be parametrized, similar to Macros. The parameters typically encode properties that otherwise are encoded in f-structure. For instance, the DP rule in our analysis is parametrized with respect to its type (standard, interrogative, relative).

Parameters are of the form [_para]. Whenever [_para] appears on the left hand side of a grammar rule, it is interpreted as a parameter. If a category on the right hand side also contains the parameter [_para], they are identified, i.e. the value of _para is "inherited" by the daughter category. As an example, cf. the following rule.

```
DP[_type] -->
     D[_type]
     NP.
```

Similar to macros, the value of the parameter in a complex category such as DP[_type] is determined when the category is introduced in a rule. For instance, standard DPs can be introduced by the German VP rule.

```
VP -->
     DP[std]: { (^SUBJ) = !
              | (^OBJ) = ! }*
     V
```

The VP rule "calls" the category DP[_type] with the value _type = std, that is a category DP[std] is inserted in the c-structure representation. Moreover, the value _type = std is passed on to the category D[_type], i.e. D[std] represents the daughter of DP[std], cf. the tree representation below.

```
              VP
            /    \
      DP[std]      V
         |
       D[std]
```
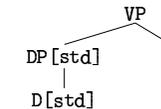
In other contexts, values other than _type = std are specified, e.g. in relative clauses.

```
CPrel -->
     DP[rel]: { (^SUBJ) = !
              | (^OBJ) = ! }
     Cbar.
```

Similar to f-structure annotations, the parameter can also be used to restrict the internal syntax of a category. Compare the two versions of the DP rule, the first making use of a feature TYPE encoding the DP type, the second employing the parameter _type. Both rules introduce an optional D by the disjunction { D | e }. In both rules, the determinerless option is restricted to the standard type.

```
DP --> { D: ^=!
        | e: (^TYPE) = std }
        NP.


DP[_type] -->
        { D[_type]
        | e: _type = std }
        NP.
```

In the definition of a parametrized rule, the possible values of the parameter(s) can be enumerated.

```
DP[_type $ {std int rel}] -->
        { D[_type]
        | e: _type = std }
        NP.
```

The main reason for using parametrized rules is efficiency. Whereas the computation of the c-structure representation is polynominal, the computation of the f-structure can be exponential (in the worst case). That is, shifting suitable f-structure features to the c-structure improves the performance of the grammar.

In this section, various means of abbreviation applying to c-structure rules have been presented. They can be used to increase the transparency of rules and tree representations, to encode linguistic generalizations, and to improve efficiency. In the next section, abbreviations applying to f-structure are described.

3.3.4 Abbreviation Means in F-Structure

XLE provides for abbreviation means in f-structure that are similar to macros. They are called templates (sec. 3.3.4.1). Alternations of subcategorization frames are encoded by lexical rules (sec. 3.3.4.2).

3.3.4.1 Templates

Similar to macros, templates can be used to bundle f-structure annotations. For instance, extraposed relative clauses are annotated by multiple f-structure constraints, as shown below (cf. p. 60).

```
... CPrel: (^ {SUBJ|OBJ} ADJUNCT $ OBJ) = %ANTECEDENT
           (%ANTECEDENT ADJ-REL) = !
           (%ANTECEDENT NUM) = (!NUM)
           (%ANTECEDENT GEND) = (!GEND)
```

The annotations can be encoded by a template @REL-CONDITIONS. The annotation of the category CPrel is modified accordingly.

```
... CPrel: @(REL-CONDITIONS);


REL-CONDITIONS =
           (^ {SUBJ|OBJ} ADJUNCT $ OBJ) = %ANTECEDENT
           (%ANTECEDENT ADJ-REL) = !
           (%ANTECEDENT NUM) = (!NUM)
           (%ANTECEDENT GEND) = (!GEND).
```

Templates may also contain parameters. For instance, the annotation of PP arguments depends on the position of the PP. The template PPfunc_desig encodes the variable annotation path, via the parameter _desig ("designator"). The value of _desig is determined by the rule calling the template, e.g. _desig = (↑XCOMP) by the VP rule, and _desig = (↑ { COMP | XCOMP }*) by the CP rule.

```
VP --> PP: @(PPfunc_desig (^XCOMP*));
       V.


CP --> PP: @(PPfunc_desig (^ { COMP | XCOMP }*) );
       Cbar.
```

The parameter _desig in the template PPfunc_desig is instantiated by the above values. Hence, oblique PPs dominated by a VP are annotated by (↑XCOMP* OBL) = ↓, whereas topicalized oblique PPs are annotated by (↑ { COMP | XCOMP }* OBL) = ↓, see the definition of PPfunc_desig below.

```
PPfunc_desig(_desig) = {     "non-semantic OBLtheta"
                             (_desig OBL) = ↓
                             (↓ PTYPE) = nosem

                     |       "semantic OBLtheta"
                       {  (_desig OBL-AG) = ↓
                       |  (_desig OBL-LOC) = ↓
                       |  (_desig OBL-DIR) = ↓
                       |  (_desig OBL-MANNER) = ↓
                       }
                             (↓ PTYPE) = sem
                     }.
```

"F-structure macros"   Functional uncertainty paths, such as { COMP | XCOMP }*, are often defined by means of "f-structure macros", cf. the macro @FU-TOPIC.

```
FU-TOPIC = { COMP | XCOMP }*.
```

In contrast to ordinary c-structure macros, these f-structure macros are invoked without '@' (similar to meta-categories); cf. the modified CP rule.

```
CP --> DP: @(DPfunc_desig (^FU-TOPIC) );
       Cbar.
```

(Templates encode entire f-structure annotations. In contrast, "f-structure macros" encode f-structure designators (cf. p. 55).)

The template @IF   The template called @IF is defined as follows.

```
IF(P Q) = "If P then Q"
          { ~P
          | "~~P: constraining equation, not defining"
            ~~P   Q}.
```

That is, @IF encodes logical implication: if P then Q is true iff: either P is not valid (~P) or Q is valid (Q). (The additional constraint ~~P within the second disjunct makes the two disjuncts exclusive, i.e. they cannot be valid simultaneously. Otherwise, if P were not true but Q, both of the above disjuncts would be fulfilled, which would result in spurious ambiguities.)

For instance, count nouns such as *Hund* 'dog' may call the template @COUNT-NOUN in their (stem) lexicon entry.

```
Hund   N-S xle (^PRED) = '%stem'
             @(COUNT-NOUN).

COUNT-NOUN =
       @(IF (^NUM) = sg
       (^SPEC)
       ).
```

@COUNT-NOUN ensures that singular count nouns have a specifier: If the f-structure of the count noun contains the feature-value pair [NUM sg] (= first parameter), then it must also contain the feature SPEC (= second parameter).

The template @COMPLETE   @COMPLETE is a predefined template. It can be used for a more efficient computation of "local" constraining equations (=c) or existential constraints.

Normally, such constraints are global, i.e. the decision that an existential constraint like (↑SPEC) is not fulfilled cannot be made until the entire f-structure is constructed. However, it is clear that the specifier feature of a DP in German cannot be supplied by, e.g., a topicalized determiner (since determiners cannot be extracted in German).

The template @COMPLETE allows for a specification of all features that have to be satisfied locally. This can improve the performance of the grammar. As an example, cf. the DP rules below.

```
DP -->                                DPx -->
   DPx: @(COMPLETE (^SPEC)).            (D)
                                        NP.
```

The expression @(COMPLETE (^SPEC)) is to be read as follows. If there is a constraining equation (or an existential constraint) imposed on the f-structure of the DP (= denoted by ^ within (^SPEC)) which concerns the feature SPEC, then the DP's f-structure can be marked as incomplete if the f-structure of DPx does not contain that feature (when processed bottom-up). In other words, the feature SPEC must be supplied by some constituent within the DPx phrase.

(The reason of the additional intermediate projection DPx is the encoding of the special template @COMPLETE.)

3.3.4.2 Lexical Rules

Lexical rules are used to encode alternations of subcategorization frames. For instance, passive, which maps the logical object to the surface subject, can be described by the following mapping rule (or rewrite rule). (The logical subject is mapped to NULL, i.e. it is deleted.)

```
(^OBJ) --> (^SUBJ)
(^SUBJ) --> NULL
```

We make use of a passive template @PASSIVE containing the rewrite rules. However, note that the mapping rules apply to both transitive and ditransitive verbs. Therefore, the template @PASSIVE must contain a parameter for the original subcategorization frame (transitive or ditransitive), which is to be rewritten.

Below we show the passive template and two example lexicon entries, for *schließen* 'close' and *geben* 'give'. The parameter _frame is instantiated by the expression (^PRED)='%stem<(^SUBJ)(^OBJ)>' in the case of the transitive *schließen*, and by (^PRED)='%stem<(^SUBJ)(^OBJ)(^OBJ2)>' in the case of the ditransitive *geben*.

```
PASSIVE(_frame) =
    { _frame  "original, active version"
    | _frame  "passive version"
      (^OBJ) --> (^SUBJ)
      (^SUBJ) --> NULL
    }.
```

```
schließen V-S xle @(PASSIVE (^PRED)='%stem<(^SUBJ)(^OBJ)>').
geben     V-S xle @(PASSIVE (^PRED)='%stem<(^SUBJ)(^OBJ)(^OBJ2)>').
```

The expanded lexicon entry of *schließen*, with the template call replaced by the content of the template, looks as follows.

```
schließen V-S xle
    { (^PRED)='%stem<(^SUBJ)(^OBJ)>' "original, active version"
    | (^PRED)='%stem<NULL(^OBJ)>' "passive version"
    }.
```

Finally, cf. the f-structure analysis of the passive sentence (38), as displayed by XLE

(38)   *Die Bank wird   geschlossen.*
        the bank becomes closed
        'The bank is being closed.'

```
        "Die Bank wird geschlossen."

        ⎡PRED   'schließen<NULL, [1:Bank]>'        ⎤
        ⎢                                           ⎥
        ⎢       ⎡PRED 'Bank'                ⎤       ⎥
        ⎢SUBJ   ⎢                          ⎥       ⎥
        ⎢      1⎣SPEC ⎡DET ⎡PRED 'die'⎤⎤⎦  ⎥       ⎥
        ⎢                                           ⎥
        84⎣TOPIC  [1:Bank]                          ⎦
```

In this section, we presented abbreviation means that apply to the f-structure representation. Similar to macros, they can increase transparency of the grammar code, encode linguistic generalizations (e.g. lexical rules), and improve efficiency (e.g. the template @COMPLETE).

3.3.5   Optimality Projection

XLE allows the user to define additional projections, e.g. a semantic projection. In our implementation, we use an o-projection (optimality projection), mainly to reduce ambiguities and to improve the robustness of the grammar (Frank *et al.* 2001).

Ambiguity reduction   As an example, consider the functions of PPs. A PP can function as an oblique argument or an adjunct. The reading as an oblique arguments is usually the preferred one, cf. (39). In the oblique reading, Hans is waiting for the money. In the modifier reading, Hans is sitting on money while waiting.

(39)   *Hans wartet auf Geld.*
        H.   waits on money
        'Hans is waiting for/on money.'

The o-projection can be used to mark one of the readings, e.g. the modifier reading. This is done by projecting a so-called OT mark, e.g. PPAsAdjunct. This OT mark can be declared to be dispreferred. Then, if a sentence receives several analyses, the analysis with a dispreference mark is suppressed (it is "suboptimal").

O-structure annotations are attached to c-structure rules in a way similar to f-structure annotations, cf. the VP rule. (Remember that 'o::*' denotes the o-structure of the current node (cf. p. 56).) Hence, the OT mark PPAsAdjunct is said to be an element of the o-projection of the PP. O-projections are multi-sets.

```
VP --> PP: { (^OBL) = !
           | ! $ (^ADJUNCT)
             PPAsAdjunct $ o::*
           }
        V.
```

Another possibility is to mark the oblique reading by an OT mark PPAsObl, declaring the mark as preferred.

Robustness   In addition to their use in ambiguity reduction, we use OT marks to increase the robustness of our grammar. For instance, in contrast to mass nouns, singular count nouns usually need a specifier in German. This is implemented in our grammar, by suitable constraints on the feature SPEC, based on a noun lexicon that lists mass nouns.

However, the mass noun lexicon is far from being complete. Hence, all nouns that are not listed by that lexicon are, by default, assumed to be count nouns. However, in a dispreferred analysis they may also be mass nouns, which covers instances without a specifier.

Besides preferred and dispreferred marks, there are two other types of marks: NOGOOD marks and STOPPOINT marks.

NOGOOD   OT marks that are declared as NOGOOD deactivate the constructions they appear in. For instance, constructions typical of technical text, such as verb-final imperatives, can be marked by `TechnicalConstr`.

```
CP --> { ...
        | "verb-final imperatives"
          VP: TechnicalConstr $ o::* }.
```

When parsing text other than technical instructions, the OT mark `TechnicalConstr` can be declared as NOGOOD, and the disjunct is deactivated.

This method can also be used to improve the performance of the grammar, by deactivating "expensive" constructions.

STOPPOINT   STOPPOINT OT marks can be seen as a compromise between ordinary OT marks and NOGOODs. They also deactivate the construction they appear in. However, if the sentence does not get an analysis at all, the construction is reactivated again and the sentence is parsed once more.

STOPPOINT OT marks can be used to mark constructions that are rare and expensive to compute. Hence, many sentences are parsed more efficiently, since the expensive rule is deactivated.

However, in all the cases where a sentence does not get an analysis, the parsing time increases to a high degree. First, because the sentence has to be reparsed, and second, because all kinds of expensive constructions are now allowed.

As we have seen, OT marks can be used to reduce ambiguities and to increase robustness, by marking certain analyses as suboptimal. In addition, NOGOOD OT marks (and to a certain degree, STOPPOINT OT marks) can improve efficiency. Finally, OT marks may be used for grammar specialization.

3.4   Summary

In this chapter, we presented an introduction to XLE, the grammar development platform we use in the implementation of the German LFG grammar.

XLE is a grammar development platform, suitable for the implementation of large-scale LFG grammars. XLE allows for the integration of external modules, such as tokenizer, morphological analyzer, and large lexicons. XLE also provides various means of abstraction, e.g. macros. These means allow for a modularization of the grammar and, hence, affect the transparency and maintainability of the grammar code.

As will be seen in the next chapter, our implementation focuses on a maximum of transparency and maintainability of the grammar code. These are prerequisites of the successful development and flexible usage of a large-scale grammar.

Part II

How to Document a Grammar

Chapter 4

Aspects of Grammar Documentation

Contents

This chapter addresses aspects of documenting a large-scale grammar. The line of reasoning goes as follows. We argue that a grammar implementation (as opposed to canonical software programs) exhibits special properties. We show that these properties have an impact on the form of the documentation. (In the next chapter, a documentation technique is proposed that incorporates the required features.)

Grammar development can be seen as the development of a special kind of software, namely the grammar. That is, the implementation of a large-scale grammar constitutes a large software project. As such, it should adhere to the techniques and design principles that are known from software engineering, such as modularity, maintainability.

We claim, however, that grammar implementation differs from canonical software engineering in important aspects.

(i) Relevance of grammar code: due to the linguistic underpinning of the underlying syntactic theory, people other than grammar developers show interest in the actual code. This fact distinguishes a grammar implementation from other pieces of software, where details of the implementation are only relevant to software developers.

(ii) Grammar modules are not black boxes: the concept of modularity in a grammar implementation differs in parts from the concept of modularity in software engineering. Similarly to modules in ordinary software programs, grammar modules encode generalizations. However, these modules encode linguistic insights and, hence, are not black boxes (whose internal structure is irrelevant).

In the following sections, we first provide evidence to support these claims ((sec. 4.1) and (sec. 4.2)). We then show that these properties impose special restrictions on the content and structure of documentation (sec. 4.4).

4.1   Relevance of Grammar Code

Usually, people are only interested in the functionality, that is the input-output behaviour, of a piece of software. Details of the implementation are relevant to software developers only. In contrast, a grammar implementation represents important information by itself in that it encodes formalizations of linguistic phenomena (in a particular linguistic framework). Hence, the grammar code is relevant not only to grammar developers but also to linguists who are interested in the implementation of a linguistic theory.

A grammar implementation may be of interest to linguists for different reasons: (i) to verify a linguistic theory; (ii) for new data and their formalizations.

Verification of a linguistic theory   A grammar implementation may be used to verify theoretical linguistic analyses, by providing answers to questions such as: (i) Does the proposed analysis of a specific phenomenon correctly map the input to the desired output? (ii) Does the analysis give rise to unwarranted overgeneration? (iii) How do analyses of different phenomena interact? That is, does each single analysis still produce the correct output, and without overgenerating?

Through such verifications, shortcomings of the theory in question can be detected, resulting in new insights and developments in linguistic theory. For instance, the proposal of m-structure, a morphosyntactic projection (Butt *et al.* 1996, Frank and Zaenen 2002), goes back to findings that have emerged from work in the Pargram project.

New data and formalizations   Large-scale grammars are supposed to cover substantial real-life texts, e.g. newspaper text. Many construction types that occur in such texts have not thus far been the topic of theoretical research, i.e. no (formal) analyses for these constructions have been proposed yet.

Therefore, the analyses implemented by the grammar writer represent the first formalizations of these constructions, within an overall consistent, large grammar fragment. (Note, however, that the grammar writer is often guided by pragmatic considerations, such as minimizing the effects on efficiency or ambiguity.)

The fact that the grammar code itself represents important information has an effect on the concept of modularity: in contrast to modules in canonical software, the internal structure of (many) grammar modules is relevant to the "outside", i.e. these grammar modules are not black boxes.

4.2   Grammar Modules are not Black Boxes

In this section, we discuss some differences between typical/ideal modules in canonical software and (certain) modules in a grammar implementation.

We start by presenting two prominent properties of modules in ordinary software engineering: (i) modules as functional units, and (ii) modules as black boxes. We will then show that many modules of a grammar implementation are only defined by property (i).

4.2.1   Modules in Ordinary Software

In software engineering, modularity is a central design principle (McConnell 1993, ch. 6). Modularity implies that the software code consists of different modules, which are "black boxes" to each other. That is, the input and output

of each module (i.e. the interfaces between the modules) are clearly defined, while the module-internal routines that map the input to the output are invisible to other modules.

A module consists of pieces of code that belong together in some way (e.g. they perform similar actions on the input). That is, the code is structured according to functional considerations ("module cohesion").

Modular code design supports transparency, consistency, and maintainability of the code. (i) Transparency: irrelevant details of the implementation can be hidden in a module, i.e. the code is not obscured by too many details. (ii) Consistency is furthered by applying once-defined modules to many problem instances. (iii) Maintainability: if a certain functionality of the software is to be modified, the software developer ideally only has to modify the code within the module encoding that functionality. In this way, all modifications are local in the sense that they do not require subsequent adjustments to other modules.

We now turn to the question of which sorts of modules can be distinguished in a grammar.

4.2.2 Modules in Grammar Implementations

In the overall architecture of a grammar implementation, certain components can be clearly modularized. These are all the "external modules" (as we call them), i.e. the tokenizer, morphology, guesser, and the lexicon (cf. sec. 3.2). These modules are perfect instances of the concept of modules, as proposed in software engineering: (i) Each module handles a specific aspect of the analysis of an input sentence. The tokenizer splits the sentence into tokens; the morphological analyzer maps tokens to morphological tags, etc. That is, each module is a functional unit. (ii) The output of one module represents the input to the next module, i.e. the modules do not interact but are black boxes. Only if a module is modified in such a way that its output is affected are subsequent modifications of other modules necessary.

The grammar in itself also represents such a module. Ordered morphological tags serve as its input, full syntactic analyses represent its output.

In contrast to these large modules, it is not as obvious which parts of the grammar code itself constitute (lower-level) modules.

We claim that all encodings of linguistic generalizations represent some kind of modules, namely in the sense of representing functional units (property (i)). However, not all of them are black boxes (property (ii)).

In the following sections, we discuss different features of the grammar that encode linguistic generalizations: (i) projection levels (sec. 4.2.2.1); (ii) syntactic rules and X′-projections (sec. 4.2.2.2); (iii) macros, templates (sec. 4.2.2.3); (iv) f-structure (sec. 4.2.2.4).

4.2.2.1 Projection Levels

The theory of LFG represents a modular approach to the analysis of natural language, in that various projection levels are distinguished (e.g. c-structure and f-structure). Each projection level covers a specific aspect of the formal analysis of a sentence (e.g. c-structure covers the analysis of constituency) and exhibits specific properties.

That is, each projection type forms a functional unit and thus realizes some kind of module. However, projections such as c-structure and f-structure are tightly interwoven by the ϕ-projection and interact (we come back to this topic below (cf. sec. 4.2.2.4)). This means that these projection-modules do not conform to the black-box principle known from software engineering.

4.2.2.2 Syntactic Rules and X′-Projections

Syntactic rules represent smaller units of the grammar code. At this level of detail, each single rule can be viewed as a (low-level) module. A syntactic category occurring on the right-hand side of a rule then corresponds to a module call. For instance, the DP rule-module is "called" in different places in the grammar code, for example in the VP rule (in other words, the VP rule introduces a DP).

Among other things, the call of a rule-module causes the creation of a node in the c-structure (e.g. a DP node) and introduces partial f-structure specifications. In addition, a rule-module calls other rule-modules, e.g., the DP rule may call the rule-modules D and NP. Terminal rules call modules consisting of lexicon entries.

In this view, the code of a large-scale grammar consists of a large number of low-level modules, which create a highly complex "module structure", which is mirrored by the c-structure tree.

Higher-level modules consist of cohesive groups of low-level modules. In a grammar, lexical or functional categories (X) and their projections (X′, XP), as defined by X′-theory, can be seen as such a (parametrized) higher-level module (not to be confused with the above LFG projection levels). The X′-module defines possible expansions of the general category XP, depending on certain parameters. In this view, a call of the VP rule-module corresponds to a call of the X′-module with parameters set, e.g., to [+predicative,+transitive,+lexical] (for the features [+predicative] and [+transitive], cf. Bresnan 2001, ch. 6.2).

The assumption of such a linguistically motivated X′-module is favoured by the following two aspects:

(i) X′-projections are functional units: lexical/functional categories and their projections obviously form a (linguistically motivated) logical unit, sharing important properties.

(ii) X′-projections seem to be black boxes: typically, only maximal projections (XP) are called by other rules. The internal expansion of the XP is irrelevant to the calling rule. For instance, the VP rule, which introduces a DP, does not impose restrictions on the internal structure of the DP. (However, this is not entirely true, as we will see below (cf. sec. 4.2.2.4).)

To sum up, the projections of a syntactic category (XP, X′, X) can sensibly be viewed as parts of a larger module since they form a linguistically motivated unit with functional properties (in the sense of modularization). They model essentially a linguistically determined input-output behaviour. Put differently, the projections, and grammar modules in general, encode linguistic generalizations.

While X′-generalizations are widely assumed in theoretical work, they are usually not encoded explicitly/overtly. Abbreviations such as "VP" are preferred to feature specifications such as "[+predicative,+transitive,+lexical]", for reasons of clarity.

Generalizations that remain implicit are error-prone. If the analysis of a certain phenomenon is modified, all constructions that adhere to the same principles should be affected as well, automatically—which is not the case with implicit generalizations.

A way of encoding X′-generalizations explicitly within XLE is by means of parametrized macros or parametrized rules (Kuhn 1999). In our implementation, however, we stick to the simpler notation "VP", "DP", etc. We do, though, make use of macros (and templates) for the encoding of other generalizations.

4.2.2.3 Macros, Templates

We start with a (simplified) example from our implementation that illustrates the use of macros and templates.

@FU-TOPIC and @PPfunc_desig   In German, the functional annotation of a DP or PP argument depends on the position of that argument. For instance, in the topicalized position (i.e. in the specifier position of CP, the so-called "Vorfeld"), a PP argument is annotated by the functional uncertainty equation (↑ { COMP | XCOMP }* GF) = ↓ (for the definition of GF, see below). In contrast, a PP within the VP (in the so-called "Mittelfeld") is annotated by (↑XCOMP* GF) = ↓.

That is, the annotation of the VP-internal PP does not feature the function COMP. This reflects the fact that VP-internal PPs cannot be extracted from finite clauses, in contrast to topicalized PPs. If the feature variable GF is set to OBL, the two types of annotations account for the data in (40) (topicalized PP *auf wen* 'for whom'), (41) (VP-internal PP *auf Otto* 'for Otto'), and (42) (ungrammatical VP-internal PP *auf Otto* 'for Otto').

(40)   *Auf wen   glaubt   Maria, daß Hans wartet?*
       [on whom SPEC-CP] believes M.   that H.   waits
       'Who does Maria believe Hans is waiting for?'

(41)   *daß Maria auf Otto hat warten wollen*
       that [M. on O.   has wait   wanted VP]
       'that Maria wanted to wait for Otto'

(42)   **daß Maria auf Otto glaubt,   daß Hans wartet*
       that [M.   on O.   believes VP] that H.   waits

Besides this difference, however, there are many properties shared by topicalized and VP-internal PPs. For instance, both topicalized and VP-internal obliques (OBLs) are marked by the feature [PTYPE nosem].

Further, irrespective of the PP's position, the feature variable GF can be instantiated by different functions: OBL (idiosyncratically marked PP obliques, as in the above examples), OBL-AG (oblique agent), OBL-LOC (locative argument), OBL-DIR (directional argument), OBL-MANNER (manner argument).

As a consequence, the complete respective annotations of a topicalized PP argument and a VP-internal PP argument might look as follows:

```
CP → ...
    PP: {     "non-semantic OBL"
              (↑ { COMP | XCOMP }* OBL) = ↓
              (↓ PTYPE) = nosem

        |     "semantic OBLtheta"
        {  (↑ { COMP | XCOMP }* OBL-AG) = ↓
        |  (↑ { COMP | XCOMP }* OBL-LOC) = ↓
        |  (↑ { COMP | XCOMP }* OBL-DIR) = ↓
        |  (↑ { COMP | XCOMP }* OBL-MANNER) = ↓
        }
              (↓ PTYPE) = sem
        }
    ...
```

```
VP →  ...
      PP:  {      "non-semantic OBL"
                  (↑ XCOMP* OBL) = ↓
                  (↓ PTYPE) = nosem

              |      "semantic OBLtheta"
            {  (↑ XCOMP* OBL-AG) = ↓
            |  (↑ XCOMP* OBL-LOC) = ↓
            |  (↑ XCOMP* OBL-DIR) = ↓
            |  (↑ XCOMP* OBL-MANNER) = ↓
            }
                  (↓ PTYPE) = sem
          }
      ...
```

Such an implementation obviously misses several generalizations. For instance, it is not encoded explicitly, i.e. as a generalization, that the annotations of all topicalized PP arguments feature the same functional uncertainty, namely { COMP | XCOMP }*. It could be sheer coincidence in the CP rule as formulated above.

A way of making the generalization explicit is by using a macro to encode the functional uncertainty path, cf. the macro @FU-TOPIC and the modified CP rule below. (Remember that f-structure macros are invoked without '@' (cf. p. 70).)

```
FU-TOPIC = { COMP | XCOMP }*.
```

```
CP →  ...
      PP:  {      "non-semantic OBLtheta"
                  (↑ FU-TOPIC OBL) = ↓
                  (↓ PTYPE) = nosem

              |      "semantic OBLtheta"
            {  (↑ FU-TOPIC OBL-AG) = ↓
            |  (↑ FU-TOPIC OBL-LOC) = ↓
            |  (↑ FU-TOPIC OBL-DIR) = ↓
            |  (↑ FU-TOPIC OBL-MANNER) = ↓
            }
                  (↓ PTYPE) = sem
          }
      ...
```

This encoding technique has the further advantage that the annotation paths are (often) more perspicuous and easier to maintain. For instance, the grammar writer might decide to restrict functional uncertainty in topicalized position for efficiency reasons. Then she/he simply has to modify the definition of the macro @FU-TOPIC.

Another generalization that is still missing from the above reformulation is the fact that the grammatical functions OBL, OBL-AG, etc. are shared by topicalized and VP-internal PPs. This fact can be captured by a template which generates all possible functions. A parameter, _desig, allows for a variable functional uncertainty path, which is instantiated in different ways by topicalized PPs and VP-internal PP arguments, cf. the template @PPfunc_desig below.

```
PPfunc_desig(_desig) = {      "non-semantic OBLtheta"
                              (_desig OBL) = ↓
                              (↓ PTYPE) = nosem

                          |      "semantic OBLtheta"
                        {  (_desig OBL-AG) = ↓
                        |  (_desig OBL-LOC) = ↓
                        |  (_desig OBL-DIR) = ↓
                        |  (_desig OBL-MANNER) = ↓
                        }
                              (↓ PTYPE) = sem
                      }.
```

The definitions of the CP and VP rules look much simpler now (and modifications of the analysis of PP functions now only require an update of a single template, @PPfunc_desig).

```
CP →  ...
      PP: @(PPfunc_desig (↑ FU-TOPIC) )

      ...

VP →  ...
      PP: @(PPfunc_desig (↑ FU-MITTELF) )

      ...

FU-MITTELF = XCOMP*.
```

To sum up, macros and templates allow for the encoding of generalizations and thus make the grammar code easier to maintain and more perspicuous. In another sense, however, the grammar code is now obscured: to understand the

final version of the CP rule, the definitions of the template @PPfunc_desig and the macro @FU-TOPIC has to be known. The definitions may even be stacked, and thus need to be traced back to understand the rule encodings. (Below we will argue that such dependencies call for special documentation techniques.)

In other words, templates and macros are modules, since they encode generalizations and, hence, represent logical units. They are not, however, black boxes, since their internal structure (= their definition) is relevant to the outside, i.e. to the rules that call the macros/templates.

We further use macros to improve code transparency. For instance, the DP rule in our implementation introduces an optional adverb, the head DPx, and finally different types of postnominal arguments and modifiers. The postnominal elements are covered by a very large disjunction. To keep the definition of the rule DP simple, the disjunction is hidden in a macro, @DPpost. Note that this macro is not used anywhere else, i.e. it does not encode a property shared by different categories. Instead, its sole purpose is to elucidate the structure of an otherwise complex rule.

```
DP[_type $ {std int rel}] -->
      ( e: _type = std;
       ADVdp: @(ADJUNCT) )

      DPx[_type]: @(DP-COMPLETE);


      ( "optional constituents may follow DP"
       @(DPpost) ) .
```

4.2.2.4   F-Structure

Finally, we return to X′-projections and our (simplified) statement that the internal expansion of an XP were irrelevant to the calling rule (cf. p. 82). On this view, the VP rule, which introduces a DP, would not impose restrictions on the internal structure of the DP.

It is indeed true, with respect to c-structure level, that the dominating VP does not impose restrictions on the DP's internal structure. However, the VP may influence the DP's structure indirectly via f-structure constraints. For instance, the DP within the VP rule could be annotated by the existential constraint (↓SPEC), thus requiring that the DP contain a specifier (i.e. expands to D and NP).

That means, as far as the level of c-structure is concerned, the DP module is a real black box. In contrast, the f-structure of the DP is accessible from outside, via unification. (Note that XLE allows for certain restricted c-structure

constraints. For instance, the VP-internal DP could be annotated by an existential constraint such as (* RIGHT_SISTER), thus requiring that the DP have a right sister (cf. sec. 3.3.2). However, this type of c-structure constraint is not part of core LFG.)

Moreover, due to powerful referencing means within global f-structures, as provided by LFG (outside-in/inside-out equations, functional uncertainty equations), f-structure restrictions are not (and can in general not be) limited to local subtrees. That is, the VP rule may impose restrictions on constituents that are arbitrarily distant from each other.

This is made possible by the fact that in LFG the f-structure analysis of a sentence does not consist of disconnected, single f-structures but of one (complex) f-structure. This is accomplished by annotating each non-terminal rule with f-structure annotations that relate ↑ and ↓ (i.e. the mother node's f-structure and the f-structure of the node itself are related), and by annotations within lexicon entries that refer to ↑.

In a way, f-structure information thus represents what is called "global data" in software engineering (McConnell 1993, ch. 10.6). All rules and lexicon entries are essentially operating on the same "global" data structures. In a certain sense, unification combined with LFG's powerful referencing means are the opposite of the concept of modularity/encapsulated information. Due to unification, there are no "local", encapsulated f-structures. Instead, all pieces of f-structure can be accessed from outside-in, and from inside-out.

Hence, whenever the grammar writer modifies the f-structure annotation of a rule (e.g. by renaming a feature), she/he has to be aware of the effects the modifications potentially have on other modules. It is for this reason that the use of global data is deprecated in software engineering, as it weakens the program's modularity and makes the program code difficult to manage (McConnell 1993, ch. 10.6). And it is for exactly the same reason that grammar engineering represents a particular challenge both for code transparency and modularity (and for its documentation, as we shall discuss below).

One strategy to minimize this problem is the consistent use of general templates (and macros) to encode and factor feature annotations. Ideally, all rules, macros, and lexicon entries that refer to a certain feature make use of one and the same (parametrized) template (i.e. the template serves as an "access routine"). In this way, a modification of this template suffices if this feature is to be modified, and all respective rules/macros/lexicon entries are affected automatically.

The grammar writer can be supported in her/his work in further ways, e.g. by a feature declaration (i.e. a complete list of admissible features and values), or by an index of all occurrences of a specific feature or rule in the code (such supporting tools are provided by XLE). In addition, a detailed grammar documentation represents an important factor in code maintainability.

In summary, assuming two prominent properties of modules—being (i) functional units and (ii) black boxes—we have shown that grammar modules roughly fall into two classes.

Modules of class one are perfect modules in that they exhibit both properties. External modules and, to a large extent, syntactic rules and X′-projections belong to this class (while the global nature of f-structure weakens the modularity of rules and X′-projections).

Modules of class two fulfil condition (i), by encoding linguistic generalizations. However, they fail to fulfil condition (ii) since their internal structure is relevant to the understanding of the outer context. Templates and macros are such class two modules.

In software engineering, a modular design is said to improve code transparency. As mentioned above, however, grammar modules of class two may obscure the grammar code. This tension between modularity and transparency is the topic of the next section.

4.3   Code Transparency

The encoding of linguistic generalizations was presented above as an important motivation for the use of macros/templates. Such an explicit encoding of generalizations also furthers maintainability and transparency of the grammar code. From another point of view, however, transparency may also suffer from the use of macros/templates.

In order to distinguish these opposing views more precisely we introduce two notions of transparency, which we call "intensional" and "extensional".

"Intensional transparency"   Intensional transparency of grammar code means that the characteristic, defining properties of a construction are encoded by means of suitable macros/templates, i.e in terms of generalizing definitions. Hence, all constructions that share certain defining properties make use of the same macros/templates to encode these properties.

Conversely, distinguishing properties of different constructions are encoded by different macros/templates—even if the content of the macros/templates is identical.

To give an example, a grammar may distinguish different types of empty elements, such as the head of elliptical DPs, as in (43), or the unexpressed subject of certain infinitival constructions, as in (44).

(43)   *Hans bevorzugt warmen.*
H.   prefers   [DP warm]
'(Maria likes cold tea,) Hans prefers warm one.'

(44)   *ohne   zu warten*
without [VP to wait]
'without waiting'

In our analysis, the corresponding f-structures are marked by the features [PRED 'pro'] and [PRON-TYPE null], cf. the f-structures embedded under the functions OBJ and OBJ SUBJ, respectively, in the (simplified) f-structure analyses below.

$$
\begin{bmatrix}
\text{PRED} & \text{'bevorzugen} < \text{SUBJ,OBJ} > ' \\
\text{SUBJ} & \begin{bmatrix} \text{PRED} & \text{'Hans'} \end{bmatrix} \\
\text{OBJ} & \begin{bmatrix} \textit{PRED} & \textit{'pro'} \\ \textit{PRON-TYPE} & \textit{null} \\ \text{ADJUNCT} & \left\{ \begin{bmatrix} \text{PRED} & \text{'warm'} \end{bmatrix} \right\} \end{bmatrix}
\end{bmatrix}
$$

$$
\begin{bmatrix}
\text{PRED} & \text{'ohne} < \text{OBJ} > ' \\
\text{OBJ} & \begin{bmatrix} \text{PRED} & \text{'warten} < \text{SUBJ} > ' \\ \text{SUBJ} & \begin{bmatrix} \textit{PRED} & \textit{'pro'} \\ \textit{PRON-TYPE} & \textit{null} \end{bmatrix} \end{bmatrix}
\end{bmatrix}
$$

Depending on the linguistic analysis, the grammar writer may decide that both empty elements are instances of one and the same phenomenon. Accordingly, the rules of DP and CPconj (= adverbial CP) will both make use of a (parametrized) template @EMPTY, which introduces the features marking empty elements.

```
DP  →  (DET)
       { NP
       | e: @(EMPTY ↑) }.


CPconj  →  CONJ
           VP:    (↑OBJ) = ↓
                  @(EMPTY (↓SUBJ)).


EMPTY(_desig) =  (_desig PRED) = 'pro'
                 (_desig PRON-TYPE) = null.
```

Conversely, the grammar writer may analyze the empty elements in these constructions as instances of two different phenomena, whose f-structures happen to be identical for arbitrary reasons. In this case, different templates ought to be used, as in the following rule versions.

```
DP  →  (DET)
       { NP
       | e: @(EMPTY-HEAD) }.


CPconj  →  CONJ
           VP:    (↑OBJ) = ↓
                  @(EMPTY-SUBJ).


EMPTY-HEAD =  (↑ PRED) = 'pro'
              (↑ PRON-TYPE) = null.


EMPTY-SUBJ =  (↑ SUBJ PRED) = 'pro'
              (↑ SUBJ PRON-TYPE) = null.
```

If the grammar writer decides to modify the analysis of, e.g., elliptical DPs, then a modification of the template @EMPTY-HEAD, as desired, does not affect the analysis of the infinitival constructions.

Common and distinguishing properties of different constructions can thus be easily determined. That is, it is easy to distinguish between intended/motivated and arbitrary similarities of different constructions.

"Extensional transparency" Extensional transparency means that linguistic generalizations are stated "extensionally", i.e. templates and macros are replaced by their content/definition (similar to a compiled version of the code). The grammar rules thus introduce the constraints directly rather than by calling a macro/template that would introduce them.

As an example, compare the two versions of the example from the previous section, featuring a topicalized PP, which is introduced by the CP rule. The first rule version focuses on extensional transparency, while the second adheres to the principle of intensional transparency.

(i) Extensionally transparent CP

```
CP  →  ...
          PP:  {     "non-semantic OBL"
                     (↑ { COMP | XCOMP }* OBL) = ↓
                     (↓ PTYPE) = nosem

               |     "semantic OBLtheta"
                 {  (↑ { COMP | XCOMP }* OBL-AG) = ↓
                 |  (↑ { COMP | XCOMP }* OBL-LOC) = ↓
                 |  (↑ { COMP | XCOMP }* OBL-DIR) = ↓
                 |  (↑ { COMP | XCOMP }* OBL-MANNER) = ↓
                 }
                     (↓ PTYPE) = sem
               }
          ...
```

(ii) Intensionally transparent CP

```
CP  →  ...
          PP: @(PPfunc_desig (↑ FU-TOPIC) )

          ...


PPfunc_desig(_desig) =  {     "non-semantic OBLtheta"
                              (_desig OBL) = ↓
                              (↓ PTYPE) = nosem

                        |     "semantic OBLtheta"
                          {  (_desig OBL-AG) = ↓
                          |  (_desig OBL-LOC) = ↓
                          |  (_desig OBL-DIR) = ↓
                          |  (_desig OBL-MANNER) = ↓
                          }
                              (↓ PTYPE) = sem
                        }.

FU-TOPIC = { COMP | XCOMP }*.
```

Comparing both versions, the first, extensional version may seem easier to grasp and, hence, more transparent. To understand the second, generalized version, it is necessary to follow the template and macro calls and look up the respective definitions. Obviously, one needs to read more lines of code in the

second version, and often these lines of code are spread over different places and files. For instance, the CP rule may be part of a file covering the CP internal rules, while the template @PPfunc_desig figures in some other file.

Especially for people who are not well acquainted with the grammar, the intensional version thus requires more effort. In contrast, people who work regularly on the grammar code know the definitions/functionalities of macros and templates more or less by heart. They certainly grasp the grammar and its generalizations more easily in the intensional version.

The criteria of maintainability and consistency clearly favour intensional over extensional transparency. We argue that the shortcomings of intensional transparency, namely poorer readability for casual users of the grammar, can be compensated by a special documentation structure. The next section discusses possible features of such a type of documentation.

4.4   Properties of Grammar Documentation

This section presents features and restrictions of documentation of grammar implementations. We argue that the grammar-specific properties we presented above—(i) the relevance of the grammar code, and (ii) the fact that modules in general are not black boxes—have an impact on possible modes of grammar documentation. Property (i) determines the content of the documentation (cf. sec. 4.4.1), while property (ii) influences the structure of the documentation (cf. sec. 4.4.2).

4.4.1   Documentation Content

This section starts by presenting selected properties of documentation of ordinary software, which are then applied to documentation of grammar implementations.

Documentation of ordinary software   In large software projects, code documentation consists of high-level and low-level documentation (McConnell 1993, ch. 19). High-level documentation comprises information about the function and requirements of (high-level) modules and keeps track of higher-level design decisions (e.g. what modules are distinguished). More detailed documentation includes lower-level design decisions, such as the reasons for the chosen algorithms or data structures.

The lowest level of documentation is code-level documentation. However, it reports about the code's intent rather than implementation details, i.e. it focuses on "why" rather than "how". For instance, it summarizes functions and routines. A large part of code-level documentation is taken over by "good programming

style", e.g. "use of straight-forward and easily understandable approaches, good variable names, good routine names" (McConnell 1993, p. 454).

High-level documentation is external to the software code (in large software projects, most of the documentation is code external). Code-level documentation is part of the code; such code-internal documentation is called comments.

Grammar documentation   Applied to grammar documentation, a high-level documentation comprises documentation of the function of large modules, such as the tokenizer, the lexicon, the c-structure and f-structure, etc. (similar to our introductions to LFG and XLE in chapters 2 and 3).

Further, it must include a specification of the data that are covered by the implementation. In addition, relevant c-structure and f-structure properties of the corresponding output have to be documented.

For instance, the documentation should comprise a description of the different usages of the German PP (adjuncts, semantic oblique complements, idiosyncratically marked PP obliques), the linguistic tests that distinguish them, and, moreover, how the differences are mirrored by the f-structure representations.

At a more detailed level, it is especially important that the documentation indicates whether a specific analysis is chosen for technical rather than linguistic reasons (e.g. based on performance considerations).

Whereas on the higher levels, the documentation contents of ordinary software and computational grammars seem quite similar, low-level/code-level documentation clearly differs. The differences are related to the fact that in a grammar implementation the code by itself represents important information, as argued above (cf. sec. 4.1). That is, the details of the input-output mapping represent the actual linguistic analysis. Hence, in contrast to documentation of other types of software, grammar documentation has to focus both on "why" and "how".

To give an example, in German the relative pronoun within a (potentially extraposed) relative clause must agree in gender and number with the constituent it modifies, its "antecedent". The agreement constraints are covered by the template @FIND-HEAD-REL in our implementation. We display the first lines of this template.

```
FIND-HEAD-REL =
    "relative clause in Nachfeld has to find its antecedent
    to check for NUM/GEND agreement"
    ...
```

The comment within this template is an example of documenting "why" (what is the purpose of the template?) rather than "how" (how is agreement actually achieved?).

In the next section, we will argue that grammar documentation, which necessarily focuses on "how", cannot be reasonably represented by comments but must be code-external.

4.4.2 Documentation Structure

Above (sec. 4.2) we distinguished two classes of grammar modules, (i) (nearly) "perfect" modules, which are functional units and black boxes (e.g. X′-projections), and (ii) modules that are functional units but not black boxes (e.g. templates and macros).

Suppose a grammar only made use of class one modules, and the code-level documentation were realized by code-internal comments, similarly to code-level documentation in ordinary software. In this case, the most important difference between grammar and software documentation would be the degree of detail: since grammar documentation has to report both about "why" and "how", the documentation within the grammar code would be much more detailed. For instance, a single line of code would often be accompanied by several lines of comments.

In contrast, a grammar that encodes certain generalizations by use of macros and templates (i.e., class two modules) is more difficult to document. We argue that the documentation of such a grammar demands a specialized documentation method, which cannot be realized code-internally, by means of comments.

As desribed above (cf. sec. 4.2), the content of a macro/template is relevant to the outside, i.e. to the rule that calls the macro or template. Hence, to properly understand the functionality of a certain rule, the functionality of all (cascaded) macros/templates that are called by this rule has to be understood as well. The documentation of this rule thus depends on the documentation of the respective macros/templates.

Link-based documentation   One way of encoding such dependencies is by means of links. Within the documentation of the rule, a pointer would point to the documentation of the macros/templates that are called by this rule. The reader of the documentation would simply follow these links (which might be realized by hyperlinks).

Certain programming languages provide tools for the automatic generation of documentation, based on comments within the program code (e.g. Java comprises the documentation tool Javadoc, URL: `http://java.sun.com/javadoc/`). The generated documentation makes use of hyperlinks as described above, which point to the documentation of all routines and functions that are used by the documented module.

However, a typical grammar rule calls many macros and templates, and macros often call other macros and templates. This hierarchical structure makes

the reading of a link-based documentation troublesome. We illustrate this by an example from our implementation.

The sublexical rule of determiners makes use of the macro @QUANT-EXPRESSION, among others.

```
D[_type] -->
        ...
    |
        @(QUANT-EXPRESSION det _type)
    | ...
```

The macro @QUANT-EXPRESSION, in turn, consists of a large disjunction of different types of "quantifying expressions", each covered by a macro on its own (cf. the DP documentation in Part III for the motivation of this choice of encoding). For instance, articles are covered by the macro @QUANTart.

```
QUANT-EXPRESSION(_cat _type) =
        ...
    | "articles (inserted under DET)"
      e: _type = std;
      @(QUANTart _cat)
    | ...
```

The macro @QUANTart expands to the sublexical categories that mark the lemma (ART-S) and part of speech (ART-T). Inflectional tags are again handled by another macro, @DISTR-INFL.

```
QUANTart(_cat) =
      ART-S_BASE: _cat = det;
      ART-T_BASE
      @(DISTR-INFL _cat).

DISTR-INFL(_cat) =
      "distributional and inflectional features"

    { "_cat = det/predet"
      DISTR-ATTR_BASE
      e: { _cat = det
           (^CHECK _SPEC-TYPE _DET) = attr
         | "special disjunct for predeterminers:
           don't introduce CHECK feature here
           - predeterminers marked by .Attr automatically have
           right sister (namely D)
           - CHECK feature of predet would unify with right
```

```
            sister D, which would then need a right sister
            itself; this would exclude:"
            "EX: Manch einer kam. (Some people came.)
            (since 'einer' would need right sister due to
            requirements of 'manch')"
            _cat = predet };

    | "_cat = det/predet/adj"
      DISTR-PRO_BASE
    }


    { "gender, case, number"
      @(GCN)
      "inflection type"
      { INFL-F_BASE[det]: { _cat = det
                          | _cat = predet };
      | INFL-F_BASE[adj]: _cat = adj; }

    | "invariant"
      INVAR-F_BASE: { _cat = det
                    | _cat = predet };
    }.
```

Finally, @DISTR-INFL calls the macro @GCN, which covers gender, case, and number tags.

```
GCN =
    GEND-F_BASE
    CASE-F_BASE
    NUM-F_BASE.
```

As can be seen from this example, a documentation user who is interested in the definition of the sublexical rule of D would have to follow down four levels of macro calls: @QUANT-EXPRESSION > @QUANTart > @DISTR-INFL > @GCN. We can safely conclude from this example that a (purely) link-based documentation is not a suitable way of documenting such hierarchically structured code.

(Note that routines and functions in ordinary software may be hierarchically organized as well. In contrast to grammar modules, however, these modules are (usually) black boxes. That is, a reader of the documentation is not forced to follow all the links to understand the functionality of the top-most module.)

Copy-based documentation   As an alternative, we have designed a documentation method which results in a user-friendly presentation of the documentation parts.

Our documentation is highly detailed, therefore the amount of documenting text clearly exceeds the amount of code. Hence, it seems natural to switch the roles of text and code in our approach: in ordinary software documentation, code-level documentation consists of comments that are embedded in the code. In our documentation, in contrast, it is the code which is embedded in the code-level documentation. (This method is described extensively in the next chapter.)

In our approach, the documentation of a rule comprises copies of the relevant macros rather than simple links to these macros. In a way, our documentation tool mirrors a compiler, which replaces each macro call by the content/definition of the respective macro. In contrast to a (simple) compiler, however, our documentation keeps a record of the macro calls (i.e. the original macro calls are still existent). In the terminology introduced above (cf. sec. 4.3), our documentation thus combines extensional transparency (by copying the content of the macros) with intensional transparency (by keeping a record of the macro calls).

The copy-based method has the advantage that the structure of the documentation is totally independent of the structure of the code which is being documented.

Code-external documentation   The fact that our documentation is code-external has further advantages. For instance, code-external documentation is not limited in space, in contrast to code-internal documentation (for reasons of clarity, the number of comment lines should not exceed the number of code lines).

In addition, our documentation permits the inclusion of illustrating pictures. It also allows for a nice formatting of the text, thus easing, e.g., the presentation of example sentences including literal translations.

In essence, code-external documentation is more flexible than code-internal comments and, hence, may profit from all kinds of technical means. Among other things, it makes the documentation independent from the hierarchical structure of the code. (Note, however, that we make also use of comments within the code, which complement the external documentation.)

4.5   Summary

This chapter dealt with methodological and technical aspects of documenting a large-scale grammar. Large grammars are similar to other types of large software in that modularity plays an important role in the maintainability and,

hence, reusability of the code.

However, grammars differ from other software in two (related) aspects. (i) The grammar code by itself represents important information in that it encodes linguistic analyses. That means, grammar users are not only interested in the input and output of the grammar but also in the implementation details. (ii) Similar to software modules, grammar modules represent functional units in that they encode linguistic generalizations. Yet, in contrast with software modules, certain of the grammar modules (macros, templates) are not black boxes because their internal structure/content is relevant to the outside. In addition, grammars do encode and manipulate, in large parts, "global data" (represented by the f-structure analysis of a sentence).

These two grammar-specific properties have an impact on the content and structure of the documentation. Property (i) implies that large parts of grammar documentation consist of code-level documentation. Property (ii) makes the documentation of a rule dependent on the documentation of macros/templates that are called by this rule. These dependencies, in combination with the hierarchical structure of grammar code, run counter the requirements of (extensional) code transparency, and call for a special documentation technique.

We suggest a code-external documentation method that permits copying of relevant grammar parts (such as macros), thus making the structure of the documentation independent of the structure of the code. Moreover, an external documentation is not restricted in form and space. Therefore, this method allows for a user-friendly presentation of the grammar documentation.

In the next chapter, we propose an XML-based method of writing grammar documentation that incorporates the features discussed above. Part III of this dissertation represents a documentation example generated by our proposed method.

Chapter 5

XML-based Grammar Documentation

Contents

This chapter proposes a method of writing documentation that is sufficiently powerful and flexible to fulfil the restrictions on grammar documentation that have been presented in the previous chapter. The documentation is code-external and consists of an XML document (XML: eXtensible Markup Language). The XML source document is transformed to the final output documentation by an XSLT processor (XSLT: eXtensible Style Language for Transformations).

In the first section, we present a short introduction to XML (cf. sec. 5.1) and XSLT (cf. sec. 5.2). Subsequently, we illustrate how XML-based documentation responds to the various demands on the documentation that have been described above (cf. sec. 5.3).

XML-based documentation may further support the process of grammar development, e.g. by automatically providing a test suite and an index of all rules, macros and templates in the documentation (cf. sec. 5.4). Aspects of the maintainability of the documentation are also addressed (cf. sec. 5.5).

Part III of this work (chapters 7–11) presents, as an example, the documentation of the DP of the German Pargram grammar. This documentation is fully generated by the proposed method.

5.1   Introduction to XML

XML is a standardized document exchange format (URL: `http://www.w3.org/XML/`, cf., e.g., Harold and Means 2002). It is a markup language, i.e. allows a document to be enriched by markup. The markup is used to encode semantic properties of specific parts of the document. Which properties are encoded is decided by the user.

XML markup consists of pairs of opening and closing tags ('<tag>' and '</tag>', respectively) which encode a semantic property of the part enclosed by the tags. A pair of tags is also called an "element", and the part enclosed by the tags is called the "content" of that element.

For instance, an LFG grammar rule can be annotated by markup. The tags '<c_cat>' and '<f_annots>' can be used to mark c-structure categories and f-structure annotations in syntactic rules. As a first example, compare the two version of the VP rule below, one version in XLE notation, the other in XML notation.

```
VP --> DP: (^OBJ)=!
           (!CASE)=acc;
       V: ^=!.
```

```
<rule>
   <c_cat>VP</c_cat> -->
```

```
      <c_cat>DP</c_cat>
          <f_annots>(^OBJ)=! (!CASE)=acc</f_annots>
      <c_cat>V</c_cat>
          <f_annots>^=!</f_annots>
</rule>
```

At first sight, the XML code seems to obscure the actual rule. Note, however, that the XML version is mostly self-explanatory. In contrast, the meaning of, e.g., the colon and the semicolon in the XLE version has to be stated explicitly: the colon separates a category from its f-structure annotation; the semicolon marks the end of the f-structure annotation. Both punctuation marks are superfluous in the XML version.

Typical XML documents are hierarchically organized; most elements (i.e. tags) embed further elements. Furthermore, (opening) tags can be specified by attributes: '<element attr1="value1" attr2="value2">' (equivalently, single quotes may enclose the values). Finally, elements may be empty: '<empty/>' (not shown here).

To illustrate the power and flexibility of XML, we show two further alternative ways of encoding the above example in XML. The first alternative makes use of linguistic terminology. To ease the reading of the XML examples, we use a different colouring for the element content and feature values on the one hand, and XML tags on the other. (Note that typical XML documents are generated automatically; hence, it is not problematic that the amount of markup exceeds the amount of actual content.)

```
<rule>
  <mother>VP</mother>
  <daughters>
    <daughter>
        <c_cat>DP</c_cat>
        <f_annots>
            <f_annot>(^OBJ)=!</f_annot>
            <f_annot>(!CASE)=acc</f_annot>
        </f_annots>
    </daughter>
    <daughter>
        <c_cat>V</c_cat>
        <f_annots>
            <f_annot>^=!</f_annot>
        </f_annots>
    </daughter>
  </daughters>
</rule>
```

The second alternative follows the software-oriented terminology that we introduced in the previous chapter (cf. sec. 4.2.2.2). (In this alternative, f-structure annotations can be viewed as arguments that are passed through function/module calls.)

```
<module type="rule" id="VP">
  <called-modules>
     <module ref="DP">
        <arguments>
           <argument>(^OBJ)=!</argument>
           <argument>(!CASE)=acc</argument>
        </arguments>
     </module>
     <module ref="V">
        <arguments>
           <argument>^=!</argument>
        </arguments>
     </module>
  </called-modules>
</module>
```

The two alternatives illustrate that the same information can be encoded by an element or an attribute, compare the element '<rule>' in the first alternative and the attribute 'type="rule" ' in the second. (Note that this element/attribute makes the right arrow superfluous. Similarly, the final full stop can be omitted.)

XML-typical attributes are 'id' and 'ref'. They (uniquely) identify an element and allow other elements to refer to it.

XML allows for a declaration of the structure and content of elements that occur in a document. The declaration is called DTD (Document Type Definition). A DTD may define certain attributes as unique attributes, i.e. the XML processor will print out a warning if there are different elements with identical values of such attributes. For instance, the DTD of the last XML example comprises the following specifications.

```
<!ELEMENT module (called-modules)>
<!ATTLIST module type CDATA #REQUIRED
              id   ID    #REQUIRED
>
```

The DTD fragment defines the fact that the element 'module' embeds another element, 'called-modules'. In addition, the tag 'module' must specify the attributes 'type' and 'id' (this is determined by the keyword '#REQUIRED'). The attribute 'id' is of type 'ID', i.e. it is a unique attribute/identifier.

The above XML examples show that the markup can be used to encode all kinds of relevant information, depending on the needs of the user. XML tags serve two purposes: they structure a document and they encode semantic properties.

Obviously, an XML document as such is not suited for the human user. Therefore, further tools are used to process an XML document. These tools operate on the XML tags and the content. They allow the user to define the layout of the document and the formatting of specific elements (e.g. the content of certain elements is to be printed in bold-face).

In our documentation, we use XSLT to generate the final output document on the basis of the XML source documentation. This is the topic of the next section.

5.2 Transformations via XSLT

XSLT is a transformation language that allows the user to define transformation rules (URL: `http:/www.w3.org/TR/xslt`, cf., e.g., Tidwell 2001). An XSLT processor (e.g. Xalan, URL: `http://xml.apache.org/xalan-j/`) takes an XML document as its input and produces an output according to the specifications of the rules. The following figure outlines the generation of the output document.
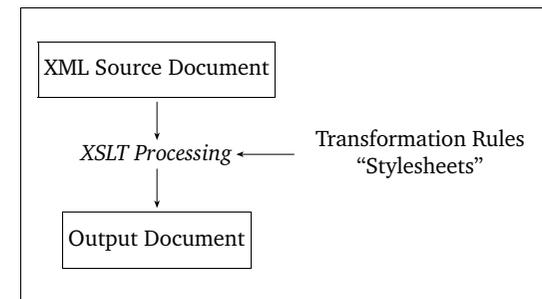
Figure 5.1: XSLT transformation of an XML document

In addition to the formatting issues mentioned above, XSLT can be used to hide/delete elements and to order them in alternative ways. The exact modifications are to be defined by the user by means of so-called stylesheets (the transformation rules). The rules (recursively) take XML elements as their input and produce an output, according to the specifications of the rules.

1st example: generating XLE output For instance, the user may want to define rules that operate on the tags <rule>, <mother>, etc., as used in the fragment below.

```
<rule>
  <mother>VP</mother>
  <daughters>
...
```

Suppose that the user wants to generate a rule similar to XLE notation. In this case, the user might define that whenever the XSLT processor encounters a tag <rule>, it adds a linebreak to the output. If it encounters a <mother> tag, it prints out the content of the tag (i.e. 'VP' in the example). If the processor encounters a tag <daughters>, it prints out a space, followed by a right arrow.

That is, the above fragment of the XML input gives rise to the partial output shown below.

```
VP -->
```

An example stylesheet/rule is given for the <daughters> tag. (Note that an XSLT stylesheet itself represents an XML document.)

```
 <xsl:template match="daughters">
    <xsl:text>--></xsl:text>
    <xsl:apply-templates/>
 </xsl:template>
```

The stylesheet can be read as follows: If the XSLT processor encounters a tag that matches the string "daughters", it outputs some text, namely the string ' -->' (i.e. a space, followed by the right arrow). (The next line, '<xsl:apply-templates/>', triggers the recursive processing of all remaining tags.)

The remaining rules can be defined as follows. For the tag <daughter>, no output is defined. The rule for <c_cat> defines the printing of a linebreak and the element content (i.e. first 'DP' and later 'V'). The rule for <f_annots> contains the instruction to first insert a colon, then to print the content of all embedded tags (<f_annot>), each preceded by a space, and finally a semicolon. The closing tag </rule> triggers the insertion of a full stop.

The final output of XSLT then looks as follows (the layout can certainly still be improved).

```
VP -->
DP: (^OBJ)=! (!CASE)=acc;
V: ^=!;.
```

2nd example: generating LaTeX output It is also possible to define variants of rules that apply depending on the context. For instance, in certain contexts the user may prefer to generate an annotated tree representation of the XML code, as shown below.
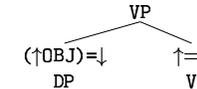
For this task, an output format has to be chosen that can represent trees, e.g. LaTeX. The definition of the stylesheet rules then comprise LaTeX commands as the output of certain XML tags.

For instance, the stylesheet rule for the <rule> tag specifies that the LaTeX command '\begin{tabular}' is added to the output (the 'tabular' environment is often used to encode trees in LaTex). Next, all "inner" tags (<mother>, <daughters>, etc.) are processed, due to the statement '<xsl:apply-templates/>'. The processing of the entire <rule> tag is finished by the output of the LaTeX command '\end{tabular}'.

```
<xsl:template match="rule">
    <xsl:text>\begin{tabular}</xsl:text>
    <xsl:apply-templates/>
    <xsl:text>\end{tabular}</xsl:text>
</xsl:template>
```

3rd example: generating HTML output As a final example, we sketch a possible HTML representation of the above VP rule. We repeat the first part of the XML representation of the VP rule.

```
<rule>
  <mother>VP</mother>
  <daughters>
    <daughter>
        <c_cat>DP</c_cat>
        <f_annots>
...
```

An important reason for generating HTML output are hyperlinks. For instance, one might define hyperlinks from each category/macro/template that is called on the right hand side of a rule (e.g. the DP category in the above VP

rule) to the corresponding definition (i.e. the DP rule). In this way, the reader of the documentation can easily look up the definitions of called modules.

In this scenario, each rule/macro/template definition has to be marked by an anchor, and each called module is accompanied by a pointer to the appropriate anchor. The HTML output, e.g. called 'grammar.html', might represent the above VP rule fragment as follows. ('>' in HTML denotes the '>' sign, '<br/>' a line break, ' ' a space. Anchors are represented by the tag '<a name="...">'; pointers are encoded by the tag '<a href="...">'.)

```
<a name="vp_rule"/>
VP --&gt;
<br/>
   <a href="grammar.html#dp_rule">DP</a>:
```

Displayed by a browser, the above HTML fragment looks as follows (the fact that the symbol DP is underlined indicates that it represents a pointer/hyperlink).

```
VP -->
    DP :
```

In the following section, we describe the way we make use of XML and XSLT to generate grammar documentation that responds best to the various demands described in the previous chapter.

5.3 Documenting via XML and XSLT

Our grammar documentation consists of a hand-written XML file. In addition, we use Perl scripts that add XML markup to the XLE grammar files. Both types of files, documentation XML files and grammar XML files, feed the XSLT processing, which generates the final output (in our case: LaTeX), as sketched by the following figure.

Figure 5.2: Documenting via XML and XSLT

In the following sections, we describe each step of the above figure: (i) the creation of the two source files (cf. sec. 5.3.1); (ii) the Perl processing, adding XML markup to the source grammar (cf. sec. 5.3.2); (iii) the interaction of XML and XSLT in the creation of the output documentation, followed by the LaTeX processing, which generates the final output file (cf. sec. 5.3.3). For ease of reference, we repeat the above figure in each section, highlighting the relevant parts.

5.3.1 Creation of the Source Files

The grammar and documentation writer only modifies and updates the two files at the top of the figure: the source grammar and the source documentation. All further files are either generated automatically (XML grammar, LaTeX documentation, ps documentation), or else they are pre-defined, stable transformations programs (Perl scripts, stylesheets).
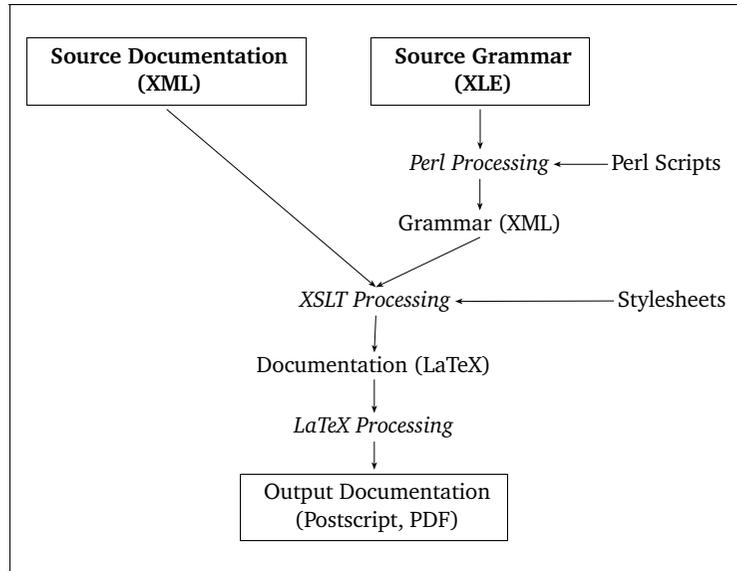
Figure 5.3: Documenting via XML and XSLT: creation of source files

(i) XLE source grammar: creation and maintenance of the XLE source grammar is supported by a specific LFG mode for Emacs (which is delivered as part of the XLE distribution). Among other things, this mode provides highlighting of comments vs. code, an index-based search, and automatic rule formatting.

(ii) XML source documentation: creation and maintenance of the XML source documentation is eased (i) by the fact that the amount of content (i.e. text) far exceeds the amount of markup within the documentation, and (ii) by specific editing facilities, as provided by many text editors.

Amount of content vs. markup Our source documentation mainly consists of pure text, interspersed with XML tags—in contrast to canonical XML documents that usually contain more markup than content. This is mainly due to the fact that the grammar code by itself represents condensed and important information (as argued in the previous chapter) and, hence, requires highly detailed documentation. One line of code often corresponds to several lines or paragraphs of documenting text.

Editing facilities The creation and maintenance of the XML source documentation is supported by many text editors. For instance, the XML mode for

Emacs comprises highlighting of XML tags, attributes, and values. In addition, it allows the user to navigate easily within an XML document by, e.g., moving the cursor to the next or previous element. Elements can be "folded", i.e. their content can be hidden in the display. Finally, the Emacs XML mode eases the insertion of XML tags: based on the DTD, it automatically inserts obligatory further elements and forces the user to specify required attributes.

To give an example, assume the following (partial) DTD.

```
<!ELEMENT module (called-modules)>
<!ATTLIST module type CDATA #REQUIRED
                 id   ID    #REQUIRED
>
```

Whenever the user inserts the opening tag '<module>' into the XML document, the embedded element '<called-modules>' and the closing tag </module> are added automatically. The user is further asked to specify the values of the obligatory attributes 'type' and 'id'. Moreover, the XML mode checks whether the specified value of 'id' is unique within the document.

5.3.2 Adding Markup to the Source Grammar

One type of source file is represented by the XLE grammar, which contains the definitions of rules, macros, and templates. These files are enriched by XML markup, added by a Perl script.
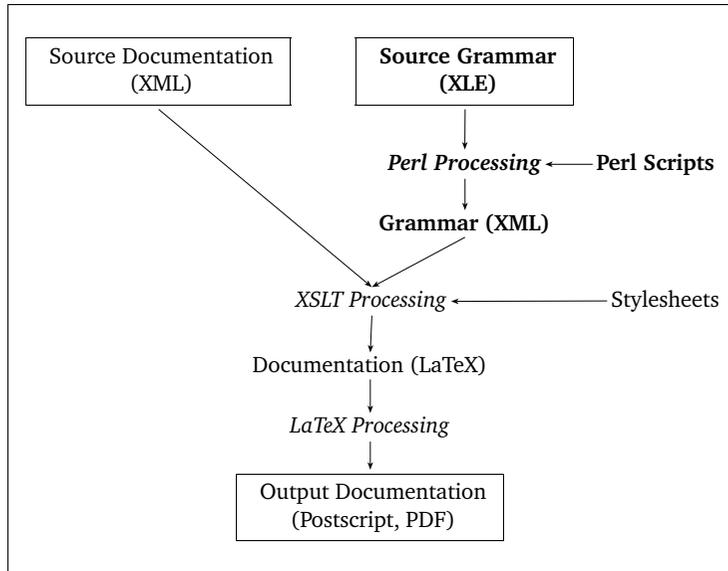
Figure 5.4: Documenting via XML and XSLT: adding markup

As an example, compare two versions of the template @PRENOM-GENITIVE. The original source version, encoded in XLE format, is shown first. (Comments in XLE are put in double-quotes. We use curley brackets to enclose example phrases in German ('{EX: ...}') and their English translations ('{E: ...}').)

```
PRENOM-GENITIVE =
    (^SPEC POSS) = !
    "{EX: DP[std]: Karls Hund}{E: Karl's dog}"
    @(CASE_desig ! gen)
    "must be morphologically marked:"
    "{EX: DP[std]: *Karl Hund}{E: Karl dog}".
```

The version with XML markup, generated automatically by the Perl processing, looks as follows. (For reasons of clarity, we slightly modify the layout of the XML version, by adding and removing white space.)

```
<unit id='dp.tmpl.PRENOM-GENITIVE' name='PRENOM-GENITIVE'
      delim='equal' type='template'>
PRENOM-GENITIVE =
```

```
(^SPEC POSS) = !

<comment>
<ex cat='DP[std]'><g>Karls Hund</g></ex><e>Karl's dog</e>
</comment>

@(CASE_desig ! gen)

<comment>must be morphologically marked:</comment>
<comment><ex judge='*' cat='DP[std]'><g>Karl Hund</g></ex>
<e>Karl dog</e>
</comment>. </unit>
```

In the following, we describe the meaning of each tag.

The <unit> tag The scope of each rule/macro/template is now marked by opening and closing <unit> tags.

```
<unit id='dp.tmpl.PRENOM-GENITIVE' name='PRENOM-GENITIVE'
      delim='equal' type='template'>
```

Each unit is marked by an 'id' attribute with a unique, complex value, e.g. 'dp.tmpl.PRENOM-GENITIVE'. The first two parts of this value are extracted from the name of the file the unit is part of. For instance, the above unit is part of the file 'dp.tmpl.lfg'. The third part consists of the name of the rule/macro/template itself ('PRENOM-GENITIVE').

These values permit easy location of the definition of a rule (or macro or template), via the first parts of the 'id' value. This property will be exploited by our copying mechanism, which copies the definition of a rule, guided by references to the 'id' value of this rule (cf. sec. 5.3.3). Moreover, the 'id' values are guaranteed to be unique, via the third part of the 'id' value, since XLE only allows for unique rule/macro/template names. (Note that XLE allows for multiple entries of a lexical item. Hence, the 'id' value of lexicon entries has to be determined manually.)

The attribute 'delim' encodes the delimiter between the left and right hand side of the rule/macro/template. Units encoding macros and templates are marked by 'delim='equal' ', rule units are marked by 'delim='arrow' '. This feature is used whenever only a part of a unit is to be displayed rather than the entire unit (cf. sec. 5.4.3).

Finally, the attribute 'type' will be used in the generation of the index, as represented by appendix C. The index entries are sorted according to their type, i.e. there is an index for rules/macros and one for templates (cf. sec. 5.4.5).

The <ex> tag   The <ex> tag contains a further tag, <g>, which marks German example phrases. English translations are marked by <e> tags.

```
<ex judge='*' cat='DP[std]'><g>Karl Hund</g></ex>
<e>Karl dog</e>
```

The <ex> tag is specified by the attribute 'cat', which indicates the phrasal category of the German example, e.g. 'DP[std]'. This attribute will be used in the creation of testsuites (cf. sec. 5.4.6). Ungrammatical or deviant, marginal data are further specified by the attribute 'judge', with values '*', '?', '??', etc. Such examples can be used to create negative test items, which should be rejected by the grammar.

The <comment> tag   Grammar comments (which are enclosed by double quotes in the original code) are now marked by the tag <comment>. This tag will be used to trigger a different colouring of the comments, thus mirroring the Emacs LFG mode.

```
<comment><ex cat='DP[std]'>...</comment>
```

The grammar code could certainly be marked up by further tags, similar to the example VP rule shown above, which exhibits tags marking c-structure and f-structure elements (cf. p. 101). However, our XML markup focuses on those tags that are prerequisites to our copying mechanism, which is presented in the next section.

5.3.3   Joining Documentation and Grammar Code

The two types of XML files, source documentation and grammar code with markup, represent the input to XSLT processing. Among other things, the XSLT processing performs the above mentioned copying task, finally resulting in a user-friendly display of the documentation (in our case: a postscript or PDF file, which is the output of subsequent LaTeX processing). The copying mechanism is presented in detail in this section.
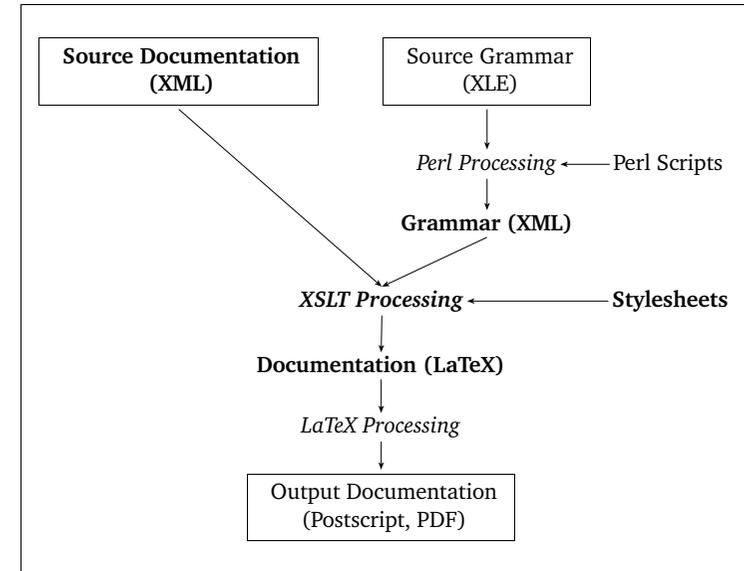
Figure 5.5: Documenting via XML and XSLT: joining text and code

The target documentation   As argued in the previous chapter, the documentation of a rule (or macro) is dependent on the macros/templates that are called by this rule (or macro). For instance, the template @PRENOM-GENITIVE, whose XML version has been discussed above (cf. p. 110), is called within the macro @SPEC-DP, cf. the simplified version of @SPEC-DP.

```
SPEC-DP(_type) =
    "Prenominal genitives"
    { "pronouns"
      PRON[_type]: @(PRENOM-GENITIVE)

    | "proper nouns"
      NAMEP: _type = std
             @(PRENOM-GENITIVE);
    }.
```

According to our reasoning, to make clear the functionality of @SPEC-DP, the documentation of @SPEC-DP has to refer to the definition of the template @PRENOM-GENITIVE (and to the definition of templates called by @PRENOM-GENITIVE, such as @CASE_desig).

Moreover, we argued that due to the hierarchical structure of the grammar, link-based documentation (which, e.g., links the template calls and the respective definitions) is not suitable. As an alternative, we designed a documentation method that comprises copies of the relevant grammar code rather than simple links to these code parts. In this way, all pieces of code that are relevant to the understanding of a certain rule (e.g. macro or template definitions, lexicon entries, feature declarations) can be gathered and presented jointly in the documentation section that addresses this rule. This is an important aspect of "user-friendly" and transparent documentation.

We illustrate this account by some simplified example documentation of the macro @SPEC-DP. For instance, we might want our target documentation to look as follows.

```
Prenominal genitives are covered by the macro @SPEC-DP. They
are realized by pronouns or proper nouns.

SPEC-DP(_type) =
    "Prenominal genitives"
    { "pronouns"
      PRON[_type]: @(PRENOM-GENITIVE)

    | "proper nouns"
      NAMEP: _type = std
             @(PRENOM-GENITIVE);
    }.


The prenominal constituents are represented by the feature
SPEC POSS, cf. the template @PRENOM-GENITIVE.

PRENOM-GENITIVE =
    (^SPEC POSS) = !
    "EX: Karls Hund (Karl's dog)"
    @(CASE_desig ! gen)
    ...

The constituents have to be marked as genitives, as required
by the call of the template @CASE_desig, with the second
parameter '_case' set to 'gen'.

CASE_desig(_desig _case) =
    (_desig CASE) = _case
    ...
```

Dependent macros and templates are often spread over different files. For instance, the templates @PRENOM-GENITIVE and @CASE_desig in our implementation are part of different files since the first is a DP-specific template (hence it is part of the file dp.tmpl.lfg) whereas the second is a general template, called by verbs, adjectives, nouns, etc. (hence it is part of the general file misc.tmpl.lfg).

Thus, the above example documentation abstracts from the grammar structure (hierarchical code structure and distributed file structure) and therefore makes it easier to read and understand the rules and documentation.

How to generate the target documentation As argued above (cf. sec. 4.4.2), we keep the (linear) source documentation and the (hierarchical) source grammar code apart. Hence, we need a mechanism to join the corresponding parts of the documentation and the code, as illustrated by the above example. (Certainly, the reader would not want to switch repeatedly between the documenting text and the grammar code.)

Therefore, the XML source documentation makes use of specific <code> tags which point to the relevant pieces of code. When the XML documents are processed by the XSLT processor, these tags trigger the insertion of the code bits that they point to. For instance, the definition (or parts of the definition) of a template might be copied to the output document by this mechanism.

Hence, the actual input source documentation of the above example would contain <code> tags rather than actual grammar code. Compare the actual input documentation with the above example. (The <code> tag is an instance of an empty element or tag, i.e. its content is empty. The notations '<code></code>' and '<code/>' are equivalent.)

```
Prenominal genitives are covered by the macro @SPEC-DP. They
are realized by pronouns or proper nouns.

<code ref="SPEC-DP.dp.gram"/>

The prenominal genitives are represented by the feature SPEC
POSS, cf. the template @PRENOM-GENITIVE.

<code ref="PRENOM-GENITIVE.dp.tmpl"/>

The constituents have to be marked as genitives, as required
by the call of the template @CASE_desig, with the second
parameter '_case' set to 'gen'.

<code ref="CASE_desig.misc.tmpl"/>
```

The <code> tags within the source documentation are marked by the attribute 'ref', which points to the 'id' attribute of the <unit> tags within the XML grammar code, cf. the <unit> tag which marks the template @PRENOM-GENITIVE, repeated below.

```
<unit id='dp.tmpl.PRENOM-GENITIVE' name='PRENOM-GENITIVE'
      delim='equal' type='template'>
PRENOM-GENITIVE =
    ...
```

The XSLT stylesheets specify that ordinary input text is simply copied to the output file. However, when the stylesheet processor encounters a <code> tag, it looks for a file named according to the first parts of the the 'ref' attribute of the <code> tag (e.g. 'dp.tmpl.lfg'). This file contains a <unit> tag with the referenced id ('dp.tmpl.PRENOM-GENITIVE'). The XSLT processing then copies the content of this <unit> tag to the output file.

Hence, the XSLT processing results in an output file similar to the above, user-friendly example documentation.

Generating LaTeX output   Besides inserting fragments of code, the XSLT processing performs further tasks. For instance, it specifies that copied code parts are printed "verbatim", i.e. in typewriter font and including the original indentation (triggered by the LaTeX commands '\begin{alltt}' and '\end{alltt}').

Further, comments within the grammar code are put in a different colour (by the LaTeX command '\textcolor{magenta}{...}'). Moreover, German examples are preceded by 'EX:', while the English translations are enclosed in parentheses. That is, the actual LaTeX representation of (the middle part of) the above example looks as follows.

```
The prenominal genitives are represented by the feature SPEC
POSS, cf. the template @PRENOM-GENITIVE.

\begin{alltt}PRENOM-GENITIVE =
    (^SPEC POSS) = !
    \textcolor{magenta}{"EX: Karls Hund (Karl's dog)"}
    @(CASE_desig ! gen)
    ...
```

The postscript (ps) or PDF file that results from this LaTeX fragment can be sent to a printer or viewed by a previewer, such as ghostview (this is the way the DP documentation in Part III is presented):

```
The prenominal genitives are represented by the feature SPEC POSS, cf.
the template @PRENOM-GENITIVE.

PRENOM-GENITIVE =
    (^SPEC POSS) = !
    "EX: Karls Hund (Karl's dog)"
    @(CASE_desig ! gen)
    ...
```

The following figure illustrates the copying of the template @PRENOM-GENITIVE and generation of LaTex output via XSLT.

which replaces <code> tags by copies of the referenced code fragments. Note that this approach guarantees that the code fragments that are displayed in the documentation are always up-to-date: whenever the output documentation is newly created by XSLT processing, the fragments are newly copied from the most recent version of the grammar.

The described documentation method is a powerful tool. Besides the copying task, it can be used to further enhance the readability of the documentation, e.g. by automatically providing c-structure and f-structure analyses of example sentences. Such enhancements are the topic of the next sections.

5.4 Further Features of the XML Documentation

The XML markup of the documentation can be exploited in various other ways, both to further the readibility of the documentation and to support the task of grammar writing (see also the suggestions by Erbach 1992). We address the following topics.

- Documentation for different readers (sec. 5.4.1)

- Different output formats (sec. 5.4.2)

- Refined code links (sec. 5.4.3)

- Snapshots of c-structure and f-structure analyses (sec. 5.4.3)

- Indices (sec. 5.4.5)

- Testsuites (sec. 5.4.6)

5.4.1 Documentation for Different Readers

Ordinary software products usually come with two types of documentation, one for the software developers, the other for the software users. The users only need to know about the input and output of the software. In contrast, the developers require documentation that explains the details of the implementation.

In the previous chapter, we argued that a grammar implementation differs from other types of software in that the code represents important information by itself and, in consequence, that the code is relevant not only to grammar developers but also to other people, such as linguists (cf. sec. 4.1).

The documentation of a grammar is a highly time-consuming and complex task. We therefore assume that one and the same grammar documentation
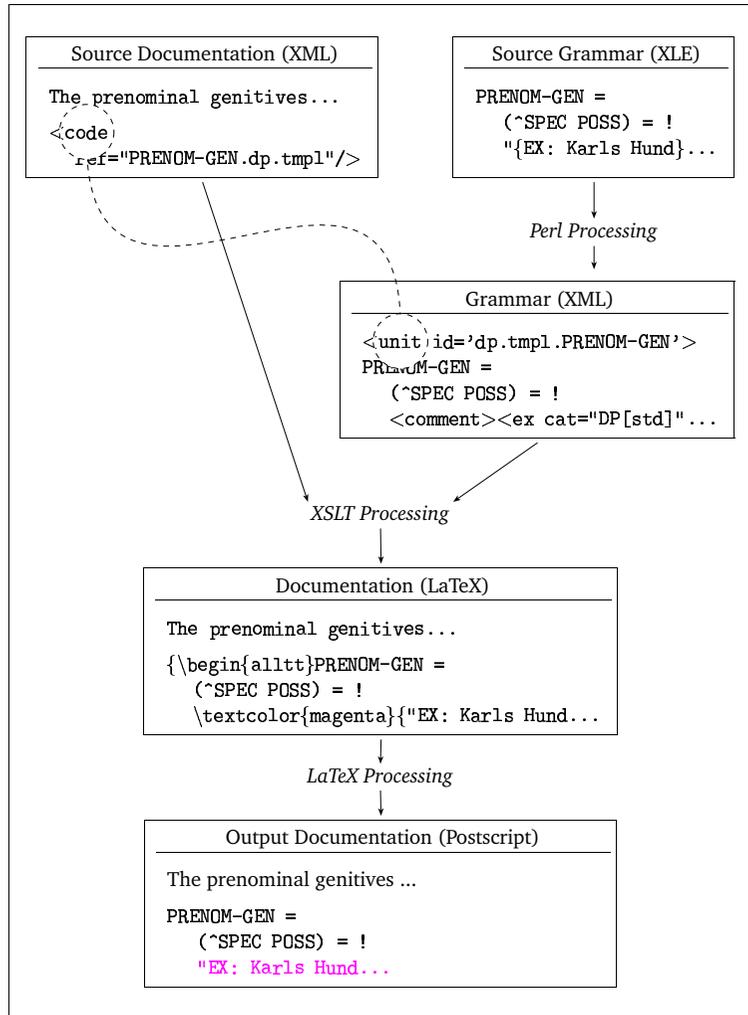

Figure 5.6: Documenting via XML and XSLT: an example

Summary   In this section, we presented the basic mechanisms of our XML/XSLT-based documentation. Two types of XML-marked-up input files, the documentation text and the grammar code, are joined by XSLT processing,

should serve different types of users. However, not every user of the grammar is interested in all implementation details. Often, it will be sufficient to know the phenomena covered by the implementation and basic properties of the c-structure and f-structure analyses of these phenomena.

In our approach, the source documentation of a syntactic projection, such as DP, is organized in such a way as to allow the generation of different variants of documentation that vary as to the degree of detail they deliver. Each single (source) documentation consists of several parts, addressing different aspects of the syntactic projection in question.

- An overview of the phenomena covered by the implementation and basic properties of the c-structure and f-structure analyses of these phenomena

- A description of the linguistic data and their analyses in the literature

- An overview of the implementation, focussing on the points where the implementation diverges from the theoretical analyses (without presenting the actual grammar code)

- Details of the f-structure analysis

- Details of the c-structure analysis

Each part is marked by a specific XML-attribute 'level', with values such as 'overview', 'basic-ling', 'basic-impl', 'details-fs', etc. The 'level'-attribute determines whether the respective part is to be included in the final output documentation.

For instance, users who are interested in a quick overview receive a documentation variant that only contain the overview of the phenomena and basic properties (marked by the attribute 'level="overview" '). Other users might be interested, e.g., in the differences between determiners and quantifying adjectives. These users need the sections about the linguistic data and their theoretical analyses ('level="basic-ling" '), and the respective implementation ('level="basic-impl" ').

The most complete documentation version contains all of the above parts. Chapters 7–11 of this dissertation present such a complete documentation version of our implementation of the German DP.

The documentation variants can easily be generated by means of different (or parametrized) XSLT stylesheets. For instance, for the simple overview documentation, the stylesheet copies only the overview part of the source file to the output documentation.

This documentation method is very flexibel, in that it allows for various kinds of documentation variants. At the same time, only one source documentation needs to be created and maintained.

5.4.2 Different Output Formats

Our approach allows for the generation of different output formats. For instance, a hypertext document (HTML, PDF with hyperlinks) can represent the final output. Hyperlinks are especially suited for grammar documentation, since they allow for easy switching between different grammar modules. (Remember, however, that the hierarchical structure of grammar code is, in certain ways, in conflict with (purely) link-based documentation (cf. p. 94).)

Similar to the case of the above output variants, there is only one XML source documentation necessary. This source documentation is processed by different (or parametrized) stylesheets, which determine the type of output, as illustrated by the following schema.
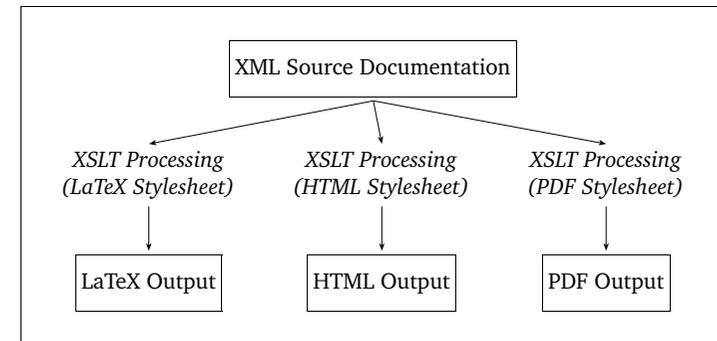
Figure 5.7: Documenting via XML and XSLT: varying output formats

5.4.3 Refined Code Links

Above we described how the definitions of grammar rules, macros or templates are copied to the output documentation (cf. sec. 5.3.3). Such rule/macro/template definitions can be very large and complex. Often, however, only small portions of the definitions are relevant to the current context of the documentation. Consider the following macro definition.

```
QUANT-EXPRESSION(_cat _type) =
      "used by quants (D and Aquant)"
      { "articles (inserted under DET)"
        e: _type = std;
        @(QUANTart _cat)

      | "demonstratives (inserted under DEM)"
        e: _type = std;
```

```
    @(QUANTdemon _cat)


  | "possessives (inserted under POSS)"
    e: _type = std;
    @(QUANTposs _cat)


  | "indefinites (inserted under QUANT)"
    e: _type = std;
    @(QUANTindef _cat)


  | "interrogatives (inserted under INT)"
    e: _type = int;
    @(QUANTint _cat)
  }.
```

The above macro covers all kinds of determiner-like expressions. In the context of the documentation of articles, only the first disjunct is actually important. Displaying the entire, complex macro would hinder the readability of the documentation since the reader might be distracted by irrelevant portions of the code.

Therefore, we (manually) added specific tags to the grammar code, marking portions of code that are relevant to the documentation. The tag <u> is used for unspecified "units" of code, <d> marks disjuncts. Both tag types are obligatorily specified by identifier attributes, called 'subid', see the following XLE macro fragment.

```
QUANT-EXPRESSION(_cat _type) =
      "used by quants (D and Aquant)"
      { <d subid='art'>"articles (inserted under DET)"
        e: _type = std;
        @(QUANTart _cat)
        </d>


      | "demonstratives (inserted under DEM)"
        ...
```

Similar to the 'id' attribute in the copy mechanism described above, the 'subid' attribute is used to mark the piece of code that is to be copied to the output. In these cases, the <code> tag in the source documentation is (manually) augmented by the attributes 'u' or 'd', which refer to the corresponding tags in the grammar code.

For instance, in the context of the documentation of articles, only the first disjunct would be referenced by the following <code> tag.

```
<code ref="QUANT-EXPRESSION.dp.gram" d="art"/>
```

This tag gives rise to the following, partial display of the macro @QUANT-EXPRESSION (for the syntax of such partial displays (cf. p. 141)).

```
QUANT-EXPRESSION(_cat _type) =
      ...
    | "articles (inserted under DET)"
      e: _type = std;
      @(QUANTart _cat)
    | ...
```

This approach allows the copied code to be restricted to specified portions, thus enabling the reader to focus on the lines of code that are actually relevant in the current context.

5.4.4   Snapshots of C-Structure and F-Structure Analyses

Grammar documentation is much easier to read if pictures of c-structures and f-structures illustrate the analyses. XLE supports the generation of snapshot postscript files, displaying trees and f-structures, which can be included in a LaTeX document. Note, however, that after any grammar modification such snapshots have to be updated, since the modified grammar may now yield different trees and f-structure analyses. Therefore, snapshots are automatically generated in our documentation.

Again, this is triggered by XML markup. Snapshots in our documentation are always associated with example phrases. These are marked by the tag <ex>, cf. the following example.

```
<ex cat="PP[std]">
<g>mit Maria</g>
<l>with M.</l>
<e>with Maria</e>
</ex>
```

The <ex> tag contains the German example, enclosed by <g>, the literal translation <l>, and the English translation <e> (for the attribute 'cat' (cf. sec. 5.4.6)). (For technical reasons, <ex> tags within the grammar code are different from <ex> tags within the source documentation. For instance, the <e> tag within the grammar code is not contained by the <ex> tag (cf. p. 112).)

Above all, these tags trigger the formatting of the examples in the documentation. The German example is printed in italics, the literal translation is aligned with the German words, and, finally, the English translation is quoted, cf. (45), which represents the final display of the above example.

(45)   *mit Maria*
       with M.
       'with Maria'

The example tag <ex> can be specified by further attributes. If it is specified by
the attribute 'cs="ord" ' (= ordinary), the XSLT processing produces a snapshot
of the c-structure of the corresponding example (by running XLE with the cur-
rent version of the grammar). Similarly, the attribute 'fs' triggers a snapshot of
the f-structure: 'fs="full" ' triggers a snapshot of the full f-structure, 'fs="pred" '
corresponds to an f-structure that displays the PRED features only.

For instance, the following example tag gives rise to the example in (46)
and the c-structure and f-structure snapshots displayed below.

```
<ex cat="PP[std]" cs="ord" fs="pred">
<g>mit Maria</g>
<l>with M.</l>
<e>with Maria</e>
</ex>
```

(46)   *mit Maria*
       with M.
       'with Maria'

```
CS 1:      PP[std]
              |
           PPx[std]
            /      \
       P[pre]    DP[std]
          |          |
         mit      DPx[std]
                     |
                    NP
                     |
                  NAMEP
                     |
                   Maria
```

```
      "mit Maria"

      ⎡PRED  'mit<[27:Maria]>'      ⎤
     0⎢OBJ 27[PRED 'Maria']         ⎥
      ⎣                             ⎦
```
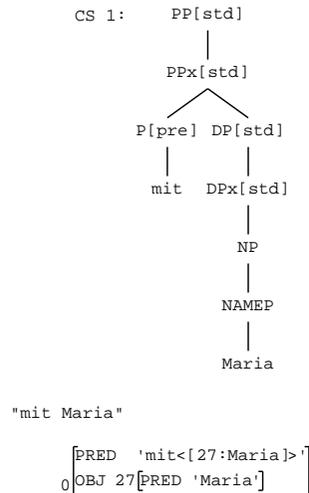
Alternatively, the attribute 'cs' may be specified as 'cs="morph" ' (= morpholo-
gy), as in the following example.

```
<ex cat="P[pre]" cs="morph">
<g>mit</g>
<l>with</l>
<e>with</e>
</ex>
```

In this case, sublexical trees are displayed, cf. the example in (47).

(47)   *mit*
       with
       'with'

```
CS 1:           P[pre]
              /    |    \
   ADPOS-S_BASE ADPOS-T_BASE[pre] CASE-F_BASE
        |            |              |
       mit         +PREP          .Dat
```
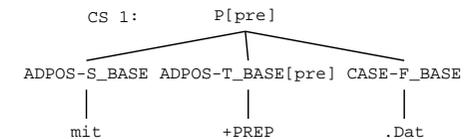
Since the stylesheet creates the snapshots based on the current version of the
grammar, the snapshots are automatically updated for the entire documenta-
tion by running the stylesheet.

Note that example phrases are often ambiguous, i.e. our grammar assigns
several analyses to them. The question is how the desired analysis is selected
for the snapshots of such ambiguous examples. One possibility is to carefully
choose the examples, in such a way that either they do not receive more than
one analysis or else the differences between the analyses should not matter
in the current context (this is our current approach). Alternatively, the exam-
ples could be annotated by target features, which are used to filter out unde-
sired analyses. (Annotation with the number of analyses at time of writing the
documentation could further be used to generate warnings in case a different
number of analyses comes up.)

5.4.5   Indices

In our approach, the documentation does not follow the grammar structure but
assembles grammar code from different modules. Moreover, the documentation
may refer to partial rules only (or macros/templates). That is, the complete
documentation of an entire rule can be spread over different sections of the
documentation.

User-friendly documentation therefore has to include an index that asso-
ciates a grammar rule (or macro or template) with the documentation sections
that comment on this rule. That is, besides referencing from the documentation

to the grammar (by copying), the documentation must also support referencing (indexing) from various parts of the grammar to the relevant parts of the documentation.

Index of rules, macros, and templates   Such an index can be generated automatically based on XML tags. Any reference to a rule/macro/template definition, via the <code> tag (cf. p. 115), triggers an entry in the index. The index thus contains a list of rules, macros, and templates, which are associated with the page numbers (of the final documentation) where the corresponding code fragments occur.

We provide separate indices for c-structure elements (i.e. rules and macros) and f-structure elements (templates). The index entries are sorted according to their syntactic category. For instance, the template @COUNT-NOUN is called by lexicon entries of nouns, hence it is part of the file 'np.tmpl.lfg' and therefore listed under the header 'NP'.

Below we show an extract from the index of templates of the DP documentation (the complete index is displayed in appendix C).

Index of features   The above indices of rules/macros and templates only comprise pages that display the definition of a rule, macro, etc. That is, a purely textual reference to a rule is not indexed. The reason is that we assume that the documentation sections that are relevant to the understanding of a certain rule are exactly those that cite the definition of this rule.

In the case of features, the situation is different. In the previous chapter, we argued that f-structures are global data, since rules and lexicon entries operate on the same data structure (cf. sec. 4.2.2.4). That means, if the f-structure analysis of a construction is modified, all other constructions are potentially

affected. As a consequence, each occurrence of a feature is potentially relevant.

Therefore, we provide an index of features that lists any textual reference to a feature. Paragraphs that are especially important for the understanding of a certain feature can be marked manually. In our documentation, the corresponding page numbers of important paragraphs are underlined.

Sometimes, the referenced feature is part of a complex feature path, such as CHECK _LEX _PRON _APP. Such entries are sorted according to the first feature, i.e. all feature paths starting with CHECK are listed under the header feature CHECK.

Below we show an extract from the feature index of the DP documentation (for the complete index, see appendix C). (Note that our index only lists occurrences of features in the text, i.e. it does not list feature occurrences in the grammar code. XLE provides an index-based search that can be used instead.)

Index of OT marks   Similarly to the index of features, any textual reference to OT marks is indexed in a separate index, cf. appendix C.

These indices can be generated once for the entire documentation. Or else the documentation sections of the different syntactic categories receive separate indices. In this case, the categorial sorting of the index entries may indicate to what degree the syntactic categories are interwoven. For instance, the indices of the DP documentation contain entries that adhere to the category NP, but no entries that adhere to the category CP.

5.4.6   Testsuites

As already mentioned, all example sentences in the source documentation are marked by a special tag <ex>. The example markup can be used to automatically generate a testsuite. In this way, the grammar writer can easily check

whether the supposed coverage, as reported by the documentation, and the actual coverage of the grammar are identical.

Consider the following example tag, featuring the ungrammatical example *mit eine Frau*.

```
<ex judge='*' cat="PP[std]">
<g>mit eine Frau</g>
<l>with a[ACC] woman</l>
<e></e>
</ex>
```

To create a testsuite, the content of the tag <g> (which represents the German example) is extracted. The attribute 'cat' of the <ex> tag specifies the phrasal category of the example, e.g. 'PP[std]' (by default, our grammar assumes that the input string represents a sentence; otherwise, the phrasal category has to be stated explicitly). Moreover, the attribute 'judge', with values '*', '?', '??', etc., marks ungrammatical or deviant, marginal data. This markup is used to create negative test items, which should be rejected by the grammar.

It is also possible to create specialized testsuites. For instance, one might want to create a testsuite of interrogative DPs. In this case, only test items that are marked by the attribute 'cat="DP[int]" ' are extracted. Or else, to test the PP, all examples occurring within the PP documentation are considered (regardless of the specification of the attribute 'cat').

To sum up, the information that is encoded by XML-based documentation can be exploited in various ways to support the reader of the documentation and the grammar writer.

Except for the different output formats, all of the above features are implemented. Moreover, the syntactic category DP is documented according to the structure described above (cf. sec. 5.4.1) (Part III of this dissertation presents the DP documentation).

Up to now, we have seen how to create and exploit XML-based grammar documentation. The next section addresses the question of how to maintain such a type of documentation.

5.5   Maintainability of the Documentation

A grammar implementation is a complex software project and, hence, often needs to be modified, e.g. to fix bugs, to widen coverage, to reduce overgeneration, to improve performance, or to adapt the grammar to specific applications. Obviously, the documentation sections that document the modified grammar parts need to be modified as well.

In our approach, grammar code and documentation are represented by separate documents. Compared to code-internal comments, such code-external documentation is less likey to remain up-to-date, because it is not as closely associated with the code (McConnell 1993, ch. 19.2). This section discusses techniques that could be applied to support maintenance of our XML-based documentation.

Automatic update   In some respects, the (output) documentation is updated automatically by our XML/XSLT-based approach. XSLT operates on the most recent version of the grammar, therefore all grammar-related elements within the output documentation that are generated via XSLT are automatically synchronized to the current grammar. These elements are the following.

- The pieces of code that are copied to the output documentation via XSLT

- The snapshots that illustrate the example sentences

- The indices

Manual update   The genuine documentation part (the documenting text) needs to be updated manually.

If the documentation were code-internal (i.e., realized by comments), this task seems to be obvious. Whenever some grammar code is modified, the grammar writer would need to check the comments adjacent to the modified code and to adjust them, if necessary. (Note that our grammar code does comprise comments, which have to be updated this way. This section rather addresses the maintainability of the external, XML-based documentation.)

With code-external documentation, the task is more complicated. It is not obvious where in the external documentation the modified code is mentioned and which text parts must be adjusted. (Note, however, that even with code-internal documentation the necessary updates are not necessarily restricted to adjacent comments.)

One (ineffective) method would be to keep a record of all rules, macros, etc. that have been modified by the grammar writer. In a next step, one would have to search all documentation sections for references to the modified rules (e.g. guided by the rule index). This method, however, is error-prone because a manually generated rule record may remain incomplete, or the manual search may be performed incorrectly.

Therefore, (semi-)automatic support of the manual update is important. In the following sections, we outline two kinds of support, one provided by XSLT, the other by a tool such as the UNIX command 'diff'.

Support of the manual update: XSLT   We distinguish three types of grammar modifications. (i) An existing rule (or macro or template) is deleted. (ii) An existing rule is modified. (iii) A new rule is added to the code.

In case (i), the XSLT processing indicates whether a documentation update is necessary.

- Whenever a rule is documented by the external documentation, it is (usually) referenced by the <code> tag. Suppose such a rule is deleted from the code, i.e. the referenced 'id' attribute does not exist any more. In this case, the XSLT processing prints out a warning that the referenced element could not be found.

  The grammar (or documentation) writer then needs to update the corresponding part of the documentation, e.g. by deleting the <code> tag and adjusting the textual documentation, if appropriate.

- Certain rules of the code may not be documented in the (external) documentation. If such a rule is deleted, no warning is given by the XSLT processing. However, no documentation update is necessary in this case.

In case (ii), i.e. if an existing rule is modified rather than deleted, there is no way for the XSLT processing to determine whether this affects the documentation. Hence, it does not print a warning (regardless of whether the rule is referenced or not). How can the grammar writer know whether the documentation has to be updated or not in this case? Here, the UNIX command 'diff' may help, which is used to compare (text) files. For our purposes, it is applied to different types of files.

Support of the manual update: 'diff' on LaTeX files   The output LaTeX files include copies of the referenced parts of grammar code. These are generated on the base of the most recent grammar version.

Suppose that the grammar has been modified while leaving the documentation text untouched. Now, if the LaTeX files are newly generated, the only parts that may possibly have changed are the parts citing grammar code. These parts can be located by means of the 'diff' command. (In addition, the snapshots and indices may have changed as well, see below. However, these are contained in extra files and can be ignored by the 'diff' command.)

If such changes between the last and the current LaTeX files have occurred, these changes indicate that the surrounding documentation sections may need to be updated. If no changes have occurred, despite the grammar modifications, this implies that the modified parts are not documented in the (external) documentation and, hence, no update is necessary.

By this technique, the grammar writer gets precise hints as to where to search for documentation parts that may need to be adjusted.

Support of the manual update: 'diff' on Prolog files   A further possibility is to compare the c-structure and f-structure analyses of example sentences. XLE supports different output formats of these analyses, either in postscript format (which we use for the snapshots (cf. sec. 5.4.4) or in Prolog format.

After a grammar modification, the c-structure and f-structure analyses of certain example sentences may have changed. These changes can be detected by running 'diff' on the Prolog representation. Again, such changes indicate that an update of the surrounding documentation text may be necessary.

Support of the manual update: 'diff' on XLE index   Finally, in case (iii), a newly added rule can be detected by a 'diff' command on the XLE index (which lists all grammar rules, macros, and templates). The grammar writer then has to decide whether and where to add a corresponding documentation section.

To sum up, maintenance of the documentation text can be supported by techniques that give hints as to where the text needs to be adjusted. In the scenarios sketched above, the grammar writer would first modify the grammar only and generate some new, temporary output documentation. Comparing the current with the last version of the output documentation would yield the desired hints. After an update of the documentation text, a second run of the XSLT processing would generate the final output documentation.

5.6   Summary

In this chapter, we presented a documentation method that accommodates the grammar-specific restrictions discussed in the previous chapter. The most prominent aspect of this method is the mutual independence of the grammar code and its documentation. As a consequence, the hierarchical structure/organization of the code can be documented by linear, user-friendly documentation.

In our approach, the grammar code and documentation text are stored in separate files. Both types of files are enriched by XML markup. An XSLT processing step generates the final output documentation (a postscript or PDF file) on the basis of the XML source files.

The XML markup can be exploited in various ways to further readability of the documentation. Readability is eased by (i) documentation that varies as to the degree of detail they deliver; (ii) different output formats (postscript, PDF, HTML); (iii) display of partial grammar rules, thus enabling the reader to focus on the lines of code that are actually relevant in the current context; (iv) illustrating snapshots of example sentences; (v) indices of rules, macros, templates, features, and OT marks.

In addition, the XML markup can be used to automatically generate testsuites, based on example sentences. Such testsuites, along with user-friendly documentation, provide important support for the grammar writer.

Finally, we addressed the question of how to maintain code-external documentation, i.e. the question of which documentation sections need to be updated after a grammar modification.

Our approach supports (partial) automation, in that all grammar-related elements are automatically synchronized to the current grammar. Further, if rules have been deleted, the XSLT processor prints a warning. If, instead, rules are modified or added, the UNIX command 'diff' can be applied to various source files, providing hints as to where the documentation text may need to be adjusted.

We now come to Part III of this dissertation, which represents some example documentation generated by our proposed method.

Part III

Example: Documentation of the German DP

Chapter 6

Preliminaries

Contents

This chapter supplies the reader with information that eases the understanding of the DP documentation chapters. This information concerns both the grammar and the documentation, namely (i) the architecture of the grammar (sec. 6.1); (ii) encoding conventions that have been applied in the implementation of the grammar (sec. 6.2); (iii) the structure of the documentation (sec. 6.3); (iv) encoding conventions of the documentation (sec. 6.4).

6.1   Grammar Structure

In this section, we present the overall organization of the grammar. We distinguish three different types of files, containing (i) the grammar code, (ii) the tag lexicon, and (iii) the stem lexicons. (For the terms "tag lexicon" (cf. sec. 3.2.4) and "stem lexicon" (cf. sec. 3.2.5).)

(i) Grammar code files   The code of the German grammar is quite extensive. We therefore decided to split the code into parts. Each part corresponds to a syntactic category such as DP, or NP. Moreover, each part is distributed across three files, which contain: (i) the rules (including sublexical rules) and macros of that category; (ii) the corresponding templates; and (iii) the corresponding lexicon entries. That is, there are files such as dp.gram.lfg (containing the DP rules and macros), dp.tmpl.lfg (for DP templates), and dp.lex.lfg (for determiner lexicon entries). (The XSLT copying mechanism makes use of these file names (cf. p. 115).)

We distinguish the following syntactic categories, each of which corresponds to three grammar code files.

- Adjective Phrase (AP)

- Adverbial Phrase (ADVP)

- Clausal Phrase (CP)

- Determiner Phrase (DP)

- Noun Phrase (NP)

- Preposition Phrase (PP)

- Verb Phrase (VP)

- Miscellaneous: rules, macros, and constructions that do not adhere to one specific syntactic category (e.g. coordination rules; header constructions; general templates, such as templates for case assignment or agreement).

The rule/macro files are organized in a top-down manner, i.e. they start with the rule of the maximal projection (e.g. DP), proceed to the intermediate (Dbar) and head projections (D), and end with the sublexical rules. Finally, there are rules and macros that are not part of the core rules. Such rules encode modifiers and complements of the projection (e.g. postnominal modifiers).

Templates are arranged in a similar way. That is, templates that are called by the rule of the maximal projection precede the templates of the intermediate projections, etc.

The lexicon files contain lexicon entries that specify information which is not provided by the external stem lexicon (see below).

Appendix D shows the complete grammar code of the DP implementation, comprising the DP rules and macros (cf. sec. D.1), templates (cf. sec. D.2), and lexicon entries (cf. sec. D.3).

(ii) Tag lexicon files   The tag lexicon (cf. sec. 3.2.4) is stored in a separate file, which contains the entries of all morphological tags, as provided by the morphological analyzer DMOR1, i.e. part-of-speech tags (+NN, +NPROP) and inflectional tags (.Masc, .Sg).

In addition, this file provides the default entries of lemma tags, namely –LUnknown and –Lunknown. Finally, the entries for the tags provided by the guesser (e.g. +MUnknown, .Alphanum) are defined in this file.

We adhere to the following naming conventions for sublexical categories (cf. p. 44).

- Entries for part-of-speech tags end with '-T' ("basic tag"), e.g. ART-T for the part-of-speech tag of articles

- Entries for inflectional tags end with '-F' ("feature"), e.g. GEND-F for gender features

- Entries for lemma tags end with '-S' ("stem", lemma), e.g. ART-S for article stems

(iii) Stem lexicon files   The entries of the stem lexicons (cf. sec. 3.2.5) are provided by an external module. (The source files are stored in a database, the IMSLex, cf. Lezius *et al.* 2000, and are mapped to the XLE-specific notation, cf. Bröker and Dipper 1999.) The entries in these lexicon files specify:

- Subcategorization information for verbs, adjectives, nouns (Eckle-Kohler 1999)

- Noun types (e.g. mass nouns such as *Wasser* 'water', or measure nouns, such as *Liter* 'litre')

- Name types (first names, last names, and names of cities and countries)

Most of this information has been acquired (semi-)automatically. In consequence, the external stem lexicon files contain errors and, more importantly, many gaps. For this reason, there are manual lexicon entries (e.g. in np.lex.lfg) which overwrite and/or complete the information provided by the stem lexicon files.

The following table lists the different types of lexicons that implement the interface between the morphological tags and the grammar.

Tag Type	Corresponding Lexicon
Lemma tag (e.g. *Wasser*)	manual lexicons (e.g. np.lex.lfg) + external lexicons (IMSLex)
Part-of-speech tag (e.g. +*NN*)	manual tag lexicon
Inflection tags (e.g. .*Neut*)	manual tag lexicon

Table 6.1: Overview of lexicon types

6.2   Grammar: Encoding Conventions

In this section, we give examples of encoding conventions applied in the implementation of our grammar in order to ease the understanding of the grammar code and output that are displayed in the following documentation chapters.

One type of convention concerns the notation of templates and macros. The other concerns the encoding of features that are technically motivated.

Notation conventions

- Parameters of macros and templates all start with an underscore (e.g. '_case'). Canonical parameters are written in lower case.

```
CASE(_case) =  (^CASE) = _case .
```

In contrast, parameters that are input to lexical (rewrite) rules are written in capital letters ('_COMMON-RESTR'), cf. the passive template below.

```
PASSIVE-OBJ-TO-SUBJ(_COMMON-RESTR _ACTIVE-RESTR) =
  "lexical restrictions that are modified by rewrite rules"
  _COMMON-RESTR
```

```
  { _ACTIVE-RESTR "restrictions for active"
  | "passive"
    (^OBJ) --> (^SUBJ)
    (^SUBJ) --> NULL
    (^CHECK _PASSIVE) = +
  }.
```

- Many templates come in two variants, one with an additional parameter for the specification of a designator, the other one without such a parameter. The template exhibiting the additional parameter contains the suffix '_desig' in its name, cf. the gender template below (the parameter _desig may, e.g., be instantiated by '(^OBJ)' or simply '^').

```
GEND(_gend) =
    @(GEND_desig ^ _gend) "default designator: ^ ".

GEND_desig(_desig _gend) =
    (_desig GEND) = _gend.
```

Features that are technically motivated   Many features that appear in the output f-structures of our implementation are motivated by technical rather than linguistic considerations. For instance, whenever the passive rewrite rule has been applied, a special feature, CHECK _PASSIVE, is introduced to the f-structure of the verb (see above). This feature determines the form of the auxiliary which is to be used for (analytic) passive in German.

That is, the feature CHECK _PASSIVE plays an important grammar-internal role, in the interaction of the rules, but does not convey any functional or morpho-syntactic information. (Instead, the functional and morpho-syntactic information of the passive construction is represented by the PRED feature of the verb and the features projected by the auxiliary.)

Such technically motivated features are called "CHECK features" in our documentation, because they are all embedded under a feature CHECK (which may occur in f-structures of all types). To indicate them clearly as special features, they all start with an underscore, e.g. [CHECK _PASSIVE +].

6.3   Documentation Structure

This section addresses the organization of the documentation. As already mentioned, the documentation of a syntactic category is devided into five parts, which focus on different aspects (cf. p. 120). We repeat these aspects below and indicate the corresponding DP chapters.

- Chapter 7: Linguistic introduction. (Note that since the documentation focuses on the implementational aspects, theoretical literature has only been considered if relevant to the implementation.)

- Chapter 8: Basics of the implementation.

- Chapter 9: Details of the f-structure analysis.

- Chapter 10: Details of the c-structure analysis.

  The c-structure documentation proceeds top-down, i.e. it starts with the DP rule and ends with the sublexical rules (similar to the structure of the DP rule/macro file).

- Chapter 11: Overview. In this dissertation, this part serves as a kind of (brief) summary of the complete DP documentation.

Note that the highly detailed documentation is further structured. Certain parts are marked by a magnifying glass as additional background information—as exemplified by this note.

The documentation chapters are completed by four appendices.

- Appendix A contains corpus data from the Frankfurter Rundschau (the FR corpus is delivered by the European Corpus Initiative, URL: `http://www.elsnet.org/resources/eciCorpus.html`).

  The entire FR corpus has been automatically annotated at the IMS by a tagger (Schmid 1994). The tags encode part of speech and lemma (according to the tagset described in Schiller *et al.* 1999). The FR corpus comprises about 40 million tokens.

  Furthermore, parts of the FR corpus have been manually annotated in the NEGRA project by trees encoding syntactic categories and functions (Skut *et al.* 1997). The NEGRA corpus contains about 350,000 tokens (20,000 sentences).

  We acquired our corpus data by means of the query tools CQP (Corpus Query Processor, cf. Christ *et al.* 1999), and TIGERSearch (Lezius 2002).

  Corpus data are always associated with examples in the text. For instance, the example phrase in (48) refers to the corpus sentence with the identifier 'DP7.1'.

  (48)   (see corpus example DP7.1, Appendix A)
          vieles Gegenwärtige
          much present
          'many aspects of these days'

The appendix A displays the corpus sentences, highlighting relevant constituents.

  DP7.1   Durch seine höchst riskante und triftige Äquilibristik von historisch-politischem Ernst und spielerisch-ästhetischer Selbstreflexion stellt es **vieles Gegenwärtige** in den Schatten .

- Appendix B: List of tables of the DP documentation.

- Appendix C: Index of features, OT marks, rules, macros, and templates of the DP documentation (cf. sec. 5.4.5).

- Appendix D: The complete grammar code of the German DP.

6.4   Documentation: Encoding Conventions

We adhere to some notation conventions in the documentation. These are described briefly in this section.

Features and values   Features and values are printed in a special font, e.g. ($\uparrow$CASE) = acc.

Macros and templates   Macro and template names are preceded by '@' in the text, e.g. @QUANT-EXPRESSION.

Copied code   The copy mechanism may copy entire rules (macros, templates) from the grammar code or only parts of them (cf. sec. 5.4.3). If only parts are copied, the copied part either represents a complete disjunct (i.e. represents a valid expansion of the rule); such parts are preceded by '... |' and followed by '| ...' (to indicate the disjunction).

```
QUANT-EXPRESSION(_cat _type) =

      ...
    | "articles (inserted under DET)"
      e: _type = std;
      @(QUANTart _cat)
    | ...
```

Or else the copied part is an incomplete fragment of the rule; such parts are enclosed by '...'

```
DP[_type $ {std int rel}] -->
      ...

      ( "optional constituents may follow DP"
       @(DPpost) )
      ...
```

(Comments within copied code are are put in a different colour, thus mirroring the Emacs LFG-mode.)

Manual vs. copied code   Besides copied code, our documentation features rules, macros, templates, and lexicon entries that are generated manually. Usually, the context makes clear whether the displayed code is copied or manually generated.

In addition, manual code can be distinguished from copied code by the shape of the '↑/↓' arrows. Manual code makes use of '↑' and '↓', copied code uses '^' and '!'.

As an example, compare the presentations of the template @PRENOM-GENITIVE.

- Manual code

```
PRENOM-GENITIVE =
      (↑SPEC POSS) = ↓
      "EX: Karls Hund (Karl's dog)"
      @(CASE_desig ↓ gen)
      "must be morphologically marked:"
      "EX: *Karl Hund (Karl dog)".
```

- Copied code

```
PRENOM-GENITIVE =
      (^SPEC POSS) = !
      "EX: Karls Hund (Karl's dog)"
      @(CASE_desig ! gen)
      "must be morphologically marked:"
      "EX: *Karl Hund (Karl dog)".
```

A further difference is that copied lexicon entries always contain the keywords 'xle' or '*'. Compare the manual entry of *für* 'for' and the copied entries of *für* 'for' and *zu* 'to'.

- Manual entry

```
für     Ppred  @(OBJ_subc %stem)
               @(CASE_desig (↑OBJ) acc)
```

- Copied entries

```
für     !Ppred * @(OBJ_subc %stem)
               @(CASE_desig (^OBJ) acc); ETC.


zu      !ADPOS-S xle @(PREP-DIR %stem); ETC.
```

Manual vs. XLE c-structures and f-structures   C-structures and f-structures that are generated automatically (as snapshots) can be distinguished from manual ones by XLE labels. XLE c-structure snapshots are always preceded by labels 'CS 1:', 'CS 2:', etc. The outermost f-structure of an XLE f-structure snapshot is labelled by '0' (or some other number). Moreover, the input string of the f-structure is quoted.

Manual trees and f-structures are never labelled this way.

As an example, compare the c-structure and f-structure analyses of the proper noun *Hans*.

Manual Structures	XLE Snapshots
NAMEP	CS 1: NAMEP
|	|
Hans	Hans
	"Hans"
[PRED 'Hans']	0[PRED 'Hans']

Abbreviated tags   Abbreviated tags are used in the literal translations of German example sentences, as exemplified by (49).

(49)   *mit einer Frau*
       with a[DAT] woman

These tags encode (i) morpho-syntactic information, (ii) categorial information, (iii) positional information, or (iv) functional information.

(i) Morpho-syntactic information, e.g. ART (article), NEUT (neuter)

- The tag is enclosed by '[...]' and is attached to the word it marks, e.g. *das Auto* 'the[NEUT] car' (*das* 'the' is an article in neuter form).

- Multiple tags are separated by a comma, e.g. *das Auto* 'the[NEUT,SG, NOM] car'. Relative order of multiple tags: person > gender > number > case > inflection type.

- Ambiguous tags are separated by a slash, e.g. *das Auto* 'the[NEUT,SG, NOM/ACC] car' (*das* 'the' is the form of both the nominative and accusative).

This is the complete list of morpho-syntactic tags.

	Abbreviation Tag	Meaning
Person	1	first
	2	second
	3	third
Gender	MASC	masculine
	FEM	feminine
	NEUT	neuter
Number	SG	singular
	PL	plural
Case	NOM	nominative
	GEN	genitive
	DAT	dative
	ACC	accusative
Inflection type	ST	strong
	WK	weak
	UNINFL	uninflected/invariant
Part of speech	ART	article
	DEM	demonstrative
	INDEF	indefinite
	INT	interrogative
	POSS	possessive

Table 6.2: Morpho-syntactic tags within example sentences

(ii) Categorial information, e.g. PP (prepositional phrase)

- The constituent is bracketed by '[...]', and the categorial tag precedes the closing bracket, e.g. *Maria wohnt in Stuttgart.* 'M. lives [in Stuttgart PP].' (*in Stuttgart* represents a PP).

- DP categories may be marked by a tag encoding the DP's case; e.g. *wegen der Leute* 'because_of [the people GEN]' (*der Leute* 'the people' represents a genitive DP).

- Categories are sometimes numbered if there are several instances of the same category; e.g. *eine Anfrage im Landratsamt nach Meßwerten* a request [in_the district_office PP1] [for measure_values PP2].

- Besides purely categorial information, the following special tags occur: EXPL (expletive); REFL (reflexive pronoun); PART (particle); VPART (verb particle).

(iii) Positional information, e.g. SPEC-DP (specifier of DP) Similar to categorial marking, the constituent is bracketed by '[...]', and the positional tag precedes the closing bracket, e.g. *Karls Auto* '[K. SPEC-DP] car' (*Karls* occupies the DP's specifier position).

Positional tags comprise: SPEC-DP (the constituent occupies the DP's specifier position), SPEC-CP (specifier position of CP); LEFT-DISL (left dislocation); VORF (Vorfeld), MITTELF (Mittelfeld), NACHF (Nachfeld).

(iv) Functional information, e.g. SUBJ (subject) Again, the constituent is bracketed by '[...]', and the functional tag precedes the closing bracket, e.g. *Maria wohnt in Stuttgart.* '[M. SUBJ] lives in Stuttgart.' (*Maria* represents the subject).

Functional tags comprise: SUBJ (subject); OBJ, OBJ2 (secondary/indirect object); ADJ-GEN (genitive modifier); APP (appositive).

We now turn to the actual DP documentation, starting with a linguistic introduction.

The source files of the documentation consist of XML documents, which are automatically converted to LaTeX. As a result, the layout is restricted in certain aspects, e.g. tables and figures may not float, and references to sections are always enclosed by parantheses and start with 'cf.', e.g. (sec. 6.1).

Chapter 7

Basics of the German DP

Contents

This chapter presents an introduction to linguistics aspects of the German DP. That is, constituents that belong to the DP projection are discussed as well as their relevant properties. The constituents are: the category DP, the specifier position SPEC-DP, and the head D. Properties of other projections are only addressed if they are directly related to the DP projection. For instance, inflectional properties of attributive adjectives are discussed here because they are determined by the determiner.

In the literature outside of LFG, quite a lot of work can be found about DP analyses in general and the DP in German in particular (for the German DP, cf. Bhatt 1990, Gallmann 1996, Haider 1988, Netter 1994, Olsen 1991, Pafel 1994).

In the context of LFG, however, details of the internal structure of nominal phrases are often left open. There is some amount of literature about the analysis of the DP in Northern Germanic languages, cf. Börjars (1998), Börjars *et al.* (1999). They focus on the feature DEF, which in these languages can or must be expressed via a noun suffix. Among other things, this feature determines the inflection type of attributive adjectives within the DP: [DEF +] triggers weak adjective agreement, [DEF –] triggers strong adjective agreement (for the notion of strong/weak adjective (cf. sec. 7.2)).

German, however, does not have a noun suffix that indicates definiteness. Furthermore, although German also has weak and strong adjective agreement, most (non-LFG) analyses of the German DP assume that definiteness plays no role in adjectival inflection (for a different view, see Pafel 1994). This is easily seen by the behaviour of the indefinite article *eine* 'a', which triggers strong or weak adjectival inflection depending on case. So, clearly, the German DP differs from DPs in Northern Germanic languages in important aspects. Our presentation of the German DP therefore mainly refers to literature not related to LFG.

In this chapter, we discuss important properties of the German DP and their (theoretical) analyses in the literature. The question of how our implementation relates to these analyses is addressed in the following chapter.

We present the following topics:

- Basic c- and f-structure properties of the German DP (sec. 7.1)

- Determiner and adjective inflection: strong vs. weak (sec. 7.2)

- Determiners and quantifying adjectives (sec. 7.3)

- Pronouns (sec. 7.4)

- Genitive DPs (sec. 7.5)

7.1   C- and F-Structure

This section addresses basic c- and f-structure properties of the German DP. The basic annotated c-structure for a DP looks as follows.
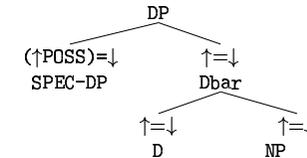
In the following sections, we briefly present canonical instances of the terminal nodes of the above tree.

D projection   D, the head of the DP, can be occupied by different kinds of determiners (50) or quantifying expressions (51).

(50)   *eine Frau*
       a   woman

(51)   *alle Frauen*
       all   women

The determiner introduces a feature SPEC that indicates the type of determiner (e.g. indefinite). The NP is the f-structure co-head of D, cf. the c-structure and f-structure of the above example *eine Frau* 'a woman'.
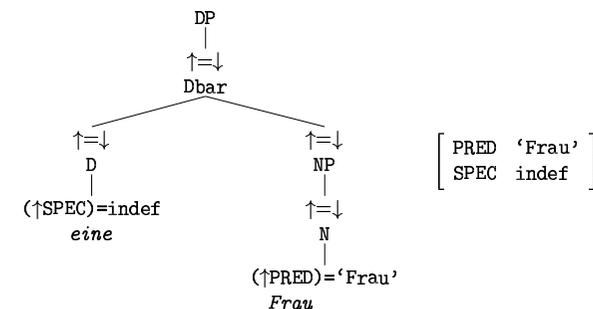
SPEC-DP position SPEC-DP, the specifier position of DP, can be filled by genitive DPs (52). (Note that the term 'SPEC-DP' denotes a property of c-structure elements, whereas the feature SPEC, introduced by determiners, refers to f-structure.)

(52) *Karls   Frau*
      [K. GEN] wife
      'Karl's wife'

They function as possessors, i.e. their f-structures are represented by the function POSS (Falk 2001, p. 73).

```
                    DP
           ┌─────────┴─────────┐
      (↑POSS)=↓              ↑=↓
      SPEC-DP                Dbar
         │                    │
   (↑PRED)='Karl'            ↑=↓
      Karls                   NP          ⎡ PRED  'Frau'          ⎤
                              │           ⎢                       ⎥
                             ↑=↓          ⎣ POSS  [ PRED  'Karl' ]⎦
                              N
                              │
                        (↑PRED)='Frau'
                            Frau
```
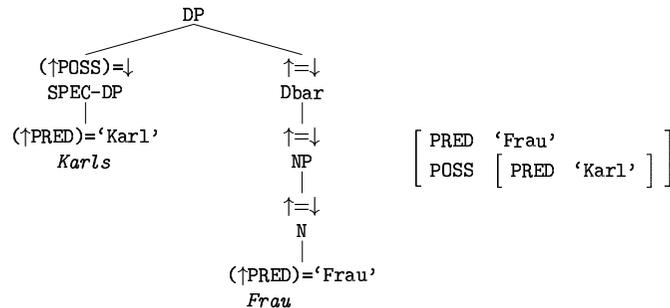
We use the term 'SPEC-DP' throughout the DP documentation to refer to the specifier position of DP. There is, however, no node labelled by SPEC-DP in the c-structure representation of our implementation.

NP projection NP, the f-structure co-head of D, dominates attributive APs and the semantic head N (53).

(53) *eine junge Frau*
      a   young woman

```
                         DP
                          │
                         ↑=↓
                        Dbar
              ┌───────────┴───────────┐
             ↑=↓                     ↑=↓
              D                       NP
              │              ┌─────────┴─────────┐
      (↑SPEC)=indef    ↓∈(↑ADJUNCT)            ↑=↓
          eine              AP                   N
                            │                    │
                      (↑PRED)='jung'      (↑PRED)='Frau'
                          junge                 Frau
```

```
⎡ PRED     'Frau'              ⎤
⎢ SPEC     indef               ⎥
⎢ ADJUNCT  { [ PRED  'jung' ] }⎥
⎣                              ⎦
```
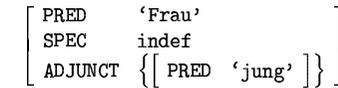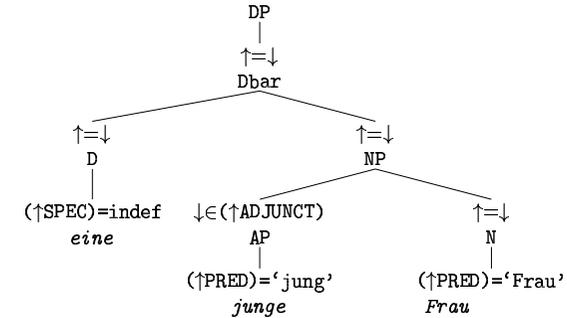
An issue that is often discussed in the literature is whether there is a full DP projection even if no specifier or determiner is overtly expressed, as in the case, e.g., of mass nouns or predicatives (cf. the discussion in Bhatt 1990, ch. 9).

Many authors assume that, e.g., argument selection need not refer to the categorial status of DP or NP, but rather to some functional feature indicating that a nominal phrase is "functionally complete" (Netter 1994) or "argument compatible" (= "argumentfähig", Haider 1988). Some authors nevertheless postulate a DP projection for any nominal phrase (Bhatt 1990, ch. 9).

In LFG, argument selection is defined at the level of f-structure, hence the categorial status (DP or NP) does not play a role here. (In our implementation, any nominal phrase projects a DP.)

In the following sections, we first address different aspects of the head D, concerning inflectional properties, (sec. 7.2) and (sec. 7.3), and distributional properties (sec. 7.4). We then present restrictions on the specifier position SPEC-DP (sec. 7.5).

7.2 Determiner and Adjective Inflection

This section addresses agreement phenomena involving the determiner, attributive adjectives, and the head noun. We distinguish two types of agreement: (i) adjective–noun and determiner–noun agreement, concerning the features gender, number, case; (ii) determiner–adjective agreement, concerning the feature "inflection type" (as we call it).

(i) Adj–N and D–N agreement   In attributive position, a German adjective agrees in gender, number, and case with its head noun, cf. (54), (55), (56), (57). German nouns are inherently/lexically marked for gender.

(54)   *süßer*   *Wein*
        sweet[MASC,SG,NOM] wine

(55)   *süßes*   *Bier*
        sweet[NEUT,SG,NOM/ACC] beer

(56)   *süßen*   *Wein*
        sweet[MASC,SG,ACC] wine

(57)   *süße*   *Weine*
        sweet[PL,NOM/ACC] wines

Likewise, a determiner agrees with its head noun (and with attributive adjectives, if present) (58), (59).

(58)   *der*   *Wein*
        the[MASC,SG,NOM] wine

(59)   *das*   *Bier*
        the[NEUT,SG,NOM/ACC] beer

Inflection type of D   Besides gender, number, and case, a fourth parameter is involved, which we call "inflection type". We first clarify the notion "inflection type" and then address agreement between determiners and adjectives.
   Determiners come in two inflection types.

- Either they exhibit the so-called "strong" inflection (strong inflection is indicated by 'ST'): *der/des/dem/den*, etc. 'the[MASC,SG,NOM/GEN/DAT/ACC,ST]'. (This type of inflection is also called "pronominal" inflection, since it is similar to the inflection of pronouns, cf. the interrogative pronoun *wer/wessen/wem/wen* 'who[SG,NOM/GEN/DAT/ACC]'.)

- Or they are uninflected: *viel* 'much[UNINFL]', *manch* 'some[UNINFL]'.

An overview of the forms of the definite article *die* 'the', featuring strong inflection, is given in the following table. (In German, gender is differentiated only in the singular.)

CASE	MASC,SG	NEUT,SG	FEM,SG	PL
Nominative	der	das	die	die
Genitive	des	des	der	der
Dative	dem	dem	der	den
Accusative	den	das	die	die

Table 7.1: The German definite article (= strong/pronominal inflection)

Certain determiners come with different inflection types. For example, besides uninflected *viel* 'much' as in (60), there is an inflected variant as in (61).

(60)   *viel*   *Arbeit*
        much[UNINFL] work

(61)   (see corpus example DP7.1, Appendix A)
        vieles   *Gegenwärtige*
        much[NEUT,SG,NOM/ACC] present
        'many aspects of these days'

Furthermore, the indefinite article *eine* 'a' is obligatorily uninflected in the case of [MASC,SG,NOM] and [NEUT,SG,NOM/ACC]. That is, the uninflected form *ein* occurs in examples such as (62), rather than the putative regular form *einer*, cf. (63). In all other cases, the indefinite article inflects: *eines/einem*, etc., as in (64).

(62)   *ein*   *Mann*
        a[UNINFL] man

(63)   **einer*   *Mann*
        a[MASC,SG,NOM] man

(64)   *eines*   *Mannes*
        a[MASC,SG,GEN] man

An overview of the forms of the indefinite article *eine* 'a', mainly featuring strong inflection, is given in the following table. Uninflected forms of the indefinite article are highlighted. (There is no plural form of the indefinite article.)

CASE	MASC,SG	NEUT,SG	FEM,SG
Nominative	**ein**	**ein**	eine
Genitive	eines	eines	einer
Dative	einem	einem	einer
Accusative	einen	**ein**	eine

Table 7.2: The German indefinite article (= strong or uninflected)

(ii) D–Adj agreement: co-occurrence of inflection types As we have seen above, an attributive adjective agrees with its head noun in gender, number, and case. In addition, the inflection of an attributive adjective is also sensitive to the inflection type of its determiner.

- If preceded by an inflected (= strong) determiner, the adjective comes in its so-called "weak" or "nominal" form (65). (In older stages of the language, this inflection type was restricted to nouns, hence the name "nominal", cf. Olsen 1991.)

  (65)   *der   süße   Wein*
         the[ST] sweet[WK]   wine

- If preceded by a non-inflected determiner, the adjective itself exhibits strong (pronominal) inflection (66), (67).

  (66)   *viel   süßer   Wein*
         much[UNINFL] sweet[ST]   wine

  (67)   *ein   süßer   Wein*
         a[UNINFL]   sweet[ST]   wine

  (There are a few adjectives that do not inflect, e.g. *lila* 'purple'. These can occur after strong or uninflected determiners.)

The inflectional endings of adjectives are given in the following tables. The first table shows strong (pronominal) inflection, the second shows weak (nominal) inflection. Note the similarity between the inflection of strong adjectives and the inflection of the definite and indefinite articles.

CASE	MASC,SG	NEUT,SG	FEM,SG	PL
Nominative	-er	-es	-e	-e
Genitive	-en	-en	-er	-er
Dative	-em	-em	-er	-en
Accusative	-en	-es	-e	-e

Table 7.3: Strong (pronominal) German adjective inflection

CASE	MASC,SG	NEUT,SG	FEM,SG	PL
Nominative	-e	-e	-e	-en
Genitive	-en	-en	-en	-en
Dative	-en	-en	-en	-en
Accusative	-en	-e	-e	-en

Table 7.4: Weak (nominal) German adjective inflection

Consequently, attributive adjectives that follow the indefinite article show a mixed inflection type: strong inflection after (uninflected) *ein* and weak inflection in all other cases, i.e. after *eines/einem*, etc. Traditionally, this inflection type is regarded as a type of its own, called "mixed inflection" (Drosdowski 1995, p. 279, Müller 1999, ch. 7.2). The determiner *kein* 'no' and all possessive determiners, *mein* 'my', *dein* 'your', etc., exhibit the same inflection properties as the indefinite article.

Authors who do not assume a mixed inflection type fall in two classes: Some assume that (uninflected) *ein* (and *kein*, *mein*, etc.) are weak determiners (Pollard and Sag 1994, ch. 2.2), others analyze *ein* as uninflected (Netter 1994) (as we shall do in our analysis). The first approach has the drawback that *ein* constitutes the only instance of a weak determiner, whereas within the second approach, *ein* behaves like any uninflected determiner.

The following generalization emerges from the above data. In a DP, the feature "strong" is represented (i) on the head D if present and if inflected, (ii) on attributive adjectives otherwise (similarly assumed, e.g., by Bhatt 1990, ch. 9.4, Olsen 1991). One important conclusion is that determiners and adjectives show complementary inflection.

The following table lists all possible combinations of inflection types as predicted by the generalization.

Determiner	Adjective
strong	weak (or uninflected)
uninflected	strong (or uninflected)
– (no determiner)	strong (or uninflected)

Table 7.5: Co-occurrence of inflection types

Summary A determiner, attributive adjectives, and the head noun agree in gender, number, case, and inflection type. Determiners exhibit strong inflection or they are uninflected; adjectives exhibit strong or weak inflection (certain adjectives never inflect). Determiners and adjectives show complementary inflection.

7.3 Determiners and Quantifying Adjectives

Besides quantificational determiners, German also has quantifying adjectives. The relation between both types of quantifying expressions is the topic of this section.

Quantifying adjectives An example of a quantifying adjective is *viele* 'many', as in (68) and (69).

(68) *die   vielen   Menschen*
the[ST] many[WK] people
'the numerous people'

(69) *sehr viele   Menschen*
very many[ST] people

There are three indications that *viele* 'many' in these contexts is indeed an adjective rather than a determiner: (i) It cooccurs with the definite article; (ii) it exhibits weak inflection after the definite article, just as adjectives do; (iii) it can be modified by the adverb *sehr* 'very', which is also typical of adjectives.

We now want to examine further expressions that could be classified as determiners or quantifying adjectives according to these three criteria. Candidates are:

- Possessives, e.g. *meine* 'my'

- Demonstratives, e.g. *diese* 'this', *dieselbe* 'the same'

- Interrogatives, e.g. *welche* 'which', *wieviele* 'how many'

- Indefinites, e.g. *mehrere* 'some', *irgendwelche* 'any'

In the following, we somewhat loosely use the term "quantifiying expression" to cover all of the candidate types above. Moreover, the term "indefinite" is not intended to convey any semantic implication about the expression in question. Instead, it is used for all quantifying expressions that are neither articles, possessives, demonstratives, nor interrogatives. The advantage of this terminology is that "indefinites" in this sense correspond to all expressions that are marked by +INDEF by our morphology. For example, *alle* 'all' and *keine* 'no' belong to this set as well. An overview of the quantifying expressions in the current version of our morphology is given in the summary (sec. 11.3.2).

The question now is which of the above candidates are determiners and which are quantifying adjectives. Testing a first candidate, e.g., *mehrere* 'some', for the three criteria above, the data show that *mehrere* neither cooccurs with the definite article, compare (70) and (71), nor does it allow for modification by an adverb (72).

(70) *mehrere Menschen*
some[ST] people

(71) **die   mehreren Menschen*
the[ST] some[WK] people

(72) **sehr mehrere   Menschen*
very some[ST] people

The same holds for most of the quantifying expressions in German (for a discussion of the exceptions, see below).

At first sight, these data seem to suggest that most of the quantifying expressions in German (including *mehrere*) are determiners, and that quantifying adjectives such as *viele* 'many' (see the first examples of this section) constitute an exceptional case. However, the fact that a quantifying expression is incompatible with the definite article or with modifying adverbs may well be due to the semantics of the quantifying expression in question and need not be connected to its (syntactic) status as a determiner or adjective at all. Hence, none of the above three criteria can be applied to determine the categorial (adjectival) status of these quantifying expressions.

Co-occurrence of inflection types as the criterion In our implementation, we build on the generalization regarding the co-occurrence patterns of inflection types of determiners and adjectives to derive an alternative, (morpho-)-syntactic criterion for the categorial distinction of (quantificational) determiners and adjectives. That is, we need to determine whether a quantifying expression parallels the inflection of canonical determiners (such as the definite article) or whether it parallels the inflection of ordinary adjectives.

This can be done by testing for the inflection type of a following attributive adjective. If the adjective shows the same inflection type as the quantifying expression in question (e.g., both show strong inflection), then the quantifying expression is a quantifying adjective. Otherwise, if the adjective shows complementary inflection, the quantifying expression must be a determiner.

Applying this test, e.g., to the quantifying expression *mehrere* 'some' reveals that *mehrere* behaves like an ordinary adjective in that it shows the same inflection type as a following adjective, compare (73) (featuring two ordinary adjectives) with (74). Hence, *mehrere* is classified as a quantifying adjective and therefore analyzed as occupying an adjectival position (A).

(73) *intelligente   visuelle   Erweiterungen*
intelligent[ST] visual[ST] extensions

(74) *mehrere strittige   Punkte*
some[ST] contestable[ST] points

In contrast, for the quantifying expression *alle* 'all' the test reveals that *alle* behaves like the canonical determiner *die* 'the': *alle* and the following adjective exhibit complementary inflection, compare (75) and (76). Hence, *alle* is classified as a determiner, occupying the D position.

(75) *die   derzeitigen Regierungschefs*
     the[ST] present[WK] government‗chiefs
     'the present heads of the government'

(76) *alle   politischen Parteien*
     all[ST] political[WK] parties

Multiple quantifying expressions   Furthermore, our criterion reveals that in German, multiple quantifying expressions do occur (also assumed by Bhatt 1990, p. 204ff and Pafel 1994). In DPs such as (77), both *alle* 'all' and *die* 'the' need to be classified as determiners due to their inflectional behaviour. Further examples are (78), (79).

(77) (see corpus example DP7.2, Appendix A)
     alle   die   schönen Definitionen
     all[ST] the[ST] nice[WK] definitions

(78) (see corpus example DP7.3, Appendix A)
     alle   unsere   schönen   Sprüche
     all[ST] our[ST] pretty[WK] sayings

(79) *manch   einem wissenschaftlichen Assistenten*
     some[UNINFL] a[ST] research[WK]   assistant
     'some research assistant'

These examples of complex determiners show that co-occurrence with the definite article does not provide sufficient evidence for the adjectival status of a quantifying expression. Similarly, the third property (modification by an adverb) does not help much: even unambiguous determiners such as *keine* 'no' may be modified (80) (as noted by Pafel 1994).

(80) *in fast   keiner anderen   Metropole*
     in almost no[ST] other[WK] metropolis

Ambiguous quantifying expressions   Our inflection-based criterion also shows that there are ambiguous quantifying expressions, e.g. *beide* 'both', which can be a quantifying adjective (81) or a determiner (82).

(81) *die   beiden   jungen   Frauen*
     the[ST] both[WK] young[WK] women
     'both of the young women'

(82) *beide   genannten   Verfahren*
     both[ST] mentioned[WK] methods
     'both methods mentioned'

Similarly to *beide* 'both', the indefinite article *eine* 'a' can be viewed as an instance of an ambiguous quantifying expression *eine* 'a/one': as the indefinite article ('a'), it inflects like a determiner (83), as an indefinite quantifying expression ('one'), it inflects like a quantifying adjective (84).

(83) *ein   großes Haus*
     a[UNINFL] big[ST] house

(84) (see corpus example DP7.4, Appendix A)
     das   eine   große   Lager
     the[ST] one[WK] big[WK] camp

Summary   German has quantificational determiners and adjectives. In our implementation, the inflection type of a quantificational expression determines its categorial status: the quantifying expression either parallels the inflection of an attributive adjective and, hence, is classified as a quantifying adjective. Or else, they show complementary inflection, then the quantifying expression is classified as a determiner.

Our criterion also reveals that multiple quantifying expressions and ambiguous quantifying expressions do occur in German.

7.4 Pronouns

For reasons that will become clear below, we distinguish three types of pronouns: (i) "determiner-pronouns", (ii) "adjective-pronouns", and (iii) "real pronouns".

7.4.1 "Determiner-Pronouns"

First, note that most of the demonstrative, indefinite, and interrogative determiners have pronominal counterparts, compare (85) vs. (86) (demonstratives), (87) vs. (88) (indefinites), (89) vs. (90) (interrogatives). We call these pronominal counterparts "determiner-pronouns".

(85)   *dieses Haus*
       this   house

(86)   *dieses*
       this

(87)   *alle Häuser*
       all   houses

(88)   *alle*
       all

(89)   *welche Häuser*
       which houses

(90)   *welche*
       which
       'which ones'

One idea brought up in the context of DP analyses is that such determiner-pronouns are in fact determiners. The difference between classical determiners and pronouns reduces to a difference in use: classical determiners are used as "transitive" determiners, i.e. they take an NP as their c-structure complement, while determiner-pronouns are used as "intransitive" determiners, i.e. they represent a full DP on their own (Abney 1987, ch. 4, Olsen 1991). (Our implementation follows this idea.)

Obligatorily transitive determiners   Some English determiners, however, cannot be used as pronouns, namely 'a' and 'the'. In the terminology of the analysis just mentioned, these determiners are obligatorily transitive (Abney 1987, p. 175).

In German, we find similar restrictions. The definite article cannot (in general) be used as a determiner-pronoun, compare the definite article *des* 'the[NEUT,SG,GEN]' in (91) and the putative (ungrammatical) determiner-pronoun (92) (in contrast to the "longer", correct form of the demonstrative pronoun, *dessen* 'this[NEUT,SG,GEN]', as in (93)).

(91)   *Experimente bedürfen des   Einverständnisses.*
       experiments need   the[NEUT,SG,GEN] agreement
       'Experiments must be agreed to.'

(92) **Experimente bedürfen des.*
       experiments needs   the[NEUT,SG,GEN]

(93)   *Experimente bedürfen dessen.*
       experiments needs   this[NEUT,SG,GEN]
       'Experiments need this.'

In older stages of the language, the "short" form *des* 'the[GEN]' could also be used as a pronoun (94). In fact, the definite article emerged historically from the demonstrative pronoun (Drosdowski 1995, p. 332f).

(94)   *Des   freuet sich der Engel Schar. (Martin Luther, 1524)*
       this[NEUT,SG,GEN] rejoices REFL the angels host
       'Over this the host of angels rejoices.'

The same restrictions apply to the indefinite article, compare (95) and the ungrammatical (96) (in contrast to the fully inflected form of the indefinite pronoun, *eines* 'one', as in (97)).

(95)   *Ein   Kind kommt.*
       a[UNINFL] child comes
       'A child is coming.'

(96) **Ein   kommt.*
       a[UNINFL] comes

(97)   *Eines   kommt.*
       one[NEUT,SG,NOM] comes
       'One (child) is coming.'

Note, however, that the paradigms of the definite article *die* 'the' and the demonstrative pronoun *die* 'this' are almost identical. The same holds for the indefinite article *eine* 'a' and the indefinite pronoun *eine* 'one', as shown by the following tables.

The following tables give an overview of the inflection of the definite article vs. demonstrative pronoun; differing forms are highlighted.

CASE	MASC,SG	NEUT,SG	FEM,SG	PL
Nominative	der	das	die	die
Genitive	**des**	**des**	**der**	**der**
Dative	dem	dem	der	**den**
Accusative	den	das	die	die

Table 7.6: The German definite article

CASE	MASC,SG	NEUT,SG	FEM,SG	PL
Nominative	der	das	die	die
Genitive	**dessen**	**dessen**	**deren**	**deren/derer**
Dative	dem	dem	der	**denen**
Accusative	den	das	die	die

Table 7.7: The German demonstrative pronoun

The demonstrative pronoun comes in two variants in the genitive plural, *deren* and *derer* 'these[GEN]'. *deren* is used for backward reference (anaphoric reference) as in (98), whereas *derer* is used for forward reference (cataphoric reference) (99) (Drosdowski 1995, p. 334).

(98)   *ihre Freunde und deren   Kinder*
        her friends and these[GEN] children
        'her friends and their (= her friends') children'

(99)   *die Liste derer,   die* …
        the list these[GEN] who
        'the list of those who …'

This is an overview of the forms of the indefinite article vs. indefinite pronoun. Again, differing forms are highlighted (there is no plural form of the indefinite article and pronoun).

CASE	MASC,SG	NEUT,SG	FEM,SG
Nominative	**ein**	**ein**	eine
Genitive	eines	eines	einer
Dative	einem	einem	einer
Accusative	einen	**ein**	eine

Table 7.8: The German indefinite article

CASE	MASC,SG	NEUT,SG	FEM,SG
Nominative	**einer**	**eines**	eine
Genitive	eines	eines	einer
Dative	einem	einem	einer
Accusative	einen	**eines**	eine

Table 7.9: The German indefinite pronoun

Besides the definite and indefinite articles, there are further determiners that cannot be used as determiner-pronouns, compare the determiner *manch* 'some' in (100) and its ungrammatical use as a determiner-pronoun in (101).

(100)   *Manch   Journalist arbeitet daran.*
         some[UNINFL] journalist works   there_on
         'Some journalist is working on that.'

(101)   **Manch   arbeitet daran.*
         some[UNINFL] works   there_on

The ungrammatical examples, featuring *ein* 'a[UNINFL]' and *manch* 'some[UNINFL]', seem to suggest that determiner-pronouns have to exhibit strong inflection. However, that is not true, compare the uninflected determiner *viel* 'much' (102) and its grammatical use as a determiner-pronoun in (103).

(102)   *Viel   Weizen ging kaputt.*
         much[UNINFL] wheat went broken
         'Much wheat was destroyed.'

(103)   *Viel   ging kaputt.*
         much[UNINFL] went broken
         'Much was destroyed.'

Summary   Most of the determiners in German can be used as determiner-pronouns, i.e. they can represent a full DP on their own, cf. the trees below, representing a determiner in transitive and intransitive use. Some determiners behave differently in that they are obligatorily "transitive", taking an NP as their c-structure complement. Among others, the definite and indefinite articles belong to this class.

7.4.2   "Adjective-Pronouns"

As we have seen above (cf. sec. 7.3), some quantifying expressions do not occupy the D position but rather an adjectival position, e.g. *mehrere* 'some'. Now, just as determiners, quantifying adjectives have pronominal counterparts (104) vs. (105). How can this be explained in a DP analysis?

(104)   *mehrere Häuser*
         some   houses

(105)   *mehrere*
        some

As described in the preceding section, determiners can project a DP on their own (the "determiner-pronouns"). In fact, even an ordinary attributive adjective can represent a full DP on its own. For example, the adjectives *trockene* 'dry' or *bessere* 'better' can, in a suitable context, represent a full DP, cf. (106), (107).

(106)   (see corpus example DP7.5, Appendix A)
        ihre nassen Matratzen gegen trockene austauschen
        their wet   matresses by   [dry DP] replace
        'to replace their wet matresses by dry ones'

(107)   (see corpus example DP7.6, Appendix A)
        Sie wollten sich jetzt nur bessere   erzwingen.
        they wanted REFL now only [better DP] enforce
        'They only wanted to enforce better ones.'

The natural occurrence of adjectives as full DPs implies that a (quantifying) adjective projecting a DP of its own, as, e.g., in (108), is not a special case.

(108)   *Mehrere   kamen.*
        [some DP] came

Trees featuring the different uses of a quantifying adjective (Aquant) are outlined below. (Note, however, that Aquant is not dominated by an NP in our implementation (cf. p. 178)).

```
     DP                    DP
     |                     |
    Dbar                  Dbar
     |                     |
     NP                    NP
    /  \                   |
Aquant  Nbar            Aquant
```
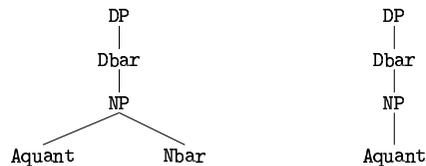
7.4.3 "Real Pronouns"

Besides the "pseudo-pronouns" described in the previous sections, there are also "real" ones: personal pronouns (109) and indefinite pronouns as in (110), (111).

(109)   *er*
        he

(110)   *jemand*
        someone

(111)   *nichts*
        nothing

These never occur in determiner position (D), i.e. they are obligatorily "intransitive", cf. (112) and (113).

(112)   **er Mann*
        he man

(113)   **jemand Mann*
        someone man

The demonstrative pronoun *die* 'this' and the indefinite pronoun *eine* 'one' (see the examples above (cf. p. 160)) also belong to this class.

These "real" pronouns are assigned the category PRON, cf. the tree below.

```
   DP
   |
  Dbar
   |
  PRON
```
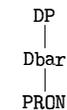
Note that the *jemand*-type pronoun can occur in a putative determiner-like construction, as in (114), (115). In this type of construction, the noun is obligatorily realized by a nominalized adjective.

(114)   *jemand Nettes*
        someone nice

(115)   *nichts Gutes*
        nothing good

However, the nominalized adjective does not represent the head of the construction. Instead, it is an appositive to the pronoun. This can be seen by the fact that case and gender agreement between the pronoun and the nominalized adjective is optional, cf. the gender mismatch in (116) (the relative pronoun indicates the gender of its head), and the case mismatch in (117) (Drosdowski 1995, p. 350).

(116)   *jemand   Nettes,   der*
        someone[MASC] nice[NEUT] who[MASC]

(117)   *von jemand   Fremdes*
        from someone[DAT] foreign[NOM]
        'from some foreigner'

This kind of case and gender disagreement is typical for appositive constructions. In contrast, determiner-noun constructions always enforce case and gender agreement.

A similar construction can be found with personal pronouns (118); this might also be a special type of appositive.

(118)   *wir Männer*
        we men
        'us men'

We now leave the quantificational expressions and come to a different topic: restrictions on DPs in genitive case, occupying the SPEC-DP position.

7.5 Genitive DPs

German nouns (NPs) can be modified by genitive DPs which precede or follow the NP. There is some debate about whether prenominal (GENpre, occupying the SPEC-DP position) and postnominal (GENpost) genitive DPs are different constructions. Proposed differences concern morphological, syntactic, and semantic properties, briefly discussed below.

Genitive ending with *-s* (morphological restriction)   GENpre must be realized by a genitive phrase ending with *-s* (Olsen 1991). That is, *Karls* qualifies as a GENpre (119), *der Maria* 'the[GEN] Maria' (120) or *des Irak* 'the[GEN] Iraq' (121) do not (although the article *des* is overtly marked by *-s*).

(119)   *Karls   Flugzeuge*
        [K. GEN] planes
        'Karl's planes'

(120)   **der Maria   Flugzeuge*
        [the M. GEN] planes

(121)   **des Irak   Flugzeuge*
        [the Iraq GEN] planes

In contrast, GENpost can be realized by any genitive phrase, i.e. *Karls* (122) as well as *der Maria* (123) and *des Irak* can represent GENpost (124).

(122)   *die Flugzeuge Karls*
        the planes   [K. GEN]
        'Karl's planes'

(123)   *die Flugzeuge der Maria*
        the planes   [the M. GEN]
        'Maria's planes'

(124)   *die Flugzeuge des Irak*
        the planes   [the Iraq GEN]
        'Iraq's planes'

Adjacency condition (syntactic restriction)   No constituent may intervene between the genitive phrase and the NP (Olsen 1991). That is, GENpre may not contain any postmodifying constituent. For instance, the PP *aus Berlin* 'from Berlin', that modifies the genitive phrase *meines Onkels* 'my uncle[GEN]', cannot intervene between the genitive phrase and the NP *Wagen* 'car', compare (125) and (126).

(125)   *meines Onkels   Wagen*
        [my   uncle GEN] car
        'my uncle's car'

(126)   **meines Onkels   aus   Berlin   Wagen*
        [my   uncle GEN] [from Berlin PP] car

In contrast, GENpost can be followed by any postmodifying element (127).

(127)   *der Wagen meines Onkels   aus   Berlin*
        the car   [my   uncle GEN] [from Berlin PP]
        'the car of my uncle from Berlin'

The adjacency restrictions could be viewed as an instance of the above morphological restriction, namely that GENpre must be realized by a genitive phrase ending with *-s*. In the ungrammatical example (128), the PP *aus Berlin* 'from Berlin' is actually part of GENpre (being a modifier of *Onkel* 'uncle'). Hence, GENpre does not end with *-s*.

(128)   **meines Onkels aus Berlin   Wagen*
        [my   uncle   from Berlin GENpre] car

Gallmann and Lindauer believe that the adjacency restriction also holds for genitive DPs followed by a postposition: compare the genitive DPs preceding the postposition *wegen* 'because' in (129) and (130) and the genitive DP following the preposition *wegen* 'because' in (131) (Gallmann and Lindauer 1994).

(129) *meines Onkels   wegen*
       [my   uncle GEN] because
       'because of my uncle'

(130) **meines Onkels   aus   Berlin   wegen*
       [my   uncle GEN] [from Berlin PP] because

(131) *wegen meines Onkels   aus   Berlin*
       because [my   uncle GEN] [from Berlin PP]
       'because of my uncle from Berlin'

Corpus data, however, provide counterevidence, cf. the FR examples with postmodifying PPs (132), or postmodifying genitive DPs (133), (134).

(132) (see corpus example DP7.7, Appendix A)
       um der Debatte   in der landwirtschaftlichen Fachwelt
       for [the debate GEN] [in the agricultural   professional_world PP]
       willen
       sake
       'for the sake of the debate among agricultural experts'

(133) (see corpus example DP7.8, Appendix A)
       um der Gesundheit   der 8000 Einwohner   willen
       for [the health GEN] [the 8000 inhabitants GEN] sake
       'for the sake of the health of the 8000 inhabitants'

(134) (see corpus example DP7.9, Appendix A)
       der einseitigen Ausrichtung   der Produktionen   wegen
       [the one-sided orientation GEN] [the productions GEN] because
       'because of the one-sided orientation of the productions'

Possessive relation (semantic restriction)   Olsen assumes a possessive relation between GENpre and the head phrase, i.e. the referent of GENpre in some way possesses the referent of the head phrase (Olsen 1991), exemplified by (135).

(135) *Karls   Flugzeuge*
       [K. GEN] planes
       'Karl's planes'

This semantic restriction excludes examples such as (136).

(136) **des Satzes   Verb*
       [the sentence GEN] verb

In contrast, the relation between GENpost and the head phrase is not semantically restricted (137).

(137) *das Verb des Satzes*
       the verb [the sentence GEN]
       'the sentence's verb'

Similarly, Bhatt notes that typical GENpre constituents are realized by proper nouns and common nouns denoting animate objects (semantic restriction) (Bhatt 1990, p. 113ff).

Feminine nouns ending with -*s* (morphological restriction)   The restrictions mentioned so far only concern GENpre. According to Olsen, a restriction concerning GENpost is the following (Olsen 1991): feminine genitive phrases ending with -*s* cannot project a GENpost, cf. (138), (139) (which Olsen considers ungrammatical; however, judgements vary here).

(138) *die Katze Omas
       the cat   [granny FEM,GEN]

(139) (*)*der Pelzmantel Tante Emmas*
       the fur_coat   [aunt E. FEM,GEN]
       'aunt Emma's fur coat'

In contrast, they can project a GENpre (140), (141).

(140) *Omas   Katze*
       [granny FEM,GEN] cat
       'granny's cat'

(141) *Tante Emmas   Pelzmantel*
       [aunt E. FEM,GEN] fur_coat
       'aunt Emma's fur coat'

Note that feminine nouns usually do not end with -*s*; only feminine proper nouns and titles may do so, optionally.

Summary   GENpre and GENpost seem to be different constructions. In general, GENpre is more restricted than GENpost. However, the data are not always clear.

Chapter 8

Basics of the Implementation

Contents

In this chapter, we present basic properties of our DP implementation. The structure of this chapter parallels the previous chapter, by addressing the following topics.

- DP projections (sec. 8.1)

- Determiner and adjective inflection: strong vs. weak (sec. 8.2)

- Determiners and quantifying adjectives (sec. 8.3)

- Pronouns (sec. 8.4)

- Genitive DPs (sec. 8.5)

In many aspects, our grammar implements the ideas of the theoretical literature presented above. In this chapter, we give a rough overview of our implementation, focussing on the points where we diverge from the theoretical analyses.

8.1   DP Projections

In our implementation, any nominal phrase projects a DP (even if no specifier or determiner is present). Otherwise we would have to introduce disjunctions in the c-structure, allowing for a DP or NP wherever a nominal phrase is introduced in a rule.

To keep the c-structure flat, our analysis does not project a Dbar node but makes use of a macro @Dbar instead (a macro does not correspond to a node in the c-structure), cf. the c-structure analyses of (142) and (143). (Note that our implementation defines an additional projection, DPx, for technical reasons (cf. sec. 10.2).)

(142)   *das Kind*
         the child

```
CS 1:     DP[std]
             |
           DPx[std]
            /    \
         D[std]   NP
           |       |
          das   N[comm]
                   |
                 Kind
```
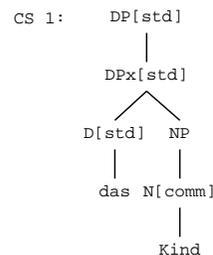
(143)   *Karls Kind*
         Karl's child

```
CS 1:     DP[std]
             |
           DPx[std]
            /    \
        NAMEP     NP
           |       |
         Karls   N[comm]
                   |
                 Kind
```
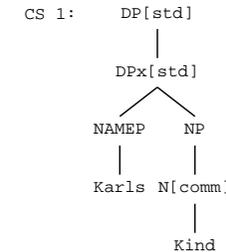
8.2   Determiner and Adjective Inflection

The generalization formulated in the theoretical chapter was: the head D should exhibit the feature "strong"; if there is no determiner or if the determiner is uninflected, attributive adjectives represent this feature (cf. p. 155). (Quantifying adjectives can be subsumed under ordinary attributive adjectives in this respect.)

Our implementation models this generalization by means of a feature INFL, with values assigned by (inflected) determiners and adjectives (more exactly: by the morphological tags marking the inflection type). All INFL features of a DP are unified with the DP's f-structure, hence all values assigned to INFL must unify.

By means of suitable constraints on the feature INFL, our implementation accounts for the restrictions on determiner–adjective co-occurrence. We distinguish three cases.

(i) [INFL strong-det**]**   The feature [INFL strong-det] marks strong determiners, that may be followed by weak adjectives, as in (144).

(144)   *das   kleine   Kind*
         the[ST] small[WK] child

"das kleine Kind"

```
⎡PRED      'Kind'                                          ⎤
⎢         ⎧ ⎡PRED   'klein<[0:Kind]>'            ⎤      ⎫  ⎥
⎢         ⎪ ⎢SUBJ   [0:Kind]                     ⎥      ⎪  ⎥
⎢         ⎪ ⎢       ⎡AMORPH [AHEAD base]    ⎤    ⎥      ⎪  ⎥
⎢ADJUNCT  ⎨ ⎢CHECK  ⎢                       ⎥    ⎥      ⎬  ⎥
⎢         ⎪ ⎢       ⎣MORPH  [CAPITAL –]     ⎦    ⎥      ⎪  ⎥
⎢         ⎪62⎣ADEGREE base–, ATYPE attributive⎦      ⎪  ⎥
⎢         ⎩                                            ⎭  ⎥
⎢         ⎡NCONSTR   [ADJ-ATTR +]                   ⎤    ⎥
⎢CHECK    ⎢                                           ⎥    ⎥
⎢         ⎣SPEC-TYPE [COUNT +, DEF +, DET attr]      ⎦    ⎥
⎢NSEM      [COMMON count]                                 ⎥
⎢NTYPE     [NSYN common]                                  ⎥
⎢SPEC      ⎡DET ⎡PRED     'die'⎤⎤                        ⎥
⎢          ⎣    ⎣DET-TYPE def   ⎦⎦                        ⎥
⎢0 GEND neut, INFL strong-det, NUM sg, PERS 3            ⎥
⎣                                                          ⎦
```

(ii) [INFL strong-adj]   The feature [INFL strong-adj] marks strong adjectives, that may be preceded by uninflected determiners, as in (145).

(145)   *ein*   *kleines*   *Kind*
         a[UNINFL] small[ST] child

"ein kleines Kind"

```
⎡PRED      'Kind'                                          ⎤
⎢         ⎧ ⎡PRED   'klein<[0:Kind]>'            ⎤      ⎫  ⎥
⎢         ⎪ ⎢SUBJ   [0:Kind]                     ⎥      ⎪  ⎥
⎢         ⎪ ⎢       ⎡AMORPH [AHEAD base]    ⎤    ⎥      ⎪  ⎥
⎢ADJUNCT  ⎨ ⎢CHECK  ⎢                       ⎥    ⎥      ⎬  ⎥
⎢         ⎪ ⎢       ⎣MORPH  [CAPITAL –]     ⎦    ⎥      ⎪  ⎥
⎢         ⎪54⎣ADEGREE base–, ATYPE attributive⎦      ⎪  ⎥
⎢         ⎩                                            ⎭  ⎥
⎢         ⎡NCONSTR   [ADJ-ATTR +]           ⎤          ⎥
⎢CHECK    ⎢                                  ⎥          ⎥
⎢         ⎣SPEC-TYPE [COUNT +, DET attr]     ⎦          ⎥
⎢NSEM      [COMMON count]                                 ⎥
⎢NTYPE     [NSYN common]                                  ⎥
⎢SPEC      ⎡DET ⎡PRED     'eine'⎤⎤                       ⎥
⎢          ⎣    ⎣DET-TYPE indef  ⎦⎦                       ⎥
⎢0 GEND neut, INFL strong-adj, NUM sg, PERS 3           ⎥
⎣                                                          ⎦
```

(iii) (↑INFL) =c strong-det   Weak adjectives introduce the constraint (↑INFL) =c strong-det. Hence, they must be preceded by a strong determiner, which supplies this feature.

More details are presented in the next chapter (cf. sec. 9.2.2).

8.3   Determiners and Quantifying Adjectives

In the preceding section, a clear line was drawn between determiners on one side and adjectives (including quantifying adjectives) on the other, based on inflectional properties. However, the borderline is not always that clear. Many quantifying expressions exhibit idiosyncratic inflection.

Traditional grammars note that after certain expressions the inflection type of attributive adjectives varies. For example, quantifying expressions preceding weak adjectives (hence determiners, according to our analysis) comprise: *solche* 'such', *irgendwelche* 'any', and *manche* 'some'. But some of these expressions also do tolerate strong adjectives (e.g. *irgendwelche*); some even prefer strong adjectives but only in plural forms (e.g. *manche*), etc.

Traditional grammars typically devote several sections to the problem of such idiosyncratic inflectional properties. Here is an example.

"[So wie nach dem definiten Artikel], aber mit bestimmten Einschränkungen werden die Adjektive flektiert nach den Artikelwörtern *mancher* (Plural überwiegend wie nach Nullartikel [= wie ohne Determiner]), *irgendwelcher* (durchgehend auch wie nach Nullartikel möglich), *solcher* (gelegentlich wie nach Nullartikel, nicht aber im Sg. Nom. und Akk. aller Genera und Gen. Mask. und Neutr.), *welcher* und *aller* (selten auch wie nach Nullartikel)." (Helbig and Buscha 1993, p. 301, [...] added by S.D.)

"After the following quantifying expressions, adjectives show weak inflection (with certain restrictions, listed in parentheses): *manche* 'some' (in plural predominantly strong), *irgendwelche* 'any' (strong inflection equally possible), *solche* 'such' (sometimes strong, but not in [SG,NOM/ACC] and [MASC/NEUT,GEN]), *welche* 'which' and *alle* 'all' (rarely strong)." (free translation)

Ambiguous quantifying expressions   An explanation for this phenomenon could be that such quantifying expressions are (more or less) ambiguous: they can be determiners as well as quantifying adjectives. However, this "explanation" does not explain why certain expressions prefer being determiners, others only tolerate it, and a third class tolerates it but only in specific forms, etc. Phonetic properties may play a role here.

We already mentioned instances of ambiguous determiner-adjectives: *beide* 'both' and *eine* 'a/one'. In these cases, the presence or absence of the definite article helped in distinguishing both readings (cf. p. 159). Most of the quantifying expressions dealt with in this section, though, are not compatible with the definite article: inflection is the only piece of evidence we have.

In our implementation, we capture the generalization in the following way. In general, quantifying expressions can be determiners as well as quantifying adjectives (with the exception of clear cases such as the definite and indefinite article). If in a given sentence, the quantifying expression precedes an attributive adjective, the inflection of that adjective (strong/weak) determines

(i.e. disambiguates) whether the quantifying expression is a determiner (D) or quantifying adjective (Aquant). If the quantifying expression does not precede such a disambiguating attributive adjective, the determiner reading is, somewhat arbitrarily, preferred via an OT mark.

Identical f-structure   Despite the variance in inflection, quantifying expressions are uniformly represented in f-structure. That is, a quantifying expression in the Aquant position—although inflecting like an ordinary adjective—does not function as a modifier (contrary to adjectives); instead, it functions as a specifier. Hence, in our analysis, the c-structure distinction D vs. Aquant does not correspond to an f-structure distinction.

Note that, to allow for multiple quantifying expressions, a simple feature SPEC does not suffice. Instead, we make use of complex features such as SPEC DET, SPEC POSS, SPEC QUANT, etc., that are projected by the different types of quantifying expressions (articles, possessives, indefinites, respectively). For instance, in the lexicon entry, the indefinite *irgendwelche* 'any' introduces the constraint (↑SPEC QUANT) = ↓.

Below we illustrate the analysis of the quantifying expression *irgendwelche* 'any' in its use as a determiner (D) (146) or quantifying adjective (Aquant) (147). Note that the f-structures of both examples are identical except for the value of the feature INFL. The quantifying expression is represented as a quantificational specifier (SPEC QUANT).

(146)   *irgendwelche kleinen   Kinder*
         some[ST]   small[WK] children
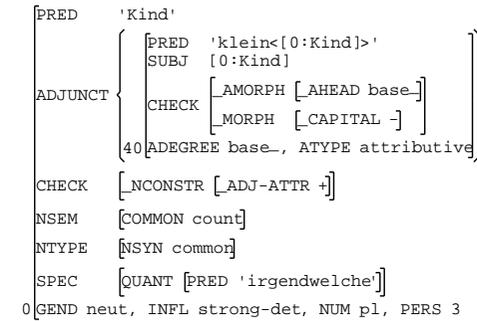
(147)   *irgendwelche kleine   Kinder*
         some[ST]   small[ST] children

```
"irgendwelche kleine Kinder"
```

$$
\begin{bmatrix}
\text{PRED} & \text{'Kind'} \\
\text{ADJUNCT} & \left\{ \begin{bmatrix} \text{PRED} & \text{'klein<[0:Kind]>'} \\ \text{SUBJ} & \text{[0:Kind]} \\ \text{CHECK} & \begin{bmatrix} \text{AMORPH} & [\text{AHEAD base}] \\ \text{MORPH} & [\text{CAPITAL -}] \end{bmatrix} \\ 40 \text{ADEGREE base}, \text{ATYPE attributive} \end{bmatrix} \right\} \\
\text{CHECK} & [\text{NCONSTR} [\text{ADJ-ATTR +}]] \\
\text{NSEM} & [\text{COMMON count}] \\
\text{NTYPE} & [\text{NSYN common}] \\
\text{SPEC} & [\text{QUANT} [\text{PRED 'irgendwelche'}]] \\
0 & \text{GEND neut, INFL strong-adj, NUM pl, PERS 3}
\end{bmatrix}
$$

Unambiguous quantifying expressions In our analysis, quantifying expressions in general can be determiners as well as quantifying adjectives. Some of the quantifying expressions, though, show a clear preference for D or Aquant. For instance, *jene* 'that' almost exclusively inflects like a determiner, whereas *mehrere* 'some' almost always inflects like an adjective. These quantifying expressions are restricted to the categories D or Aquant, respectively, via their lexical stem categories.

Which of the quantifying expressions are restricted in such a way is decided on the basis of corpus data: If a quantifying expression inflects like a determiner in 98% or more of the (unambiguous) corpus instances, then the quantifying expression is always analyzed as a D. If it inflects like an adjective in 98% or more of the instances, it is always analyzed as an Aquant. (Note that most of the corpus instances are ambiguous, either because the inflection of the adjective is ambiguous (can be strong or weak) or because no adjective follows.)

C-structure position of Aquant The position Aquant is not dominated by NP but by DP. There are two reasons for this: (i) quantificational adjectives always precede all other adjectives (148). This is directly modeled by putting Aquant in the higher DP projection.

(148)   *kleine mehrere Kinder*
         small some   children

(ii) More importantly, quantificational adjectives can be interrogative: (149).

(149)   *wieviele   deutsche   Aussiedler*
         how_many[ST] German[ST] emigrants

Treating *wieviele* as a quantifying adjective within NP, we would need to parametrize the NP rule, since we encode interrogativity in c-structure (cf. sec. 10.1). Moreover, this would be in contrast to the generalization we otherwise observe: that the type of a DP is determined by elements of the D projection, never by some element within NP.

Note that cardinals and ordinals may follow attributive adjectives (150), (151), hence they are dominated by NP rather than DP (despite being represented as specifiers (SPEC)).

(150)   *die übrigen   acht Mannschaften*
         the remaining eight teams

(151)   *bis zum   vollendeten 25. Lebensjahr*
         up to_the finished   25th life_year
         'before attaining the age of 25'

A detailed analysis of D and Aquant is presented below (cf. sec. 10.5).

8.4   Pronouns

In our implementation, we allow for three types of pronouns, "determiner-pronouns", "adjective-pronouns", and "real pronouns" (cf. sec. 7.4).

"Determiner/adjective-pronouns" "Determiner-pronouns" are DPs projected by a single determiner (152), "adjective-pronouns" are DPs projected by a single quantifying adjective (153).

(152)   *dieses*
         this

```
CS 1:    DP[std]
            |
         DPx[std]
            |
          D[std]
            |
          dieses
```

```
"dieses"
```

$$
\begin{bmatrix}
\text{PRED 'pro'} \\
0 \text{SPEC} [\text{DEM} [\text{PRED 'diese'}]]
\end{bmatrix}
$$

(153)   *mehrere*
         some

```
        CS 1:     DP[std]
                     |
                  DPx[std]
                     |
                 Aquant[std]
                     |
                  mehrere
```

```
     "mehrere"
        ┌                          ┐
        │PRED 'pro'                │
       0│SPEC [QUANT [PRED 'mehrere']]│
        └                          ┘
```
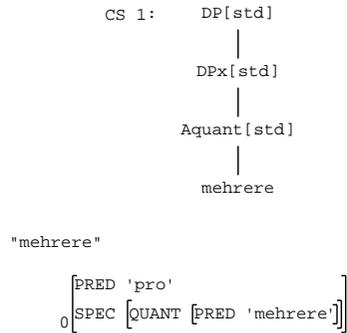
In both cases, the head noun is omitted and thus cannot provide the value of the DP's PRED feature. Instead, the feature [PRED 'pro'] is introduced to indicate that the referent of the DP is to be constructed from the context. The determiner or quantifying adjective functions as the specifier (SPEC).

"Real pronouns"   These are DPs dominating a pronoun of category PRON (154).

(154)   *jemand*
         someone

```
        CS 1:     DP[std]
                     |
                  DPx[std]
                     |
                 PRON[std]
                     |
                  jemand
```

```
   "jemand"
     ┌                                                        ┐
     │PRED  'pro'                                             │
     │CHECK [LEX [PRON [APP +]]]                              │
     │                                                        │
     │NTYPE [NSYN pronoun]                                    │
    0│GEND masc, NUM sg, PERS 3, PRON-FORM jemand, PRON-TYPE indef│
     └                                                        ┘
```
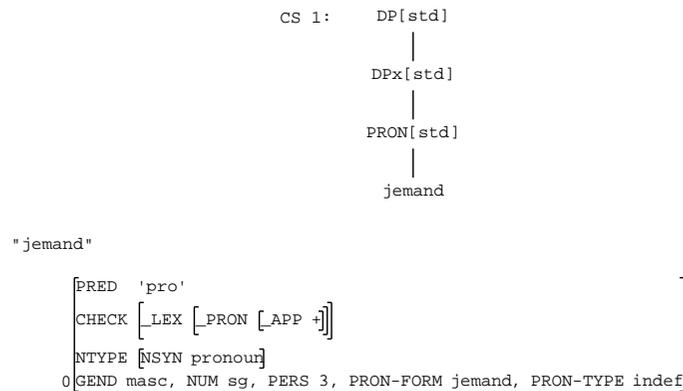
These pronouns cannot function as specifiers. Similar to the other types of pronouns, they introduce the constraint (↑PRED) = 'pro'. The actual form of the pronoun is recorded in the feature PRON-FORM, its type in the feature PRON-TYPE (Butt *et al.* 1999, p. 76).

8.5   Genitive DPs

In NEGRA, 72% of the instances of the prenominal genitives are realized by proper nouns, 23% are realized by pronouns. The only instances of ordinary, full DPs in SPEC-DP are idiomatic-like expressions. Therefore we restrict prenominal genitive DPs to pronouns (155) or names (156). Prenominal genitive DPs function as possessives (SPEC POSS).

(155)   *dessen   Frau*
         this[GEN]   wife
         'his wife'

```
        CS 1:     DP[std]
                     |
                  DPx[std]
                  /       \
          PRONspec[std]   NP
                |          |
             dessen     N[comm]
                           |
                         Frau
```

```
     "dessen Frau"
        ┌                          ┐
        │PRED 'Frau'               │
      58│SPEC [POSS 0[PRED 'pro']]  │
        └                          ┘
```
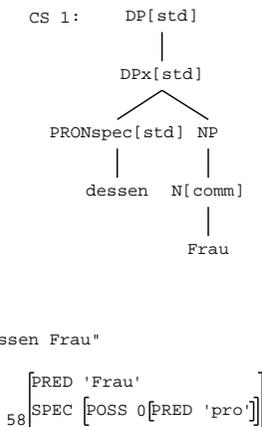
(156)   *Karl Maiers   Frau*
         [K. M. GEN] wife
         'Karl Maier's wife'

```
CS 1:        DP[std]
                |
             DPx[std]
              /     \
          NAMEP      NP
          /   \       |
       NAME   NAME  N[comm]
        |      |      |
       Karl  Maiers  Frau
```

```
"Karl Maiers Frau"
```

$$\begin{bmatrix} \text{PRED} & \text{'Frau'} \\ \text{SPEC} & \begin{bmatrix} \text{POSS} & \begin{bmatrix} \text{PRED} & \text{'Maier'} \\ \text{NAME-MOD} & \left\{ \begin{bmatrix} \text{PRED 'Karl'} \end{bmatrix} \right\} \end{bmatrix}_{17} \end{bmatrix} \end{bmatrix}_{38}$$
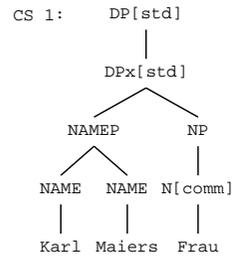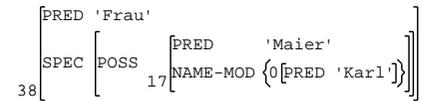
In contrast, postnominal genitives can be realized by various kinds of genitive DPs. They function as a special type of adjunct, cf. (157). (We use the special function ADJ-GEN rather than the general function ADJUNCT here for efficiency reasons.)

(157)   *das Auto Karls*
       the car   [K. ADJ-GEN]
       'Karl's car'

```
"das Auto Karls"
```

$$\begin{bmatrix} \text{PRED} & \text{'Auto'} \\ \text{ADJ-GEN} & \begin{bmatrix} \text{PRED 'Karl'} \end{bmatrix}_{78} \\ \text{SPEC} & \begin{bmatrix} \text{DET} & \begin{bmatrix} \text{PRED 'die'} \end{bmatrix} \end{bmatrix} \end{bmatrix}_{0}$$
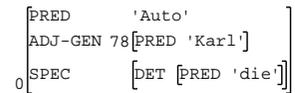
Chapter 9

F-Structure Implementation

Contents

In this chapter, we present all the features that play a role in the DP-internal syntax in our implementation. The following chapter describes how these features are projected from the c-structure. (Functions such as SUBJ and OBJ—which may be realized by DPs—are not addressed here, because the relevant properties of these functions are determined by the subcategorizing element, e.g. the verb, rather than the DP itself.)

We discuss the following features.

- SPEC features (sec. 9.1)

- Agreement features (sec. 9.2)

- CHECK features (sec. 9.3)

- COMPLETE features (sec. 9.4)

9.1 SPEC **Features**

Determiners (or quantifying adjectives) and NPs are co-heads, i.e. they are both annotated by the trivial equation ↑=↓. The determiner introduces a feature SPEC, which normally (i.e., in the LFG tradition) is an atomic feature with the values def and indef or the and a.

In the context of the Pargram project, however, SPEC is used as a complex feature with sub-classifying features for different types of specifiers or quantifying expressions. Depending on the type of the quantifying expression, it ends up under different f-structures (cf. sec. 10.7.1). Our grammar defines the following subtypes of specifiers:

- Articles: SPEC DET

- Possessives: SPEC POSS

- Demonstratives: SPEC DEM

- Indefinites: SPEC QUANT

- Interrogatives: SPEC INT

- Cardinals: SPEC NUMBER

As an example, see the f-structure analysis of (158).

(158)   *eine Frau*
         a   woman

```
"eine Frau"
```

$$0\begin{bmatrix} \text{PRED 'Frau'} \\ \text{SPEC } \begin{bmatrix} \text{DET } \begin{bmatrix} \text{PRED 'eine'} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

The f-structure feature SPEC QUANT is a Pargram feature; in our analysis, SPEC INDEF might have been more appropriate. In contrast to Pargram, we do not use a feature SPEC DET-FORM to record the lexical form of the article. Instead, the feature SPEC DET embeds a PRED feature just as the corresponding features SPEC POSS, SPEC DEM, etc. do. The advantage is that we can treat all quantifying expressions the same way.

Genitive DPs that occupy the specifier position of another DP (SPEC-DP) also function as possessors (SPEC POSS), cf. (159).

(159)   *Karls   Frau*
         [K. SPEC-DP] wife
         'Karl's wife'

```
"Karls Frau"
```

$$18\begin{bmatrix} \text{PRED 'Frau'} \\ \text{SPEC } \begin{bmatrix} \text{POSS 0} \begin{bmatrix} \text{PRED 'Karl'} \end{bmatrix} \end{bmatrix} \end{bmatrix}$$

9.2 **Agreement Features**

Our grammar defines the following agreement features (i.e. features that are involved in some DP-internal agreement relation): GEND (gender), NUM (number), CASE, and INFL (inflection type). Constituents that are involved in such agreement relations are: specifiers D and Aquant, attributive adjectives, and the semantic head N.

All agreement features of a DP are unified with the DP's f-structure. For instance, the specifier *ein* 'a[NEUT,SG,NOM]' projects the following partial f-structure, which is unified with the f-structure of the head noun.

$$\begin{bmatrix} \text{SPEC} & \begin{bmatrix} \text{DET} & \begin{bmatrix} \text{PRED 'eine'} \end{bmatrix} \end{bmatrix} \\ \text{GEND} & \text{neut} \\ \text{NUM} & \text{sg} \\ \text{CASE} & \text{nom} \end{bmatrix}$$

Adjectives introduce equations such as (↑SUBJ GEND) = neut. Since the adjective's SUBJ is identified with the head noun, the restriction on the feature GEND is unified

with the feature GEND projected by the head noun. That is, all morpho-syntactic features of a DP—be they realized on the determiner, adjective, or head noun—are unified with the DP's f-structure.

9.2.1   GEND, NUM, **and** CASE

Specifier(s), attributive adjectives, and the head noun agree with respect to the features GEND, NUM, and CASE (160). (German nouns are inherently/lexically marked for gender.)

(160)   *ein   großes   Haus*
         a[NEUT,SG,NOM] big[NEUT,SG,NOM] house[NEUT,SG,NOM]

```
"ein großes Haus"

      ⎡PRED    'Haus'                                      ⎤
      ⎢        ⎡ PRED   'groß<[0:Haus]>'           ⎤        ⎥
      ⎢        ⎢ SUBJ   [0:Haus]                   ⎥        ⎥
      ⎢ADJUNCT ⎨        ⎡AMORPH [AHEAD base]⎤      ⎬        ⎥
      ⎢        ⎢ CHECK  ⎢MORPH  [CAPITAL -]⎥      ⎥        ⎥
      ⎢        ⎣54 ADEGREE base_, ATYPE attributive⎦        ⎥
      ⎢        ⎡ NCONSTR   [ADJ-ATTR +]           ⎤        ⎥
      ⎢ CHECK  ⎢                                  ⎥        ⎥
      ⎢        ⎣ SPEC-TYPE [COUNT +, _DET attr]   ⎦        ⎥
      ⎢ NSEM   [COMMON count]                              ⎥
      ⎢ NTYPE  [NSYN common]                               ⎥
      ⎢ SPEC   ⎡DET ⎡PRED     'eine'⎤ ⎤                    ⎥
      ⎢        ⎣    ⎣DET-TYPE indef ⎦ ⎦                    ⎥
      0⎣GEND neut, INFL strong-adj, NUM sg, PERS 3         ⎦
```

9.2.2   INFL

Specifier(s) and attributive adjectives agree in the feature INFL for inflection type (161). (That is, in contrast to the above agreement features, the head noun is not involved. Except for nouns derived from adjectives, nouns are not marked by an inflection type.)

(161)   *ein   großes Haus*
         a[UNINFL] big[ST] house

```
"ein großes Haus"

      ⎡PRED    'Haus'                                      ⎤
      ⎢        ⎡ PRED   'groß<[0:Haus]>'           ⎤        ⎥
      ⎢        ⎢ SUBJ   [0:Haus]                   ⎥        ⎥
      ⎢ADJUNCT ⎨        ⎡AMORPH [AHEAD base]⎤      ⎬        ⎥
      ⎢        ⎢ CHECK  ⎢MORPH  [CAPITAL -]⎥      ⎥        ⎥
      ⎢        ⎣54 ADEGREE base_, ATYPE attributive⎦        ⎥
      ⎢        ⎡ NCONSTR   [ADJ-ATTR +]           ⎤        ⎥
      ⎢ CHECK  ⎢                                  ⎥        ⎥
      ⎢        ⎣ SPEC-TYPE [COUNT +, _DET attr]   ⎦        ⎥
      ⎢ NSEM   [COMMON count]                              ⎥
      ⎢ NTYPE  [NSYN common]                               ⎥
      ⎢ SPEC   ⎡DET ⎡PRED     'eine'⎤ ⎤                    ⎥
      ⎢        ⎣    ⎣DET-TYPE indef ⎦ ⎦                    ⎥
      0⎣GEND neut, INFL strong-adj, NUM sg, PERS 3         ⎦
```

Note that if a specifier and adjective agree with respect to the feature INFL, it means that they (typically) show complementary morphological markings for the inflection type (cf. p. 155).

All INFL features of a DP are unified with the DP's f-structure, hence all values assigned to INFL must unify. Admissible values of INFL are strong-adj and strong-det. These values determine possible co-occurrences of determiners, quantifying adjectives, and ordinary adjectives, as sketched in the following.

Strong determiners and adjectives   Strong adjectives introduce the equation (↑INFL) = strong-adj, and strong determiners introduce the equation (↑INFL) = strong-det. Given that all INFL features must unify, it follows that strong determiners and strong adjectives cannot be present simultaneously (162). The f-structure below illustrates this feature clash.

(162)   **das   kleines   Kind*
         the[ST] small[ST] child

$$
\begin{bmatrix}
\text{PRED} & \text{'Kind'} \\
\text{ADJUNCT} & \left\{ \begin{bmatrix} \text{PRED} & \text{'klein<[0:Kind]>'} \\ \text{SUBJ} & \text{[0:Kind]} \\ \text{CHECK} & \begin{bmatrix} \_\text{AMORPH} & [\_\text{AHEAD base}] \\ \_\text{MORPH} & [\_\text{CAPITAL} -] \end{bmatrix} \\ 73\,\text{ADEGREE base\_, ATYPE attributive} \end{bmatrix} \right\} \\
\text{CHECK} & \begin{bmatrix} \_\text{NCONSTR} & [\_\text{ADJ-ATTR} +] \\ \_\text{SPEC-TYPE} & [\_\text{COUNT} +, \_\text{DEF} +, \_\text{DET attr}] \end{bmatrix} \\
\text{INFL} & = \begin{bmatrix} \text{strong-adj} \\ \text{strong-det} \end{bmatrix} \\
\text{NSEM} & [\text{COMMON count}] \\
\text{NTYPE} & [\text{NSYN common}] \\
\text{SPEC} & \begin{bmatrix} \text{DET} & \begin{bmatrix} \text{PRED} & \text{'die'} \\ \text{DET-TYPE def} \end{bmatrix} \end{bmatrix} \\
0\,\text{GEND neut, NUM sg, PERS 3}
\end{bmatrix}
$$

However, strong determiners may occur without an adjective (163). And, conversely, strong adjectives may occur without a determiner (164) or with uninflected/invariant determiners (165) (since uninflected determiners do not introduce a constraint on INFL).

(163)   *das   Kind*
        the[ST] child

(164)   *mit schönem Farbton*
        with nice[ST] shade

(165)   *viel   unnützes   Gerede*
        much[UNINFL] superfluous[ST] rumour

Sequences of strong adjectives specify the same value for the INFL feature, hence they may cooccur (166).

(166)   *mit schönem blauem   Farbton*
        with nice[ST] blue[ST] shade

Weak adjectives   Weak adjectives (including Aquant) specify a constraining equation (↑INFL) =c strong-det. Therefore, they can only occur with a strong determiner (which provides the required feature), as in (167).

(167)   *welche   beiden   Leute*
        which[ST] both[WK] people
        'which two people'

The partial f-structure and the constraint on the feature INFL that is introduced by *beiden* 'both[WK]' is displayed below (the constraint on the value of INFL is indicated by '[=(c strong-det)]').

$$
\begin{bmatrix}
\text{INFL} & [=(c \text{ strong-det})] \\
\text{NUM} & \text{pl} \\
33\,\text{SPEC} & [\text{NUMBER} [\text{PRED 'beide'}]]
\end{bmatrix}
$$

Sequences of weak adjectives introduce the same constraining equation, hence they can cooccur as well (168).

(168)   *der   schöne   blaue   Farbton*
        the[ST] nice[WK] blue[WK] shade

This analysis correctly excludes examples featuring weak adjectives without any determiner, (169) and (170), as well as examples with uninflected/invariant determiners (171) and (172).

(169)   **warme   Öl*
        warm[WK] oil

(170)   **viele   Gold*
        much[WK] gold

(171)   **ein   kleine   Kind*
        a[UNINFL] small[WK] child

(172)   **viel   unnütze   Gerede*
        much[UNINFL] superfluous[WK] rumour

Uninflected or invariant determiners and adjectives   Uninflected or invariant determiners and invariant adjectives do not specify any constraint on the feature INFL. Hence they are compatible with any value of the feature INFL.

For instance, uninflected adjectives can be construed with strong (173), uninflected (174) or invariant determiners (175).

(173)   *das   lila   Kleid*
        the[ST] purple[UNINFL] dress

(174)   *ein   lila   Kleid*
        a[UNINFL] purple[UNINFL] dress

(175)   *viel   lila   Farbe*
         much[UNINFL] purple[UNINFL] colour

Uninflected adjectives can also be construed with strong (176), weak (177) or invariant adjectives (178).

(176)   *mit schönem lila   Farbton*
         with nice[ST] purple[UNINFL] shade

(177)   *der   schöne   lila   Farbton*
         the[ST] nice[WK] purple[UNINFL] shade

(178)   *der   prima   lila   Farbton*
         the[ST] great[UNINFL] purple[UNINFL] shade

(An uninflected determiner, however, cannot satisfy the constraint introduced by a weak adjective; in this case, another, strong determiner must be present.)

Multiple determiners   The analysis also accounts for multiple determiners. Multiple inflected (= strong) determiners all introduce identical equations, namely (↑INFL) = strong-det, as exemplified by (179).

(179)   *alle   diese   Fragen*
         all[ST] these[ST] questions

```
      "alle diese Fragen"

      ⎡PRED   'Frage'                                    ⎤
      ⎢CHECK  [_SPEC-TYPE [_COUNT +, _DEF +]]            ⎥
      ⎢                                                  ⎥
      ⎢NTYPE  [NSYN common]                              ⎥
      ⎢       ⎡DEM    [PRED 'diese']⎤                    ⎥
      ⎢SPEC   ⎢                     ⎥                    ⎥
      ⎢       ⎣QUANT  [PRED 'alle'] ⎦                    ⎥
      0⎣GEND fem, INFL strong-det, NUM pl, PERS 3        ⎦
```

Uninflected or invariant determiners do not introduce any constraint with regard to the INFL feature, cf. (180) and (181).

(180)   *ein   mancher Freund*
         a[UNINFL] some[ST] friend
         'some friend'

(181)   *manch   einem Freund*
         some[UNINFL] a[ST] friend
         'some friend'

Note that the f-structure constraints on INFL allow for certain ungrammatical co-occurrences of multiple determiners. For instance, if the determiners in the above examples are switched, the result is ungrammatical, cf. (182) and (183).

(182)   **mancher ein   Freund*
          some[ST] a[UNINFL] friend

(183)   **einem manch   Freund*
          a[ST] some[UNINFL] friend

To exclude such examples, we make use of constraints that apply at c-structure (cf. sec. 10.9).

9.3   CHECK **Features**

Our grammar defines three types of CHECK features that are relevant in the DP-internal syntax: (i) CHECK _SPEC-TYPE, which is related to specifiers, (ii) CHECK _LEX _PRON, related to pronouns, and (iii) CHECK _NMORPH _GENITIVE, related to genitive DPs.

(i) The feature CHECK _SPEC-TYPE serves three purposes.

- The features CHECK _SPEC-TYPE _COUNT regulates the usage of specifiers with singular count nouns (cf. sec. 10.11.1).

- The feature CHECK _SPEC-TYPE _DEF is set by specifiers marking definiteness (i.e. by the definite and demonstrative article, and by possessives in the SPEC-DP position). It is used by different constructions (e.g. the label construction) to check whether the DP is definite.

- The feature [CHECK _SPEC-TYPE _DET attr] marks determiners that are "obligatorily transitive" (cf. p. 160). That is, it prohibits the omission of the head noun after these quantifying expressions (cf. sec. 10.10).

(ii) The feature CHECK _LEX _PRON _APP is introduced by certain pronouns that allow for a special type of appositives (cf. sec. 10.2.4).

(iii) The feature CHECK _NMORPH _GENITIVE, with values + and apostrophe, effects the correct morphological form of a DP in genitive case (cf. sec. 10.3.3).

9.4   COMPLETE **Features**

Many of the DP features can be checked as being "complete" at the DP level,
i.e., they can be defined as features whose constraints have to be satisfied locally
(within the DP). Constraining equations (=c) or existential constraints on such
features can be computed more efficiently when these features are defined as
"complete", by calling the template @COMPLETE (cf. p. 71).

The template @DP-COMPLETE is called by the DP rule. Hence, constraints
on the features that represent the argument of the template @COMPLETE are
to be satisfied within the DP.

```
DP-COMPLETE =
        ...
          "(i) determiner features"
        @(COMPLETE (^SPEC INT))
        @(COMPLETE (^SPEC POSS))
        @(COMPLETE (^SPEC QUANT))
        @(COMPLETE (^SPEC AQUANT))
        @(COMPLETE (^SPEC NUMBER))

        @(COMPLETE (^CHECK _NMORPH _GENITIVE))
        ...
```

The following DP features cannot be checked as being complete since they
may be set from outside the DP, namely from a contracted preposition and de-
terminer, such as *zum* 'to the': SPEC DET, CHECK _SPEC-TYPE _COUNT, CHECK
_SPEC-TYPE _DEF, CHECK _SPEC-TYPE _DET, and INFL.

```
DP-COMPLETE =
        ...
          "might be introduced by contracted dets, e.g. 'zum':
        @(COMPLETE (^SPEC DET))
        @(COMPLETE (^CHECK _SPEC-TYPE _COUNT))
        @(COMPLETE (^CHECK _SPEC-TYPE _DEF))
        @(COMPLETE (^CHECK _SPEC-TYPE _DET))
        @(COMPLETE (^INFL))"
        ...
```

As an example, see the f-structure embedded under the function OBJ of the
partial (incomplete) f-structure that is projected by *zum* 'to the'. The f-structure
embedded under the function OBJ will be unified with the f-structure of the
complement of the (contracted) preposition, e.g. a DP.

```
"zum"
⎡PRED   'zu<[2]>'                                              ⎤
⎢       ⎡CHECK   [_SPEC-TYPE [_COUNT +, _DEF +, _DET attr]]  ⎤ ⎥
⎢       ⎢                                                    ⎥ ⎥
⎢OBJ    ⎢SPEC   [DET [PRED      'die']]                      ⎥ ⎥
⎢       ⎢           [    DET-TYPE def ]                      ⎥ ⎥
⎢       ⎢CASE dat, INFL strong-det, NUM sg                   ⎥ ⎥
⎢     2 ⎣HAS_PRED [c T]                                      ⎦ ⎥
⎢                                                              ⎥
⎢CHECK  [_MORPH [_SYNTH zum]]                                  ⎥
⎣  11 PSEM  dir                                                ⎦
```

The features that are checked for being "complete" partly arise from the NP
level, e.g. the feature NTYPE. However, pronouns do not project an NP category
but introduce the feature NTYPE nevertheless. Therefore, it is easier to check
these features at the DP level only.

```
DP-COMPLETE =
        ...
          "(ii) pronoun features"
        @(COMPLETE (^PRON-TYPE))
        @(COMPLETE (^PRON-FORM))
        @(COMPLETE (^CHECK _MORPH _CAPITAL))
        @(COMPLETE (^CHECK _LEX _PRON _APP))

        "(iii) noun features"
        @(COMPLETE (^NTYPE))
        @(COMPLETE (^CHECK _MORPH _DERIV _ADJ)) "nominalized adjs"
        @(COMPLETE (^CHECK _NCONSTR _MEASURE))
        @(COMPLETE (^CHECK _NCONSTR _ADJ-ATTR)) "attributive adjs"
        @(COMPLETE (^MOD))

        "partly assigned in rules, e.g.
        - NSEM COMMON = mass in N-APPOS-MEASURE
        - NSEM PROPER = last_name in NAME-APPOSITIVE
        @(COMPLETE (^NSEM COMMON))
        @(COMPLETE (^NSEM PROPER))"
        ...
```

In the next chapter, we describe how the features presented in this chapter
are projected from the c-structure.

Chapter 10

C-Structure Implementation

Contents

This chapter addresses core aspects of the DP's c-structure. The core structure comprises all constituents that belong to the DP projection, i.e. the maximal projection DP, the specifier position SPEC-DP, and the head. The focus of this chapter is primarily on the analysis of the head of DP, which is realized by quantifying expressions, of category D and Aquant.

Peripherial constructions, such as postnominal modifiers (which are attached to the category DP in our implementation), are not discussed, due to limited space. Similarly, the analysis of pronouns (of category PRON) is only outlined briefly.

10.1   DP Parameters: _type = std/int/rel

The category DP is a complex category in our implementation, for efficiency reasons (cf. sec. 3.3.3.3). We distinguish three main instances of DPs, with parameter values _type = std/int/rel. (CPs, PPs, and ADVPs are parametrized in the same way.)

The type of the DP is determined either by the head D (which is introduced by Dbar) or by the specifier in the SPEC-DP position. A DP may also dominate a pronoun, projecting its type. The basic structure of our DP rule therefore looks as follows.

```
DP[_type] → {  ( SPEC-DP[_type]: (↑SPEC POSS) = ↓; )
                Dbar[_type]: ↑=↓;
             |  PRON[_type]: ↑=↓ }.

Dbar[_type] → ( D[_type]: ↑=↓; )
                NP: ↑=↓.
```

The following data show instantiations of the different types of DP. Note, however, that SPEC-DP and Dbar are macros rather than rules in our implementation. Hence, there are no overt nodes labelled SPEC-DP or Dbar.

DP[std]   (184) is an example of a standard DP featuring a standard category (NAMEP) in the SPEC-DP position. (185) exemplifies a standard determiner, and (186) a standard pronoun.

(184)   *Karls Frau*
         K.   woman
         'Karl's wife'

```
CS 1:      DP[std]
              |
           DPx[std]
            /      \
         NAMEP      NP
           |         |
         Karls    N[comm]
                     |
                   Frau
```

(185)   *eine Frau*
         a   woman

```
CS 1:      DP[std]
              |
           DPx[std]
            /      \
         D[std]     NP
           |         |
         eine     N[comm]
                     |
                   Frau
```

(186)   *jemand*
         someone

```
CS 1:      DP[std]
              |
           DPx[std]
              |
           PRON[std]
              |
            jemand
```
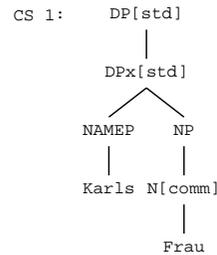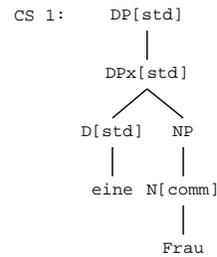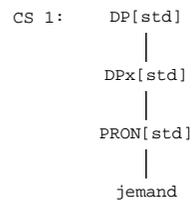
DP[int] and DP[rel]   An example of an interrogative DP is (187). For a
relative DP, cf. (188) (a relative pronoun realizes SPEC-DP).

(187)   *welcher Mann*
         which   man

```
CS 1:      DP[int]
              |
           DPx[int]
            /      \
         D[int]     NP
           |         |
        welcher   N[comm]
                     |
                   Mann
```

(188)   *dessen Frau*
         whose wife

```
CS 1:       DP[rel]
               |
            DPx[rel]
             /      \
      PRONspec[rel]  NP
            |         |
         dessen    N[comm]
                      |
                    Frau
```
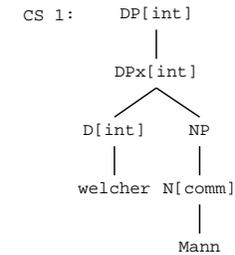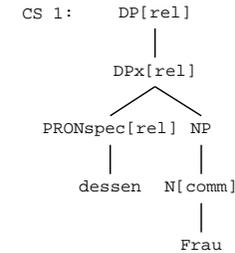
10.2   DP

We now turn to the DP core structure, i.e. the DP level, specifiers, and deter-
miners.

Note first that the "actual" DP is called DPx in our implementation. The
uppermost, additional projection, called DP, answers two purposes. (i) It de-
fines the region of DP-internal features via the template @DP-COMPLETE (cf.
sec. 9.4). (ii) Certain adverbs (of category ADVdp) and postnominal modifiers
(covered by the macro @DPpost) are added at this level in our implementa-
tion. Postnominal modifiers include genitive DPs, PPs, appositives, relative and
argument clauses.

```
DP[_type $ {std int rel}] -->
       ( e: _type = std;
        ADVdp: @(ADJUNCT) )

       DPx[_type]: @(DP-COMPLETE);
```

```
(  "optional constituents may follow DP"
   @(DPpost) ) .
```

Due to space limitations, we restrict the further presentation to the "actual" DP, called DPx. However, we use the simpler term "DP" to refer to the DPx projection.

In our implementation, the German DP projection comes in four shapes: (i) the DP only dominates an NP (sec. 10.2.1); (ii) the DP dominates a constituent in SPEC-DP and an NP (sec. 10.2.2); (iii) the DP dominates D and an NP (sec. 10.2.3); (iv) the DP dominates PRON (sec. 10.2.4).

That is, there is no Dbar projection dominating the NP. There are two reasons for skipping the Dbar projection. First, SPEC-DP and D projections do not cooccur in German, hence they can be ruled out efficiently in c-structure. Second, it keeps the c-structure flat.

10.2.1   DP → NP

In the German DP, neither the SPEC-DP nor the D position is obligatory, cf. (189).

(189)   *Frauen*
         women

```
CS 1:    DP[std]
            |
         DPx[std]
            |
           NP
            |
         N[comm]
            |
         Frauen
```
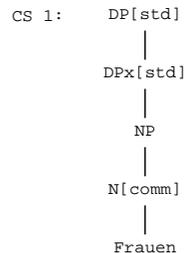
This is only possible within standard DPs (_type = std). The NP is introduced as the single daughter of the DP.

```
DPx[_type] -->
        ...
    | "no SPEC-DP, no D head"
       NP: _type = std;
    | ...
```

10.2.2   DP → SPEC-DP NP

The specifier position, SPEC-DP, may only be filled by a genitive DP (covered by the macro @SPEC-DP (cf. sec. 10.3)), cf. (190). (Since @SPEC-DP is a macro, there is no overt node labelled SPEC-DP. Instead, the genitive DP, of category NAMEP, is directly dominated by the DP.)

(190)   *Karls Frau*
         Karl's wife

```
CS 1:    DP[std]
            |
         DPx[std]
           / \
      NAMEP   NP
         |     |
      Karls  N[comm]
               |
             Frau
```
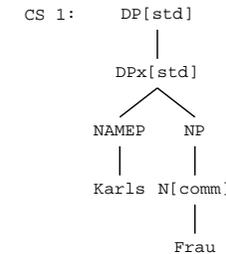
```
DPx[_type] -->
        ...
    | "with SPEC-DP, no D head
       EX: Karls Hund (Karl's dog)"
       @(SPEC-DP _type)
       NP
    | ...
```

Note that, if a specifier in the SPEC-DP position is present, there cannot be a determiner (191).

(191)   **Karls   ein Hund*
          K.[GEN] a   dog

Hence, we again skip the Dbar projection and introduce NP as the sister of SPEC-DP, see the above rule.

Note that skipping the Dbar projection also rules out sentences that contain a SPEC-DP followed by a quantifying adjective, as in (192), (193) (since Aquant is not part of the NP projection in our implementation (cf. p. 178)). However, these constructions seem to be rare.

(192)   *in Gorbatschows beiden   Büchern*
          in [G. SPEC-DP] [both Aquant] books
          'in both of Gorbachev's books'

(193)   *Karls   eines   Fahrrad*
         [K. SPEC-DP] [one Aquant] bicycle
         'Karl's one bicycle'

10.2.3   DP → D NP

As a third alternative, DP dominates a determiner and an (optional) NP (covered by the macro @Dbar (cf. sec. 10.4)) (194).

(194)   *eine Frau*
         a   woman

```
CS 1:      DP[std]
              |
           DPx[std]
              |
         D[std]    NP
            |       |
          eine   N[comm]
                    |
                  Frau
```
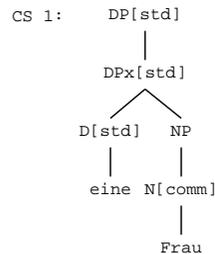
```
DPx[_type] -->
       ...
     | "with D head, no SPEC-DP
         (use macro to keep c-structure flat)
         EX: ein Hund (a dog)"
       e: _type ~= rel;
       @(Dbar _type)
     | ...
```

Since there are no relative determiners (D[rel]) in German, this alternative is ruled out for relative DPs (_type = rel).

In German, relative clauses are marked exclusively by the specifier in SPEC-DP as in (195) or by a relative pronoun as in (196).

(195)   *der Mann, dessen   Frau* ...
         the man   [whose SPEC-DP] wife

(196)   *der Mann, der   ...*
         the man   [who PRON]

In contrast, interrogativity can be marked by a specifier in SPEC-DP (197), by an interrogative determiner (D[int]) (198), or by an interrogative pronoun (199).

(197)   *wessen   Frau*
         [whose SPEC-DP] wife

(198)   *welcher   Mann*
         [which D] man

(199)   *wer*
         [who PRON]

10.2.4   DP → PRON

In the theoretical introduction, we distinguished three types of pronouns (cf. sec. 7.4): (i) "determiner-pronouns" (i.e. determiners that are used intransitively), (ii) "adjective-pronouns" (i.e. quantifying adjectives that are used intransitively), and (iii) "real pronouns" (i.e. pronouns that never occur in the D or Aquant position).

The third type of pronoun is represented by the category PRON, which is introduced as the daughter of the DP, cf. (200).

(200)   *jemand*
         someone

```
CS 1:      DP[std]
              |
           DPx[std]
              |
           PRON[std]
              |
            jemand
```
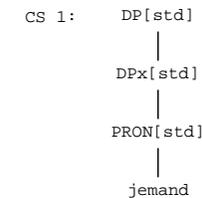
```
DPx[_type] -->
       ...
     | "pronouns"
       PRON[_type]

       ( "optional constituents may follow pronoun"
         "EX: jemand Nettes aus Stuttgart "
         "(someone nice from Stuttgart)"
         @(PRON-APPOS) )
     | ...
```

Certain pronouns can be modified by a special type of appositive (APP), covered by the macro @PRON-APPOS, as in (201).

(201)   *jemand   Nettes*
         someone nice

Alternatively, *jemand* 'someone' could be to analyzed as a determiner which is followed by the head noun, *Nettes* 'nice'. However, the fact that case and gender of the pronoun (= the putative determiner) and the putative head noun may disagree indicates that this is an appositive construction (cf. p. 165).

Due to space limitations, we cannot present a fully detailed analysis of pronouns. That is, there are no sections documenting the sublexical rules, morphological analysis, or lexicon entries of pronouns of cateogry PRON.

10.3   @SPEC-DP

The macro @SPEC-DP covers the SPEC-DP position. It is exclusively occupied by genitive DPs, which function as possessives (SPEC POSS), e.g. *Karls* 'Karl's' as in (202).

(202)   *Karls   Frau*
         K.[GEN] wife
         'Karl's wife'

```
CS 1:     DP[std]
             |
          DPx[std]
           /     \
       NAMEP     NP
          |       |
       Karls    N[comm]
                   |
                 Frau
```
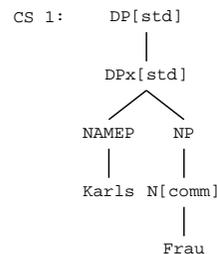
```
"Karls Frau"

    ⎡PRED 'Frau'              ⎤
 18 ⎢SPEC ⎡POSS 0⎡PRED 'Karl'⎤⎤⎥
    ⎣     ⎣      ⎣           ⎦⎦⎦
```

In our implementation, we restrict SPEC-DP to names (sec. 10.3.2) and certain pronouns (sec. 10.3.1). As a consequence, the SPEC-DP position is realized

by constituents of categories NAMEP or PRON (more exactly: PRONspec, see below), rather than DP.

This restriction is based on corpus frequencies. In the NEGRA corpus, there are 483 instances of SPEC-DP.

- 72.3% (= 349 instances) of these SPEC-DP instances consist of proper nouns.

- 23.4% (= 113 instances) are pronouns: 14% relative pronouns and 9% demonstrative ones. All pronominal examples are instances of the lemma *die* 'this' (which can be used as a relative or demonstrative pronoun).

  (Note that in NEGRA, relative and demonstrative pronouns are analyzed as "attributive pronouns", functioning as 'NK' ("noun kernel modifier"). That is, they are not marked as genitive DPs, which function as 'GL' ("genitive left".)

- The remaining 4.3% (= 21 instances) are ordinary DPs.

The above numbers show that ordinary DPs realizing SPEC-DP are rare. Corpus examples of such DPs are (203) and (204).

(203)   (see corpus example DP10.1, Appendix A)
         Babys   Herz
         [baby's SPEC-DP] heart
         'the baby's heart'

(204)   (see corpus example DP10.2, Appendix A)
         in des Gegners   Netz
         [in the opponent's SPEC-DP] net

However, most of the corpus instances with ordinary DPs in SPEC-DP are figures of speech, as in (205) or (206).

(205)   *des Rätsels   Lösung*
         [the riddle's SPEC-DP] solution
         'the solution to the riddle'

(206)   *auf des Messers   Schneide*
         on [the knife's SPEC-DP] edge
         'on a knife's edge'

10.3.1   @SPEC-DP = PRON

In our implementation, only the pronouns *die* 'this' and *wer* 'who' may realize the SPEC-DP projection. The respective genitive forms are *dessen* 'this[MASC/NEUT,SG,GEN]', *deren* 'this[FEM,SG,GEN]'/ 'these[PL,GEN]', and *wessen* 'whose'.

Since we only allow for three different forms, these forms are listed in the lexicon and assign a specific category, PRONspec, cf. the full form entry of *wessen* 'whose' and the f-structure projected by this entry. (For the feature CHECK _NMORPH _GENITIVE (cf. sec. 10.3.3).)

```
wessen    !PRONspec[int] * @(PRON wer) @(PRON-TYPE int)
                           @(GEND masc) @(NUM sg) @(CASE gen)
                           (^CHECK _NMORPH _GENITIVE) = +; ETC.
```

$$\begin{bmatrix} \text{PRED} & \text{'pro'} \\ \text{CHECK} & \left[\text{\_NMORPH} \left[\text{\_GENITIVE +}\right]\right] \\ \text{NTYPE} & \left[\text{GRAIN pronoun}\right] \\ \text{CASE gen, GEND masc, NUM sg, PRON-FORM wessen, PRON-TYPE int} \end{bmatrix}$$
0

An example featuring PRONspec is given in (207).

(207)   *wessen Frau*
         whose wife

```
CS 1:        DP[int]
               |
             DPx[int]
              / \
   PRONspec[int] NP
         |        |
      wessen    N[comm]
                  |
                 Frau
```

"wessen Frau"

$$36\begin{bmatrix} \text{PRED} & \text{'Frau'} \\ \text{SPEC} & \left[\text{POSS } 0\left[\text{PRED 'pro'}\right]\right] \end{bmatrix}$$

PRONspec is introduced by the following disjunct of @SPEC-DP.

```
SPEC-DP(_type) =
        ...
     | "certain pronouns 'dessen/deren/wessen'"
       PRONspec[_type]: @(PRENOM-GENITIVE);
     | ...
```

All specifiers in SPEC-DP are represented by the feature SPEC POSS, via the template @PRENOM-GENITIVE. In addition, special restrictions on the form of the genitive DP need to be encoded; this is effected by the template @CASE_desig (cf. sec. 10.3.3).

```
PRENOM-GENITIVE =
        (^SPEC POSS) = !
        "EX: Karls Hund (Karl's dog)"
        @(CASE_desig ! gen)
        "must be morphologically marked:"
        "EX: *Karl Hund (Karl dog)".
```

10.3.2   @SPEC-DP = NAMEP

Besides pronouns, proper names (of category NAMEP) may occupy the SPEC-DP position, cf. (208).

(208)   *Professor   Karl Maiers   Frau*
         [professor K.   M. SPEC-DP] wife
         'Professor Karl Maier's wife'

```
CS 1:        DP[std]
               |
             DPx[std]
              /      \
          NAMEP       NP
          / | \        |
  N[title] NAME NAME N[comm]
     |      |    |     |
 Professor Karl Maiers Frau
```

"Professor Karl Maiers Frau"

$$55\begin{bmatrix} \text{PRED 'Frau'} \\ \text{SPEC} \left[\text{POSS } 34\begin{bmatrix} \text{PRED} & \text{'Maier'} \\ \text{NAME-MOD} \left\{ 0\begin{bmatrix} \text{PRED 'Professor'} \\ >\text{s} & \left(\left[18:\text{Karl}\right]\right) \\ 18\left[\text{PRED 'Karl'}\right] \end{bmatrix}\right\} \end{bmatrix}\right] \end{bmatrix}$$

Similarly to above, SPEC-DP makes use of the template @PRENOM-GENITIVE.

```
SPEC-DP(_type) =
        ...
      | "proper nouns"
        NAMEP: _type = std
               @(PRENOM-GENITIVE);

        ( APOSTROPHE: (^SPEC POSS) = !
                       "EX: Hans' Auto (Hans's car)"
                       "EX: *Paris' Bahnhöfe "
                       "(the stations of Paris)"
                       { (^SPEC POSS NSEM PROPER) =c first_name
                       | (^SPEC POSS NSEM PROPER) =c last_name };
        )
      | ...
```

   The optional category APOSTROPHE in the above rule accounts for the fol-
lowing data. Person names ending with *-s* such as *Hans* are not inflected for
genitive in German (since this would yield the unusual ending *-ss*). Instead they
are followed by an apostrophe, which marks the missing *-s*. The apostrophe is
analyzed as a token of its own by our tokenizer, as shown in (209).

(209)   *Hans' Auto*
         H.   car
         'Hans's car'

```
"Hans' Auto"

      ┌                                                              ┐
      │PRED   'Auto'                                                 │
      │CHECK  [_SPEC-TYPE [_COUNT +, _DEF +]]                        │
      │NSEM   [COMMON count]                                         │
      │NTYPE  [NSYN common]                                          │
      │        ┌              ┌                              ┐  ┐    │
      │        │              │PRED   'Hans'                 │  │    │
      │        │              │CHECK  [_NMORPH [_GENITIVE apostrophe]]│
      │SPEC    │POSS          │NSEM   [PROPER first_name]    │  │    │
      │        │              │NTYPE  [NSYN proper]          │  │    │
      │        │             0│CASE gen, GEND masc, NUM sg   │  │    │
   24 │GEND neut, NUM sg, PERS 3                                     │
      └                                                              ┘
```

The presence of the apostrophe is recorded by the feature CHECK _NMORPH
GENITIVE with the value apostrophe, introduced in the lexicon entry of ' '', i.e.
the apostrophe (for the feature CHECK _NMORPH _GENITIVE, see the following
section).

```
'        APOSTROPHE * "EX: Hans' Auto (Hans's car)"
                      (^CHECK _NMORPH _GENITIVE) = apostrophe_;
         ONLY.
```

   The apostrophe is not an option for other, non-person proper names, such
as cities or countries (210). This restriction is encoded by use of special NSEM
PROPER features for the different types of (person) proper names, see the above
disjunct of @SPEC-DP.

(210)   **Paris' Bahnhöfe*
         Paris stations

   The apostrophe is only admissible in the case of prenominal genitives. That is,
postnominal genitives or genitive objects are never followed by an apostrophe. We
therefore add the apostrophe in the SPEC-DP rule rather than analyzing it as a kind of
name suffix, by adding it to the sublexical rules of names.

   Note that we have to allow for an apostrophe after any person name, not just the
ones ending with *-s*, because our morphological analyzer does not provide any feature
marking the last character of a word. That is, we allow for examples such as (211).

(211)   **Karl' Auto*
         Karl' car

10.3.3 Genitive Restrictions

For genitive DPs, there are special restrictions on the morphological form of the DP. The restrictions hold for any genitive DP, i.e. for genitive DPs functioning as complements of verbs, adjectives, or prepositions as well as genitive DPs modifying nouns (prenominal and postnominal; for differences between these two (cf. sec. 7.5); since we only allow for very simple prenominal DPs, we can ignore these differences).

The data The DP must be overtly marked for genitive case. We illustrate this restriction by DPs consisting of proper names. Person names come in two different forms in genitive case, either without any inflection (*Goethe*) or with an *-s* attached (*Goethes*), as in (212)

(212)   *die Gedichte Goethes*
      the poems   G.[GEN]
      'Goethe's poems'

They may be uninflected provided overt genitive marking is realized on some other constituent of the genitive DP, e.g. the determiner as in (213), vs. (214).

(213)   *die Gedichte des   jungen Goethe*
      the poems   the[GEN] young G.[UNINFL]
      'the poems of the young Goethe'

(214)   **die Gedichte Goethe*
      the poems   G.[UNINFL]

In the following, we present our analysis of the morphological constraints. We restrict the presentation to proper nouns, since the SPEC-DP position in our implementation is confined to proper nouns. However, the same morphological restrictions apply to common nouns.

Note that at the syntactic level, genitive DPs realized by common and proper nouns do differ. A common noun by itself (i.e. without a specifier or attributive adjectives) usually cannot represent a genitive DP, cf. (215) and (216).

(215)   **der Verkauf Brots*
      the sale   bread[GEN]

(216)   **Brots   Verkauf*
      bread[GEN] sale

In contrast, a proper noun by itself may function as a genitive DP, cf. (217) and (218).

(217)   *der Verkauf Alaskas*
      the sale   A.[GEN]
      'the selling of Alaska'

(218)   *Alaskas Verkauf*
      A.[GEN] sale
      'the selling of Alaska'

Our grammar encodes these constraints within the NP projection, hence we do not discuss them here.

The analysis Uninflected proper nouns are marked as .NGDA by the morphology, inflected (genitive) proper nouns by .Gen, cf. the morphological analyses of *Karl* and *Karls*.

```
M-Input    Karl  ('Karl')
M-Output  Karl +NPROP .Masc .NGDA .Sg

M-Input    Karls  ('Karl's')
M-Output  Karl +NPROP .Masc .Gen .Sg
```

That is, the tag .Gen marks any noun, adjective, or determiner which is morphologically (overtly) marked as a genitive. Hence, expressions marked by .Gen fulfil the above restrictions on genitive DPs. To make this property visible in the f-structure, the tag .Gen introduces a special feature CHECK _NMORPH _GENITIVE, whereas the tag .NGDA does not introduce any f-structure restriction, compare the lexicon entries below.

```
.NGDA     CASE-F xle "noun, adj/quant"; ONLY.

.Gen      CASE-F xle @(CASE gen)
                    (^CHECK _NMORPH _GENITIVE) = +;
          ADPOS-GEN-F xle "secondary prepositions"
                    (^OBJ PCASE) =c gen
                    (^CHECK _LEX _SECOND-PREP) = +;
          ONLY.

CASE(_case) = @(CASE_desig ^ _case).
```

Prenominal genitives call the template @PRENOM-GENITIVE, which in turn calls the general case template @CASE_desig.

```
PRENOM-GENITIVE =
      (^SPEC POSS) = !
      "EX: Karls Hund (Karl's dog)"
      @(CASE_desig ! gen)
      "must be morphologically marked:"
      "EX: *Karl Hund (Karl dog)".
```

If the template @CASE_desig is called with _case = gen, the feature CHECK _NMORPH _GENITIVE is required to be present in the DP's f-structure (designated by the parameter _desig, e.g. _desig = !).

```
CASE_desig(_desig _case) =
      (_desig CASE) = _case
      { _case ~= gen
      | "special restrictions for genitive case
        genitive must be marked overtly"
       _case = gen
       (_desig CHECK _NMORPH _GENITIVE) }.
```

Hence, only case-marked proper nouns (i.e. marked by the morphological tag .Gen, e.g. *Karls*) or proper nouns that are accompanied by some case-marked constituent are allowed. Examples of such constituents are determiners (219) or attributive adjectives (220).

(219)   *des   Karl*
        the[GEN] K.[UNINFL]
        'of Karl'

```
"des Karl"

⎡PRED   'Karl'                                              ⎤
⎢      ⎡_NMORPH    ⎡_GENITIVE +⎤                     ⎤      ⎥
⎢CHECK ⎢                                             ⎥      ⎥
⎢      ⎣_SPEC-TYPE ⎣_COUNT +, _DEF +, _DET attr⎦     ⎦      ⎥
⎢NSEM  ⎡PROPER first_name⎤                                  ⎥
⎢NTYPE ⎡NSYN proper⎤                                        ⎥
⎢      ⎡   ⎡PRED     'die'⎤⎤                                ⎥
⎢SPEC  ⎢DET⎢               ⎥⎥                               ⎥
⎢      ⎣   ⎣DET-TYPE def  ⎦⎦                                ⎥
⎣0 CASE gen, GEND masc, INFL strong-det, NUM sg, PERS 3     ⎦
```

(220)   *blauer   BMWs*
        blue[PL,GEN] BMW[PL]
        'of blue BMWs'

```
"blauer BMWs"

⎡PRED    'BMWs'                                              ⎤
⎢       ⎧⎡PRED   'blau<[49:BMWs]>'                ⎤⎫         ⎥
⎢       ⎪⎢SUBJ   [49:BMWs]                        ⎥⎪         ⎥
⎢ADJUNCT⎨⎢      ⎡_AMORPH ⎡_AHEAD base_⎤⎤          ⎬         ⎥
⎢       ⎪⎢CHECK ⎢                      ⎥          ⎪         ⎥
⎢       ⎪⎢      ⎣_MORPH  ⎣_CAPITAL -⎦ ⎦          ⎪         ⎥
⎢       ⎩⎣0 ADEGREE base_, ATYPE attributive     ⎭         ⎥
⎢       ⎡_NCONSTR ⎡_ADJ-ATTR +⎤                  ⎤         ⎥
⎢CHECK  ⎢                                         ⎥         ⎥
⎢       ⎣_NMORPH  ⎣_GENITIVE +⎦                  ⎦         ⎥
⎢NSEM   ⎡PROPER guessed⎤                                    ⎥
⎢NTYPE  ⎡NSYN proper⎤                                       ⎥
⎣49 CASE gen, INFL strong-adj, NUM pl, PERS 3              ⎦
```

Adjectives embed all morpho-syntactic features—including restrictions on case, such as the feature CHECK _NMORPH _GENITIVE—under the function SUBJ, cf. the f-structure projected by the adjective *blauer* 'blue[PL,GEN]'.

```
⎡PRED    'blau<[-1]>'                              ⎤
⎢       ⎡CHECK ⎡_NMORPH ⎡_GENITIVE +⎤⎤            ⎥
⎢SUBJ   ⎢                                ⎥        ⎥
⎢       ⎣-1 CASE gen, INFL strong-adj, NUM pl⎦    ⎥
⎢ADEGREE base_                                     ⎥
⎣0 CHECK  ⎡_AMORPH ⎡_AHEAD base_⎤⎤               ⎦
```

The function SUBJ is unified with the f-structure of the head noun (*BMW* 'BMW') that is modified by the adjective (cf. sec. 9.2).

Besides the morphological tag .Gen, the apostrophe introduces the genitive feature in its lexicon entry, compare the entry below. Hence, the expression *Hans'* fulfils the restrictions by @CASE_desig and may function as prenominal genitive.

```
'         APOSTROPHE * "EX: Hans' Auto (Hans's car)"
                       (^CHECK _NMORPH _GENITIVE) = apostrophe_;
          ONLY.
```

Our implementation overgenerates in that it allows for (221).

(221)   **das Auto der   Marias*
        the car the[GEN] M.[GEN]

For masculine nouns, however, the data are not so clear, compare (222) and (223). (Our implementation allows for both variants.)

(222) *das Auto des   Karl*
       the car the[GEN] K.[UNINFL]
       'Karl's car'

(223) (*) *das Auto des   Karls*
          the car the[GEN] K.[GEN]
          'Karl's car'

Apparently, for feminine names, only one overt genitive marking is admitted.

For masculine names, there seems to be a difference between person names and geographic names. For person names, the preferred form allows only one overt genitive marking, similar to feminine names. In contrast, river names preferably come with two markings (224). (For instance, in the FR corpus there is only one match for *des Main* 'the[GEN] Main[UNINFL]' vs. 137 matches for *des Mains* 'the[GEN] Main[GEN]'.)

(224) *südlich des   Mains*
       south the[GEN] Main[GEN]
       'south of the river Main'

Region names seem more flexibel, compare (225) and (226). (The lemma *Balkan* is inflected in 76.6% of the 30 genitive instances in the FR corpus).

(225) *des   Balkan*
       the[GEN] Balkan[UNINFL]

(226) *des   Balkans*
       the[GEN] Balkan[GEN]

10.4   @Dbar

The macro @Dbar expands to one or more quantifying expressions followed by an optional NP. The quantifying expression can consist of a determiner (D) and/or quantifying adjective (Aquant) (for the differences between D and Aquant (cf. sec. 10.5)).

The value of the parameter _type is determined by the first quantifying expression. The basic structure of the @Dbar macro thus looks as follows.

```
Dbar(_type) = {  D[_type]
                 ( Aquant[std] )
              | Aquant[_type] }
                 ( NP ).
```

We start with the first part of the @Dbar macro, covering the quantifying expression.

10.4.1   The Quantifying Expression: D/Aquant

(i) Determiner (D)   D can be realized by any expression (articles, possessives, demonstratives, indefinites, interrogatives) that inflects like a determiner (227).

(227) *eine Frau*
       a   woman

```
            CS 1:   DP[std]
                       |
                    DPx[std]
                     /    \
                 D[std]   NP
                    |      |
                  eine  N[comm]
                           |
                         Frau
```
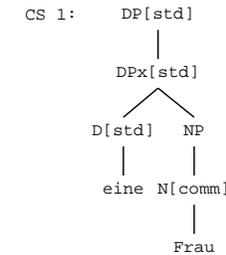
```
"eine Frau"
      ┌                                              ┐
      │ PRED   'Frau'                                │
      │ CHECK  [_SPEC-TYPE [_COUNT +, _DET attr]]    │
      │ NSEM   [COMMON count]                        │
      │ NTYPE  [NSYN common]                         │
      │        ┌     ┌ PRED     'eine'  ┐ ┐          │
      │ SPEC   │ DET │ DET-TYPE indef   │ │          │
      │        └     └                  ┘ ┘          │
    0 │ GEND fem, INFL strong-det, NUM sg, PERS 3   │
      └                                              ┘
```

```
Dbar(_type) =
    ...

        "EX: der Mann (the man)"
        "EX: alle Leute (all people)"
        "EX: welche beiden Frauen (which two women)"
        D[_type]: @(RIGHT-SISTER);
        Aquant[std]*
    ...
```

As we have seen above (cf. p. 160), not all determiners allow for omission of the NP. This restriction is handled by the template @RIGHT-SISTER (cf. sec. 10.10).

The determiner may be followed by quantifying adjectives, as in (228). Such multiple quantifying expressions are discussed below in detail (cf. sec. 10.9))

(228)   *das selbe Kind*
         the same child

Certain adverbs, like *genug* 'enough', may also represent a determiner, cf. (229).

(229)   *genug   Leute*
         enough people

```
CS 1:      DP[std]
              |
           DPx[std]
              |
         +----+----+
       D[std]      NP
         |          |
       genug     N[comm]
                    |
                  Leute
```
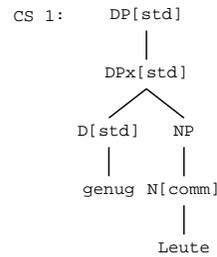
```
"genug Leute"

  ⎡PRED   'Leute'              ⎤
  ⎢NSEM   [COMMON count]       ⎥
  ⎢NTYPE  [NSYN comm  ]        ⎥
  ⎢SPEC   [QUANT [PRED 'genug']]⎥
 0⎣NUM pl, PERS 3              ⎦
```

Such adverbs are listed by full form entries in the lexicon, cf. the entry of *genug* 'enough'. (For the template @SPEC-MASS (cf. sec. 10.11.1).)

```
genug     D[std] * @(QUANT %stem) @(SPEC-MASS); ETC.
```

(ii) Quantifying adjective (Aquant)   Alternatively, a quantifying adjective, covering all quantifying expressions that inflect like adjectives, can be the "head" of DP (230).

(230)   *mehrere Kinder*
         some   children
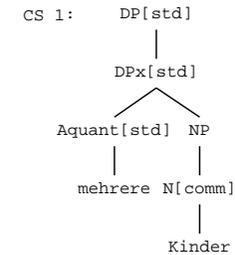
```
CS 1:      DP[std]
              |
           DPx[std]
              |
         +----+----+
     Aquant[std]   NP
         |          |
      mehrere    N[comm]
                    |
                  Kinder
```

```
Dbar(_type) =
    ...
    "EX: wieviele Frauen (how many women)"
    "EX: einige wenige Frauen (some few women)"
    Aquant[_type]
    Aquant[std]*
    ...
```

Again, multiple quantifying expressions are allowed, as in (231) (cf. sec. 10.9).

(231)   *einige wenige Kinder*
         some   few   children

10.4.2   The Optional NP

We now come to the second part of @Dbar, covering the NP. This NP is optional, i.e. the head noun can be omitted (which we call "empty head noun").

If the NP is omitted, this results in the so-called "determiner-pronouns" (232) or "adjective-pronouns" (233).

(232)   *dieses*
         this

```
CS 1:      DP[std]
              |
           DPx[std]
              |
           D[std]
              |
           dieses
```
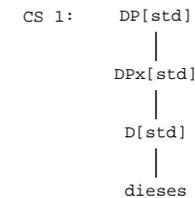
```
                "dieses"

            ⎡PRED 'pro'                    ⎤
          0 ⎣SPEC [DEM [PRED 'diese']]     ⎦
```

(233)   *mehrere*
        some

```
              CS 1:     DP[std]
                           |
                        DPx[std]
                           |
                       Aquant[std]
                           |
                         mehrere
```

```
                "mehrere"

            ⎡PRED 'pro'                     ⎤
          0 ⎣SPEC [QUANT [PRED 'mehrere']]  ⎦
```

If the NP is omitted, the equation (↑PRED) = 'pro' is introduced to the DP's f-structure, by the template @NULL, to indicate that the referent of the DP is to be determined by the context. .

```
Dbar(_type) =
      ...

      { NP
      | "empty head noun
        EX: viele der Gäste (many of the guests)"
        e: @(NULL ^)
           @(IF ~(^INFL) "for invariant determiners: no OBJ2"
           ~(OBJ2 ^) "EX: *Er hilft allerlei. "
           );
      }
      ...


NULL(_desig) =
      "for null referents (= empty head of adjectives/quants)"
```

```
{ (_desig PRED) = 'pro'
| "possibly predicative"
  (_desig PRED) = 'pro<(_desig SUBJ)>'
  (_desig XCOMP-TYPE)
}
@(PRON-TYPE_desig _desig null).
```

As an example, cf. the f-structure of (234).

(234)   *dieses*
        this

```
        "dieses"

        ⎡PRED  'pro'                                        ⎤
        ⎢CHECK [_SPEC-TYPE [_COUNT +, _DEF +]]              ⎥
        ⎢NTYPE [NSYN pronoun]                               ⎥
        ⎢SPEC  [DEM [PRED 'diese']]                         ⎥
      0 ⎣GEND neut, INFL strong-det, NUM sg, PERS 3, PRON-TYPE null⎦
```

If the quantifying expression is uninflected (i.e. does not project a feature INFL), additional constraints, on the function, apply. With the NP omitted, the DP may not function as the indirect object (OBJ2), compare (235), but not the indirect object (236).

(235)   *Hans weiß   allerlei.*
        H.   knows [various OBJ]
        'Hans knows various things.'

(236)   **Hans hilft   allerlei.*
        H.   helps [various *OBJ2]
        '(intended meaning: Hans helps various people.)'

Note that the data are not as clear, e.g. (237) seems much better than the above example.

(237) (?)*Hans geht allerlei   auf den Grund.*
        H.   goes [various OBJ2] on the bottom
        'Hans gets to the bottom of various things.'

However, *allerlei* 'various (things)' cannot be replaced by *genug* 'enough' in this example, cf. (238) (intended meaning: Hans gets to the bottom of sufficiently many things.).

(238) (*)*Hans geht genug   auf den Grund.*
        H.   goes [enough *OBJ2] on the bottom

Finally, note that the semantic difference between, e.g., *genug* 'enough' functioning as the object and *genug* functioning as a quantity modifier is often subtle, cf. (239).

(239) *Hans ißt genug.*
       H.   eats enough

10.5   D/Aquant

Quantifying expressions occur very frequently and, therefore, represent an important construction in German. In the following sections, we provide a very detailed presentation of the implementation. The casual reader may simply read the rough overview below, which addresses the important points of the subsequent sections.

The basic difference between the categories D and Aquant is the inflectional behaviour of attributive adjectives that follow D/Aquant. In our implementation of D and Aquant, different levels of representation interact: (i) morphology (sec. 10.6), (ii) lexicon entries (sec. 10.7), and (iii) sublexical rules (sec. 10.8). Through their interaction emerges a puzzling complexity and variety of forms.

Overview   Quantifying expressions can be articles, possessives, relatives, demonstratives, indefinites, and interrogatives. The expressions are marked by the morphological tags .Attr, .Subst, and .Pro. These tags are related to the syntactic category of the expressions as follows: .Attr corresponds to the category D; .Subst corresponds to PRON; .Pro maps to D as well as Aquant, as illustrated by the following figure.

```
.Attr  ──────────▶ D
.Pro   ────────╲╱──▶ Aquant
.Subst ──────────▶ PRON
```
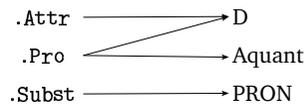
Lexicon entries may further specify categorial properties of expressions that are marked by .Pro. That is, lexicon entries may disambiguate D vs. Aquant. For instance, the stem category INDEF-S[det] determines that the quantifying expression in question must be of category D. Conversely INDEF-S[adj] determines that the expression must be of category Aquant.

```
INDEF-S[det] ──────────▶ D
INDEF-S    ────────╲╱
INDEF-S[adj] ──────────▶ Aquant
```
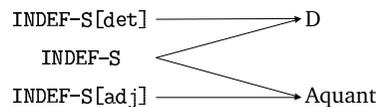
Multiple quantifying expression can occur, either in the form of multiple determiners (Dpre plus D) or in the form of determiners followed by quantifying adjectives (D plus Aquant). In both cases, restrictions on possible combinations are stated in lexical stem entries.

(For an exhaustive list of all quantifying expressions in the current version of our morphology (cf. sec. 11.3.2).)

10.6   D/Aquant: Morphology

The morphological analyzer provides four types of tags for quantifying expressions. These tags encode (i) the lemma form (sec. 10.6.1), (ii) the part of speech (sec. 10.6.2), (iii) the distribution (sec. 10.6.3), and (iv) the inflection (sec. 10.6.4), as illustrated by the tags of *manch* 'some'.

Lemma Form	Part of Speech	Distribution	Inflection
manch	+INDEF	.Attr	.Invar

10.6.1   The Lemma Form

As presented in the introductory chapter (cf. p. 36), the lemma of a word usually corresponds to the uninflected nominative singular. In case there is no uninflected or unambiguous form in nominative singular, the feminine form is chosen. Among other expressions, this concerns inflected quantifying expressions and pronouns.

As an example, compare the morphological analyses of *manch* 'some[UNINFL]' and *manchem* 'some[MASC/NEUT,SG,DAT]' with the lemmas *manch* 'some[UNINFL]' and *manche* 'some[FEM,SG,NOM]', respectively.

```
M-Input    manch    ('some')
M-Output   manch +INDEF .Attr .Invar
```

```
M-Input    manchem    ('some')
M-Output   manche +INDEF .Pro .MN .Dat .Sg .St
```

Idiosyncratic restrictions concerning only the inflected lemma or only the invariant lemma can thus be easily stated in the lexicon. For instance, only the invariant *manch* 'some' can be used as an initial determiner (Dpre) within multiple determiners, compare (240) and (241) (cf. sec. 10.9.1).

(240) *manch   ein   Assistent*
       some[UNINFL] a[UNINFL] assistant
       'some assistant'

(241) *_mancher ein   Assistent_
          some[ST] a[UNINFL] assistant

Note that many linguists assume that inflected and invariant forms are variants of the same lemma, e.g. Pafel (1994). Under this view, assuming two different lemmas can be seen merely as a technical device, to facilitate lexical encoding of distinguishing properties.

In certain cases, our version of the morphological analyzer provides special lemma forms, to distinguish otherwise ambiguous (homonymous) lemmas. For instance, the definite article _die_ 'the', the demonstrative pronoun _die_ 'this', and the relative pronoun _die_ 'that' all are actually to be assigned the (three-way ambiguous) lemma form 'die' (since in all three cases, _die_ is the feminine form of nominative singular; for the paradigms (cf. p. 161)).

The form 'die' is disambiguated by suffixes: 'die_art' is the lemma form assigned to the article _die_ 'the', whereas 'die_dem' and 'die_rel' are associated with the demonstrative and relative pronoun _die_ 'this/that', respectively; cf. the morphological analysis of _dem_ 'the/this/that[MASC/NEUT,DAT]'.

```
M-Input    dem   ('the/this/that')
M-Output   die_art +ART .Def .MN .Dat .Sg .St
           die_dem +DEM .Subst .MN .Dat .Sg .St
           die_rel +REL .Subst .MN .Dat .Sg .St
```

Similarly, the indefinite article _eine_ 'a', the cardinal _eine_ 'one', and the pronoun _eine_ '(some)one' are assigned specific lemma forms by the morphological component, cf. the analysis of _einem_ 'a/(some)one[MASC/NEUT,DAT]'.

```
M-Input    einem   ('a/one')
M-Output   eine_art +ART .Indef .MN .Dat .Sg .St
           eine_card +CARD .Attr .MN .Dat .Sg .St
           eine_indef +INDEF .Pro .Indef .MN .Dat .Sg .St
```

Similar to the varying lemma forms of _manch_ and _manche_ 'some' above, the specific lemma forms 'die_art', 'die_dem', etc. facilitate the encoding of idiosyncratic, lexical restrictions that only hold of one of the lemmas.

10.6.2   The Part of Speech

Quantifying expressions can be articles (marked by the part-of-speech tag +ART), possessives (+POSS), demonstratives (+DEM), indefinites (+INDEF), and interrogatives (+WPRO).

The part-of-speech tag follows the lemma form in the morphological analysis, cf. the morphological analysis of _kein_ 'no'.

```
M-Input    kein   ('no')
M-Output   keine +INDEF .Attr .Neut .NA .Sg .None
           keine +INDEF .Attr .Masc .Nom .Sg .None
```

10.6.3   Distribution

Three morphological tags encode distributional properties: .Subst, .Attr, and .Pro.

We first describe the tags .Subst, .Attr, and .Pro and then outline which categories in our grammar these tags correspond to.

.Subst   Expressions marked as .Subst ("substitutional") are quantifying expressions that can only represent a full DP. That is, they are obligatorily "intransitive" (in the sense introduced above (cf. sec. 7.4.3)) and correspond to the "real pronouns".

As examples, see the morphological analyses of _dessen_ 'this[GEN]' and _jemand_ 'someone' below. The distributional tag .Subst follows the part-of-speech tag (in these examples, +DEM and +INDEF).

```
M-Input    dessen   ('this')
M-Output   die_dem +DEM .Subst .MN .Gen .Sg .St
```

```
M-Input    jemand   ('someone')
M-Output   jemand +INDEF .Subst .Invar
```

These pronouns cannot be discussed in detail, due to space limitations.

.Attr   .Attr ("attributive") is the counterpart of .Subst and marks quantifying expressions that obligatorily precede an NP. That is, they never represent a DP on their own but are obligatorily "transitive". (The position of expressions of type .Attr—always in front of an NP—is similar to that of attributive adjectives, hence the name .Attr.). Examples are _kein_ 'no' and _welch_ 'which'.

```
M-Input    kein   ('no')
M-Output   keine +INDEF .Attr .Neut .NA .Sg .None
           keine +INDEF .Attr .Masc .Nom .Sg .None
```

```
M-Input    welch   ('which')
M-Output   welch +WPRO .Attr .Invar
```

.Attr expressions constitute a morphologically marked class. The "typical" representative of .Attr is uninflected: *all* 'all', *manch* 'some', *welch* 'which'. In addition, the "mixed" class of determiners (with uninflected forms in nominative singular (cf. p. 155)) belongs to .Attr, e.g. *keine* 'no', *meine* 'my'.

.Pro   .Pro marks quantifying expressions that can be either "transitive" or "intransitive". That is, they either precede an NP (similar to .Attr), or else they represent a DP on their own (similar to .Subst). Examples are *dieses* 'this' and *irgendwelche* 'any'.

```
M-Input    dieses   ('this')
M-Output   diese +DEM .Pro .Neut .NA .Sg .St
           diese +DEM .Pro .MN .Gen .Sg .St


M-Input    irgendwelche  ('any')
M-Output   irgendwelche +INDEF .Pro .Fem .NA .Sg .St
           irgendwelche +INDEF .Pro .MFN .NA .Pl .St
```

We now outline how the tags .Subst, .Attr, and .Pro relate to the categories in our grammar. The following picture illustrates the mapping from the distribution tags to the syntactic categories PRON, D, and Aquant.

```
.Attr  ══════════════➤ D
.Pro   ══════════════➤ Aquant
.Subst ──────────────➤ PRON
```
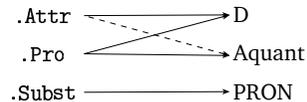
.Subst → PRON   Expressions of type .Subst, corresponding to the "real pronouns", are represented by the category PRON (242).

(242)   *jemand*
        someone

```
      CS 1:            PRON[std]
         ┌──────────┬──────────┬──────────┐
PRON-S_BASE PRON-T_BASE[std] DISTR-SUBST_BASE INVAR-F_BASE
     │          │              │             │
  jemand     +INDEF         .Subst        .Invar
```
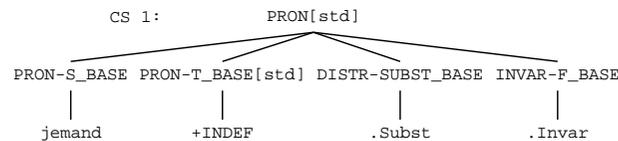
.Attr → D (/Aquant)   Expressions of type .Attr always precede an NP. In our grammar, these are quantifying expressions whose c-structure complement, the NP, may not be empty.

This property (requiring a non-empty NP) is not by itself related to the inflectional properties of a given quantifying expression. That is, a .Attr expression could in principle inflect like a determiner (D) or like a quantifying adjective (Aquant). Interestingly, however, all expressions of type .Attr come out as determiners (D) according to our inflection-based criterion for determiners. (Therefore, the line in the above picture, which connects .Attr and Aquant, is dotted only.)

As an example of an .Attr expression, cf. (243).

(243)   *welch*
        which

```
      CS 1:            D[int]
         ┌──────────┬──────────┬──────────┐
 INT-S_BASE  INT-T_BASE DISTR-ATTR_BASE INVAR-F_BASE
     │          │              │             │
   welch      +WPRO         .Attr        .Invar
```
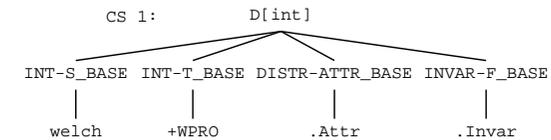
.Pro → D/Aquant   Expressions of type .Pro are either "transitive" or "intransitive". In the terms of our analysis, this means that the c-structure complement of the .Pro expression (the NP) is optional, i.e. the NP can be filled or empty.

Expressions of type .Pro represent the vast majority of the quantifying expressions in German. A subset of these expressions inflects like determiners, another subset inflects like quantifying adjectives, the third subset exhibits ambiguous inflection. Accordingly, .Pro expressions correspond to the category D, Aquant or to both categories, respectively. For instance, *irgendwelche* 'any' exhibits ambiguous inflection and can be of category D or Aquant, as sketched by the sublexical trees below.

```
                D[std]
         ┌──────────┬──────────┬──────────┐
     INDEF-S    INDEF-T  DISTR-PRO   INFL-F[det]
        │          │         │          │
  irgendwelche   +INDEF     .Pro      ...       .St
```

```
                          Aquant[std]
              ┌──────┬──────┴────────────┐
          INDEF-S  INDEF-T  DISTR-PRO      INFL-F[adj]
              │       │         │               │
        irgendwelche +INDEF    .Pro    ...      .St
```
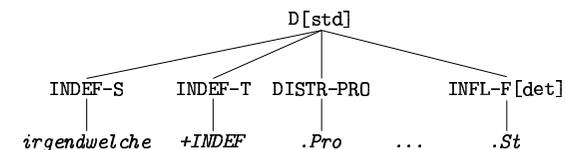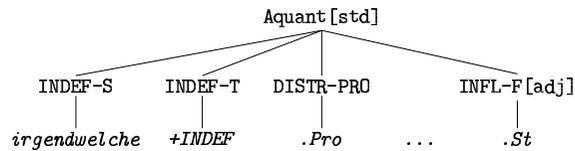
Many .Pro expressions show a genitive variant ending with *-n* instead of *-s* (in masculine and neuter singular) (Drosdowski 1995, p. 323,335), compare the regular genitive form in (244) and the variant in (245).

(244)   *dieses   Jahres*
        this[GEN]   year
        'of this year'

(245)   *diesen   Jahres*
        this[GEN]   year
        'of this year'

Such variants occur with demonstratives (see above), indefinites (246), and interrogatives (247).

(246)   *manchen   Jahres*
        some[GEN]   year
        'of some year'

(247)   *welchen   Jahres*
        which[GEN]   year
        'of which year'

We call the genitive variant "weak genitive", because the ending *-n* is characteristic of weak adjectives. The weak genitive seems to be restricted to determiner usage, hence it is marked by .Attr in the morphology. In addition, a special tag ^WEAKGEN indicates its marked status. Compare the morphological analyses of the ordinary genitive form *dieses* 'this' vs. the weak genitive form *diesen*.

```
M-Input    dieses  ('this')
M-Output   diese +DEM .Pro .MN .Gen .Sg .St
```

```
M-Input    diesen  ('this')
M-Output   diese ^WEAKGEN +DEM .Attr .MN .Gen .Sg .St
```

Note that only the weak genitive of the respective lemmas is marked .Attr whereas all other forms of the lemma are marked .Pro.

Besides the above examples of quantifying expressions, our morphology defines further similar expressions. However, these are exceptional in that they are not marked by the distribution tags .Attr, .Subst, or .Pro. These cases are (i) articles and (ii) personal pronouns.

Articles → D   The definite and indefinite article behave like .Attr expressions, without being marked by .Attr. Instead, articles are marked by the special tags .Def (definite) and .Indef (indefinite), respectively, which follow the part-of-speech tag +ART. (For the special lemma form (cf. p. 222).)

```
M-Input    des  ('the')
M-Output   die_art +ART .Def .MN .Gen .Sg .St
```

```
M-Input    ein  ('a')
M-Output   eine_art +ART .Indef .Masc .Nom .Sg .None
           eine_art +ART .Indef .Neut .NA .Sg .None
```

That means that our category D corresponds to two types of morphological tags: (i) expressions marked by .Attr, (ii) articles, marked by .Def or .Indef.

In our implementation, we chose to reinterpret the tags .Def and .Indef as equivalents of .Attr, rather than assuming extra rules for articles.

Personal pronouns → PRON   Personal pronouns behave like .Subst expressions, without being marked by .Subst. Instead, personal pronouns are marked by the tags .Pers (canonical pronouns), .Refl (reflexives), or .Rec (reciprocal), which follow the part-of-speech tag +PPRO, cf. the morphological analyses of *ich* 'I', *sich* 'himself/herself/itself/themselves', and *einander* 'each other' below. (Pronouns are not further discussed here.)

```
M-Input    ich  ('I')
M-Output   sie +PPRO .Pers .1 .Sg .MFN .Nom .None
```

```
M-Input    sich  ('himself/herself/itself/themselves')
M-Output   sie +PPRO .Refl .3 .Sg .MFN .DA .None
           sie +PPRO .Refl .3 .Pl .MFN .DA .None
```

```
M-Input    einander  ('each other')
M-Output   einander +PPRO .Rec .Invar
```

10.6.4   Inflection

Finally, all quantifying expressions are marked for inflection by the morphology. (A complete list of the morphology tags marking nominal inflection is given in the summary (cf. sec. 11.4).)

Tags for gender, case, and number follow the distributional tags. The last tag indicates the inflection type. Quantifying expressions are either .St (strong),

.Wk (weak), or .None (uninflected), cf. the morphological analyses of *dieses* 'this', *beiden* 'both', and *kein* 'no'.

```
M-Input    dieses  ('this')
M-Output   diese +DEM .Pro .Neut .NA .Sg .St
           diese +DEM .Pro .MN .Gen .Sg .St
```

```
M-Input    beiden  ('both')
M-Output   beide +INDEF .Pro .MFN .Dat .Pl .St
           beide +INDEF .Pro .MFN .NGDA .Pl .Wk
```

```
M-Input    kein  ('no')
M-Output   keine +INDEF .Attr .Neut .NA .Sg .None
           keine +INDEF .Attr .Masc .Nom .Sg .None
```

The tag .Invar marks expressions that are never inflected, e.g. *welch* 'which' (whereas .None marks expressions that are not inflected in certain cases, e.g. *kein* 'no'). Therefore, .Invar always occurs as the only inflectional tag, i.e. there are no gender, case, and number tags. Compare the morphological output of the invariant *welch* 'which' below and the uninflected *kein* 'no' above.

```
M-Input    welch  ('which')
M-Output   welch +WPRO .Attr .Invar
```

Personal pronouns such as *ich* 'I' are also marked as .None, since they are actually part of a paradigm that inflects for case: *mich/mir* 'me[ACC/DAT]'. Nevertheless, they do not have the typical strong/weak endings.

We now turn to idiosyncratic properties of quantifying expressions, which are not marked by morphological tags. Instead, lexicon entries encode these properties.

10.7   D/Aquant: Lexicon

The distributional tags .Attr and .Pro are not related to the inflectional properties of a given quantifying expression. That is, a .Attr or .Pro expression can in principle inflect like a determiner (D) or like a quantifying adjective (Aquant).

As we have seen above (cf. p. 178), some of the quantifying expressions, though, show a clear preference for the category D or Aquant (according to corpus data). Which category a quantifying expression belongs to is determined by checking the inflection of a following attributive adjective. This property clearly only shows up in the syntax. Consequently, the morphology does not provide

a tag for that. Instead the classification is done in the lexicon. We encode the idiosyncratic preferences in the lexicon entries of the concerned expressions. They interact with the sublexical rules via the sublexical stem categories.

Stem categories vary in two dimensions: (i) One dimension encodes different types of quantifying expressions (articles, possessives, etc.), which project different functions (SPEC DET, SPEC POSS, etc.) (sec. 10.7.1). (ii) The second dimension encodes D vs. Aquant preferences (sec. 10.7.2).

10.7.1   SPEC **Functions**

Each type of quantifying expression is assigned a different stem category, according to its part-of-speech classification: (i) articles: ART-S, (ii) possessives: POSS-S, (iii) demonstratives: DEM-S, (iv) indefinites: INDEF-S, (v) interrogatives: INT-S.

As a sample lexicon entry, see the entry for the demonstrative *selbe* 'same', with stem category DEM-S.

```
selbe      !DEM-S xle "EX: in selbem Umfang (to the same extent)"
                      "EX: die selbe Sprache (the same language)
                      (note: other DEM are determiners only)"
                      @(DEM %stem) @(AMBIG-INFL); ONLY.
```

The different stem categories (ART-S, POSS-S, etc.) allow each type to project different functions: (i) SPEC DET, (ii) SPEC POSS, (iii) SPEC DEM, (iv) SPEC QUANT, (v) SPEC INT, respectively.

The respective functions are introduced by different templates. For instance, the function SPEC DET is introduced by the template @DET, and the function SPEC DEM is introduced by the template @DEM, as illustrated by the entries of the article *die* 'the' (with the lemma form 'die_art') and the demonstrative *selbe* 'same' (see above), respectively.

```
die_art    !ART-S xle @(DET die) DieAsDet $ o::* "preferred"; ONLY.

DET(_stem) =
       (^SPEC DET PRED) = '_stem'.


DEM(_stem) =
       (^SPEC DEM PRED) = '_stem'.
```

The different functions become operative in the analysis of multiple quantifying expressions. As we will see (cf. sec. 10.9), most of the multiple quantifying expressions consist of two expressions of different type, e.g. an article plus a demonstrative as in (248), or an indefinite plus a possessive as in (249).

(248)   *das   selbe   Kind*
        the[ART] same[DEM] child

(249)   *all   ihre   Gedanken*
        all[INDEF] her[POSS] thoughts

Quantifying expressions are the functional head of the DP and are annotated by ↑=↓. In the case of multiple quantifying expressions, both of the quantifying expressions can be functional (co-)heads of the DP at the same time, without their f-structures clashing: the respective PRED features are embedded within the distinguishing SPEC-features DET, DEM, POSS, etc., cf. (250).

(250)   *das   selbe   Kind*
        the[ART] same[DEM] child

```
CS 1:      DP[std]
              |
           DPx[std]
          /    |     \
      D[std] Aquant[std] NP
        |      |          |
       das   selbe     N[comm]
                          |
                        Kind
```

"das selbe Kind"

$$\begin{bmatrix} \text{PRED 'Kind'} \\ \text{SPEC} \begin{bmatrix} \text{DEM} [\text{PRED 'selbe'}] \\ \text{DET} [\text{PRED 'die'}] \end{bmatrix} \end{bmatrix}$$
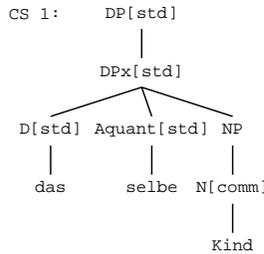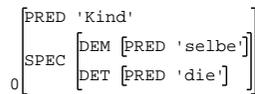
An alternative analysis would be to use a uniform stem category for all quantifying expressions, e.g. PRED-S (instead of differentiating stem categories ART-S, DEM-S, etc.), which would introduce the equation (↑PRED) = '%stem'. The sublexical rules would then introduce equations such as (↑SPEC DET) = ↓, (↑SPEC POSS) = ↓, etc., depending on the part-of-speech tag. Compare the fragments of the annotated sublexical trees below: the first corresponds to our implementation, the second illustrates the alternative analysis.

```
              D
           /  |   \
   ↑=↓
  ART-S      ART-T  ... ...
    |
(↑SPEC DET PRED)='%stem'
   die           +ART
```

```
            D
          /  |  \
(↑SPEC DET)=↓
  PRED-S     ART-T  ... ...
    |
(↑PRED)='%stem'
   die         +ART
```
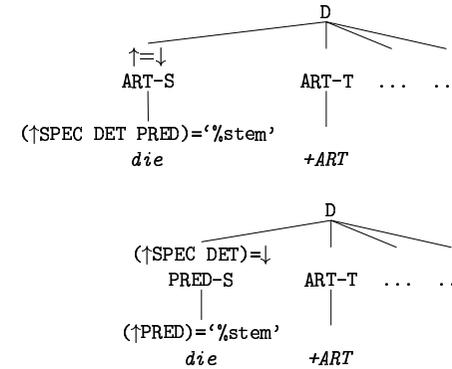
An advantage of this approach would be that one does not have to care about the stem category in each lexicon entry. The stem category would be uniformly represented by, e.g., PRED-S.

However, in our approach (with constraints such as (↑SPEC DET PRED) = '%stem', (↑SPEC POSS PRED) = '%stem', etc. introduced in the lexicon entry already), it is easier to state restrictions on multiple quantifying expressions or properties of the head noun etc., since one can refer to the DP's f-structure by ↑, without inside-out equations.

Another advantage is that full form entries (which do not make use of sublexical rules) can employ the same templates as lemma entries, cf. the entries of *welch* 'which' (of category INT-S) and the multi-word expression *was für* 'what a' (with a full form entry of category D[int]); both make use of the template @INT.

```
welch     !INT-S xle @(INT %stem) @(SPEC-COUNT);
          !Dpre-S xle @(INT %stem) @(PLUS-EINE); ONLY.


was' für  !D[int] * "EX: was für Leute (what kind of people)"
                    "EX: was für einer (what kind of person)"
                    @(INT %stem)
                    (^CHECK _SPEC-TYPE _DET) = attr
                    @(MWE);
          !Dpre[int] * "EX: was für ein Mann (what a man)"
                      @(INT %stem) @(PLUS-EINE) @(MWE);
          ONLY.


INT(_stem) =
       (^SPEC INT PRED) = '_stem'.
```

10.7.2   Categorial Preferences

Each of the different types of quantifying expressions can be further classified according to inflectional properties. We distinguish three cases: (i) the quantifying expression shows a clear preference for determiner inflection; (ii) it shows a preference for adjectival inflection; (iii) it allows for both types of inflection.

In our implementation, case (iii) corresponds to the stem categories as described above, e.g. indefinites are of category INDEF-S. Cases (i) and (ii) are encoded by the subclassified stem categories INDEF-S[det] and INDEF-S[adj], respectively.

For instance, compare the lexicon entries for the ambigous indefinite *einige* 'some' (category INDEF-S), the "determiner-indefinite" *irgendeine* 'someone' (INDEF-S[det]), and the "adjective-indefinite" *mehrere* 'some' (INDEF-S[adj]).

```
einige    !INDEF-S xle @(QUANT %stem)
                        @(NO-DEF) @(AMBIG-INFL); ONLY.

irgendeine !INDEF-S[det] xle @(QUANT %stem) @(SPEC-COUNT);
            =PRON-S xle "indefinite pronoun"; ONLY.

mehrere   !INDEF-S[adj] xle @(QUANT %stem) @(NO-DEF); ONLY.
```

The categorial information encoded in the stem category is linked to the actual inflectional ending by sublexical rules (cf. sec. 10.8). (The template @AMBIG-INFL is discussed below. For the templates @NO-DEF (cf. p. 262) and @SPEC-COUNT (cf. sec. 10.11.1).)

As mentioned above, the classification D vs. Aquant is based on corpus data: if 98% or more of the (unambiguous) instances are of one type, this type is chosen as the only possible one. Otherwise, both types are admitted.

- For instance, the quantifying expression *jene* 'that' is followed by weak adjectives in 98.9% of the unambiguous instances; hence, it is classified as a determiner (D).

- In contrast, *mehrere* 'some' is followed by strong adjectives in 98.0% of the instances; hence, it is classified as Aquant.

- Finally, *viele* 'much/many' is followed by strong adjectives only in 88.4% of the instances; hence, it is classified as ambiguous (D or Aquant).

In order to determine a reasonable threashold, we tried to get an idea of the frequency of typing errors. For this, we examined data involving the definite article *die* 'the'. The definite article obligatorily triggers weak adjectives in German. Hence, any

occurrence of a the definite article followed by a weak adjective can be interpreted as a typing error.

For our purpose, we looked at contracted determiners, such as *zum* 'to the', followed by attributive adjectives. (Non-contracted instances of the definite article are easily confounded with the relative pronoun.) We found 90,230 instances in the Frankfurter Rundschau, 99.9% of which exhibit the correct (= determiner) inflection, as in (251). 0.1% of the instances contain adjectives with incorrect inflection, as in (252). Hence taking 98% as the threshold seems to be quite restrictive.

(251)   *zum   normalen   Leben*
        to‿the[ST] normal[WK] life
        'to normal life'

(252)   (see corpus example DP10.3, Appendix A)
        **zum   gemütlichem   Beisammensein*
        to‿the[ST] comfortable[ST] together‿be
        'to a leisurely get-together'

We now turn to the question which corpus instances of a quantifying expression and an attributive adjective unambiguously indicate determiner vs. adjectival inflection.

(i) To indicate determiner inflection, the quantifying expression and the adjective must exhibit complementary inflection, e.g. *-r -e* (i.e. the quantifying expression ends with *-r*, the following attributive adjective with *-e*), as in *jeder gute Freund* 'any good friend'.

(ii) To indicate adjectival inflection, they must exhibit identical inflection, e.g. *-r -r*, as in *einiger alter Leute* 'some old people'.

(With certain forms, however, strong and weak adjectival inflection coincide (cf. the tables in the theoretical chapter (cf. p. 154)). For instance, *alte* 'old[FEM,SG,NOM,ST/WK]' represents a strong or weak feminine adjective. This means that the combination *-e -e*, as in *jede junge Frau* 'each young[ST/WK] woman', is ambiguous and, hence, cannot be taken as evidence of the categorial status of the quantifying expression.)

The following tables list unambiguous instances of determiner and adjectival inflection, respectively.

Inflection Pattern	Example
-r -e [MASC,SG,NOM]	*jeder gute Freund* 'any good friend'
-s -e [NEUT,SG,NOM]	*jedes kleine Kind* 'any young child'
-m -n [MASC/NEUT,SG,DAT]	*jedem kleinen Kind* 'any young child'
-r -n [FEM,SG,GEN/DAT] or [PL,GEN]	*jeder alten Frau* 'any old woman'

Table 10.1: Unambiguous instances of determiner inflection

Inflection Pattern	Example
-r -r [MASC,SG,NOM], [FEM,SG,GEN/DAT], or [PL,GEN]	*einiger alter Leute* 'some old people'
-s -s [NEUT,SG,NOM]	*einiges verschämtes Kichern* 'some bashful giggling'
-m -m [MASC/NEUT,SG,DAT]	*einigem verschämtem Kichern* 'some bashful giggling'

Table 10.2: Unambiguous instances of adjectival inflection

The tables below summarize the results we got from the FR corpus for a selection of quantifying expressions (only quantifying expressions with more than 50 unambiguous instances in the corpus were taken into account). The tables show the frequency of unambiguous instances for quantifying expressions; the first table lists expressions with predominantly determiner inflection, the second lists expressions with predominantly adjectival inflection.

	Relative Frequency	Absolute Frequency
die 'the'	99.9 %	90,230
jede 'each'	99.8 %	2,087
diese 'this'	99.7 %	4,324
jene 'that'	98.9 %	369
welche 'which'	96.7 %	91
alle 'all'	95.8 %	1,781
wenige 'few'	92.5 %	721
manche 'some'	79.3 %	119

Table 10.3: Expressions with predominantly determiner inflection (D)

	Relative Frequency	Absolute Frequency
mehrere 'some'	98.0 %	50
einige 'some'	92.1 %	129
andere 'other'	91.2 %	249
viele 'much/many'	88.4 %	169
solche 'such'	60.4 %	81

Table 10.4: Expressions with predominantly adjectival inflection (Aquant)

Below we list some of the "counterexamples", i.e. examples exhibiting the unusual, more marked inflection. (253), (254), (255), and (256) are at the margins of ungrammaticality, (257) and (258) are quite acceptable.

(253)   (see corpus example DP10.4, Appendix A)
(*) *bei jedem   mißglücktem Dribbling*
    on each[ST] bad[ST]   dribbling

(254)   (see corpus example DP10.5, Appendix A)
(*) *vor   diesem   wirtschaftlichem Hintergrund*
    against this[ST] economic[ST]   background

(255)   (see corpus example DP10.6, Appendix A)
(*) *mit jenem   spektakulärem Triumph*
    with that[ST] spectacular[ST] triumph

(256)   (see corpus example DP10.7, Appendix A)
(?) *laut   mehrerer   ärztlichen   Atteste*
    according several[ST] medical[WK] certificates
    'according to several medical certificates'

(257)   (see corpus example DP10.8, Appendix A)
einiges   verschämte   Kichern
some[ST] bashful[WK] giggling

(258)   (see corpus example DP10.9, Appendix A)
anderer   hessischen   Jugendzentren
other[ST] Hessian[WK] youth‿centres
'of other Hessian youth centres'

In our analysis, all quantifying expressions that show ambiguous inflection yield double analyses whenever they are used without any disambiguating attributive adjective. To avoid such unwarranted ambiguities, we introduce an OT preference mark `QuantAsDet`, via the template `@AMBIG-INFL`, preferring the determiner reading. As an example, see the entry of the ambiguous expression *einige* 'some'.

```
einige    !INDEF-S xle @(QUANT %stem)
                 @(NO-DEF) @(AMBIG-INFL); ONLY.


AMBIG-INFL =
     @(IF (^INFL) = strong-det
     QuantAsDet $ o::* "preferred"
     ).
```

Note that traditional grammars observe that the inflectional variance depends on case and number. For instance, *solche* 'such' sometimes inflects like a determiner but not in the cases of [SG,NOM/ACC] and [MASC/NEUT,GEN] (Helbig and Buscha 1993, p. 301) (cf. p. 175). Obviously our implementation does not model the variance in such detail; however, the factors that play a role in the observed variance are not yet understood.

Similarly, the default lexicon entry for quantifying expressions, encoded by the –Lunknown entry, makes use of the template @AMBIG-INFL, cf. the (manual) fragment of the –Lunknown entry below (only sublexical categories that are related to quantifying expressions are displayed).

```
 -Lunknown  DEM-S[det]  xle @(DEM %stem) @(SPEC-COUNT);
            INDEF-S     xle @(QUANT %stem) @(AMBIG-INFL);
            INT-S       xle @(INT %stem) @(AMBIG-INFL);
            ...
```

The –Lunknown entry defines ambiguous inflection for indefinites (INDEF-S) and interrogatives (INT-S), vs. only determiner inflection for demonstratives (DEM-S[det]). This decision is based on the above corpus frequency studies.

A default entry is practical since (i) it can be used for all indefinites, demonstratives, and interrogatives that do not exhibit idiosyncratic properties, and (ii) the default entry covers any quantifying expression that may be added to the morphology at a later stage, e.g. if adjectives like *weitere* 'further', *erstere* 'first', and *letztere* 'latter' are classified as indefinites in a next version of the morphological analyzer.

Articles always inflect like determiners, hence only one stem category is necessary for them: ART-S. Since there are only two instances of ART-S (*die* 'the' and *eine* 'a'), ART-S is not part of the default lexicon entry, –Lunknown. The same holds for possessives (with two instances of POSS-S, *ihre* 'her' and *ihrige* 'hers').

The –Lunknown entry should handle only those quantifying expressions that are not listed with their own idiosyncratic information in the lexicon. To prevent these expressions from matching –Lunknown, the specific entries contain the key word ONLY, thus overwriting the default entry –Lunknown. (Alternatively, one could add –INDEF-S, –INT-S, or –DEM-S[det] to subtract the default values while keeping ETC in the specific entries. The version with ONLY, however, keeps the specific entries simpler.)

```
mehrere   !INDEF-S[adj] xle @(QUANT %stem) @(NO-DEF); ONLY.
```

To avoid spurious ambiguities, invariant quantifying expressions are always dominated by D in our implementation. This is guaranteed by restricting the sublexical rule for Aquant to inflected forms. As a consequence, invariant indefinites may be irrespectively classified as INDEF-S or INDEF-S[det] in the lexicon.

In combination with sublexical and syntactic rules, the distributional and inflectional tags restrict the distribution of the quantifying expressions. This is the topic of the next section.

10.8   D/Aquant: Sublexical Rules

The basic picture we have drawn by now is as follows. We distinguish two types of quantifying expressions, determiners (D) and quantifying adjectives (Aquant). This syntactic distinction correlates with morphological and lexical distinctions.

- (i) Morphological distinction: Expressions marked .Attr are dominated by D, whereas .Pro is ambiguous and may be dominated by D or Aquant. (.Subst is always dominated by the category PRON.)

```
    .Attr ─────────────▶ D
    .Pro ─────────────▶ Aquant
```
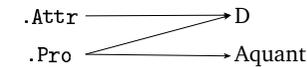
- (ii) Lexical distinction: Expressions that are morphologically marked by .Pro often exhibit idiosyncratic preferences for D or Aquant. These preferences are encoded by stem categories that directly correspond to the syntactic categories. For instance, the stem category INDEF-S[det] is always dominated by D, INDEF-S[adj] is always dominated by Aquant, and INDEF-S may be dominated by D or Aquant.

```
    INDEF-S[det] ───────────▶ D
    INDEF-S      ◀──
    INDEF-S[adj] ───────────▶ Aquant
```
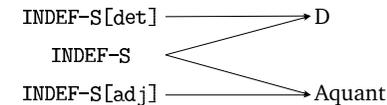
In the following sections, we present the rules by which the syntactic categories, the sublexical categories and the morphological tags are linked.

10.8.1   The Macro @QUANT-EXPRESSION

D and Aquant are both introduced by the @Dbar macro (for the template @RIGHT-SISTER (cf. sec. 10.10)).

```
Dbar(_type) =
    ...

    "EX: der Mann (the man)"
    "EX: alle Leute (all people)"
    "EX: welche beiden Frauen (which two women)"
```

```
        D[_type]: @(RIGHT-SISTER);
        Aquant[std]*

      | "EX: wieviele Frauen (how many women)"
        "EX: einige wenige Frauen (some few women)"
        Aquant[_type]
        Aquant[std]*
   ...
```

The sublexical rules for D and Aquant mostly overlap, hence we use a parametrized macro, @QUANT-EXPRESSION, to capture both.

```
D[_type] -->
        ...
      |
        @(QUANT-EXPRESSION det _type)
      | ...


Aquant[_type] -->
        ...
      |
        @(QUANT-EXPRESSION adj _type)
      | ...
```

The macro @QUANT-EXPRESSION serves two purposes.

- Type of quantifying expression: @QUANT-EXPRESSION covers all kinds of quantifying expressions in German: articles, demonstratives, possessives, indefinites, and interrogatives (represented by @QUANTart, @QUANTdemon, @QUANTposs, @QUANTindef, and @QUANTint, respectively).

- Category of the quantifying expression: @QUANT-EXPRESSION moreover covers both usages of quantifying expressions: D and Aquant. This is encoded by way of the first parameter _cat: _cat determines (i) the lexical stem category and (ii) the inflectional behaviour. With _cat = det (as in the D rule), we define determiners, with _cat = adj (as in the Aquant rule), we define quantifying adjectives.

The second parameter of @QUANT-EXPRESSION distinguishes non-interrogative vs. interrogative expansions (@QUANTart, @QUANTdemon, @QUANTposs, @QUANTindef for _type = std and @QUANTint for _type = int).

```
QUANT-EXPRESSION(_cat _type) =
        "used by quants (D and Aquant)"
        { "articles (inserted under DET)"
          e: _type = std;
          @(QUANTart _cat)

        | "demonstratives (inserted under DEM)"
          e: _type = std;
          @(QUANTdemon _cat)

        | "possessives (inserted under POSS)"
          e: _type = std;
          @(QUANTposs _cat)

        | "indefinites (inserted under QUANT)"
          e: _type = std;
          @(QUANTindef _cat)

        | "interrogatives (inserted under INT)"
          e: _type = int;
          @(QUANTint _cat)
        }.
```

@QUANTart, @QUANTdemon, @QUANTposs, @QUANTindef, and @QUANTint are macros as well, expanding to sequences of morphological tags (which encode the lemma form, part of speech, distribution, and inflection of the quantifying expression).

As an example, cf. the sublexical tree for the determiner *allerlei* 'various' in (259). This example is the result of several, stacked macro calls. First, the category D[std] calls the macro @QUANT-EXPRESSION, with the parameters _cat = det and _type = std. @QUANT-EXPRESSION in turn calls the macro @QUANTindef, which finally results in the displayed sublexical tree.

(259)   *allerlei*
         various

The distributional tags (.Attr and .Pro) and inflectional tags (agreement features, e.g. .Masc, .Invar) of all types of quantifying expressions are jointly defined in a common macro @DISTR-INFL. That is, the macros @QUANTart, @QUANTdemon, etc. all make use of the macro @DISTR-INFL. As an example, we show the definition of the macro @QUANTart. (The internal structure of the macros @QUANTart, @QUANTdemon, etc. differs only in minor points. These differences are addressed below (cf. sec. 10.8.3).)

```
QUANTart(_cat) =
      ART-S_BASE: _cat = det;
      ART-T_BASE
      @(DISTR-INFL _cat).
```

The macro @DISTR-INFL is discussed in detail in the next section.

10.8.2   The Macro @DISTR-INFL

The macro @DISTR-INFL builds the link between the morphological tags (encoding distribution and inflection) and the categorial status, in particular by restricting the parameter _cat. The first part of @DISTR-INFL covers the distributional tag (sec. 10.8.2.1); the second part covers the inflectional tags (sec. 10.8.2.2), cf. the schematic tree below.

10.8.2.1   Distributional Tags

We start by repeating the linguistic aspects of the distributional tags, accompanied by an outline of our implementation. A detailed presentation of the implementation is presented afterwards.

- Linguistic aspect: expressions marked by .Attr are obligatorily transitive, i.e. the sister NP may not be omitted.

Implementation: this restriction is triggered by a specific feature [CHECK _SPEC-TYPE _DET attr], which is associated with .Attr expressions.

- Linguistic aspect (observation): in addition, expressions marked by .Attr are always dominated by D. (All .Attr expressions come out as determiners according to our inflection-based criterion for determiners (cf. p. 225).)

  Implementation: this restriction is encoded by a specific sublexical category of .Attr, which may only be dominated by D.

- No such constraints are set on .Pro expressions.

The following picture outlines the relation between morphological tags, sublexical categories, and syntactic categories.

Implementation details   The distributional tags .Attr and .Pro. are dominated by the sublexical categories DISTR-ATTR and DISTR-PRO, respectively.

```
.Attr     DISTR-ATTR xle; ONLY.

.Pro      DISTR-PRO xle; ONLY.
```

These sublexical categories are introduced in the first part of the parametrized macro @DISTR-INFL. (The parameter _cat = predet encodes so-called predeterminers, which are discussed below (cf. sec. 10.9.1).)

```
DISTR-INFL(_cat) =
      ...
      "_cat = det/predet"
      DISTR-ATTR_BASE
      e: { _cat = det
          (^CHECK _SPEC-TYPE _DET) = attr
        | "special disjunct for predeterminers:
          don't introduce CHECK feature here
          - predeterminers marked by .Attr automatically have
          right sister (namely D)
          - CHECK feature of predet would unify with right
          sister D, which would then need a right sister
          itself; this would exclude:"
          "EX: Manch einer kam. (Some people came.)
          (since 'einer' would need right sister due to
```

```
                    requirements of 'manch')"
                _cat = predet };


    | "_cat = det/predet/adj"
      DISTR-PRO_BASE
    }
    ...
```

- The sublexical category DISTR-ATTR (and, hence, the tag .Attr) is only admissible with the parameter _cat = det. The parameter _cat = det is set by determiners (D) only. Hence, .Attr expressions are always dominated by D.

  (The feature [CHECK _SPEC-TYPE _DET attr] prohibits the omission of the sister NP (cf. sec. 10.10).)

- In contrast, .Pro (DISTR-PRO) is compatible with all values of _cat. Hence, .Pro expressions may be dominated by D or Aquant.

As noted above, articles always belong to the category D (cf. sec. 10.7.2). The morphological component, however, does not mark them by .Attr but assigns the tags .Def (definite) and .Indef (indefinite). We interpret these tags as an alternative representation of .Attr. That is, .Def, .Indef, and .Attr belong to the same sublexical category, DISTR-ATTR, and are thus uniformly represented to the outside context.

```
.Def      DISTR-ATTR xle @(DET-TYPE def) @(SPEC-DEF); ONLY.
.Indef    DISTR-ATTR xle @(DET-TYPE indef); ONLY.
```

10.8.2.2 Inflectional Tags

Following the distributional tags, morphology specifies inflectional tags. For inflected forms these consist of tags marking gender, case, number, and inflection type.

In contrast to inflection-type tags, the tags of gender, case, and number are not related to the distinction D vs. Aquant. We therefore first present the documentation of the tags marking the inflection type.

We start with an outline of the linguistic aspects of the inflection-type tags and their implementation.

- Linguistic aspect: expressions of category D on the one hand, and expressions of category Aquant or A on the other exhibit complementary

inflection. For instance, expressions of both types cannot be marked .St simultaneously.

Implementation: this restriction is encoded by the feature INFL, with values strong-det and strong-adj that are related to the categorial status, via the sublexical categories.

- Linguistic aspect: only expressions of category Aquant or A may exhibit weak inflection.

  Implementation: the sublexical category assigned to .Wk is related to Aquant or A only.

- Linguistic aspect (observation): only expressions of category D may be unmarked for inflection type, i.e. marked by .None. (.None expressions are marked for gender, case, and number, though.) (Similarly to .Attr expressions (cf. p. 225), all expressions that are marked by .None (in certain forms) come out as determiners according to our inflection-based criterion for determiners.)

  Implementation: similar to what was done above, the sublexical category assigned to .None is related to D only.

- Linguistic aspect: expressions of category D, Aquant, or A may be invariant, i.e. marked by .Invar.

  Implementation: in contrast to the linguistic analysis, we exclude uninflected Aquant, to avoid spurious ambiguities. This is achieved by restricting the sublexical rules of Aquant.

The following picture outlines the relation between morphological tags, sublexical categories, and syntactic categories. (The dotted line of .Invar indicates that .Invar is related to A but not Aquant.)
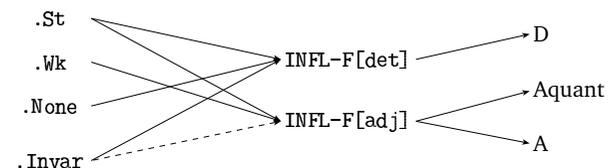
Implementation details   (For example phrases, see the previous chapter (cf. sec. 9.2.2).)

- The tag .St may be associated with determiners (D) or adjectives (Aquant or A), via the parametrized sublexical categories INFL-F[det] and INFL-F[adj], respectively. Strong determiners introduce the equation (↑INFL) = strong-det (associated with the sublexical category INFL-F[det]).

  In contrast, strong quantifying and attributive adjectives specify (↑INFL) = strong-adj (associated with the sublexical category INFL-F[adj]). Hence, strong determiners and adjectives cannot cooccur.

  ```
  "pronominal/strong features"
  .St      INFL-F[det] xle @(INFL-NOMINAL strong-det);
           INFL-F[adj] xle @(INFL-NOMINAL strong-adj);
           INFL-F xle "pronouns";
           ONLY.


  INFL-NOMINAL(_infl) =
           "for det/adj inflection"
           (^INFL) = _infl.
  ```

- The tag .Wk is only assigned the sublexical category INFL-F[adj]. Weak quantifying and attributive adjectives define the constraint (↑INFL) =c strong-det. Therefore, they can only occur with a strong determiner (which provides the required feature).

  ```
  "nominal/weak features"
  .Wk      INFL-F[adj] xle "note: there are no weak determiners
                            ('ein' is .None, 'viel' is .Invar)"
                            ~~@(INFL-NOMINAL strong-det); ONLY.
  ```

  Note that a double-negated equation ('~~') is equivalent to the corresponding constraining equation (cf. p. 57). Hence, the template call of @INFL-NOMINAL within the entry of .Wk is equivalent to the constraining equation (↑INFL) =c strong-det.

- Uninflected forms, marked by .None, are only assigned the sublexical category INFL-F[det]. They do not specify any constraints.

  ```
  .None    INFL-F[det] xle;
           INFL-F xle "personal pronouns";
           ONLY.
  ```

- Finally, for invariant forms, there is a single tag indicating the absence of any inflection, .Invar. No constraint on GEND, CASE, etc., is specified. (The entry of the sublexical category A-INVAR-F captures invariant adjectives, such as *lila* 'purple'. Only adjective-specific constraints are introduced here.)

  ```
  .Invar    A-INVAR-F xle @(AMORPH _AHEAD base_)
                          @(DEFAULT-ADEGREE);
            INVAR-F xle; ONLY.
  ```

The above sublexical categories for inflectional tags are introduced in the second part of the macro @DISTR-INFL.

```
DISTR-INFL(_cat) =
     ...

     { "gender, case, number"
       @(GCN)
       "inflection type"
       { INFL-F_BASE[det]: { _cat = det
                             | _cat = predet };
       | INFL-F_BASE[adj]: _cat = adj; }

     | "invariant"
       INVAR-F_BASE: { _cat = det
                       | _cat = predet };
     }
     ...
```

The gender, case, and number tags are covered by the macro @GCN.

```
GCN =
     GEND-F_BASE
     CASE-F_BASE
     NUM-F_BASE.
```

Just as with distributional tags (cf. sec. 10.8.2.1), the properties of the inflection-type tags vary depending on the parameter _cat.

- The parameter _cat = det allows for the sublexical categories INFL-F[det] and INVAR-F. (For _cat = predet (cf. sec. 10.9.1).)

- The parameter _cat = adj only allows for INFL-F[adj].

For invariant quantifying expressions, there are in principle always two possible analyses: D or Aquant. In an example such as (260), *viel* 'much' may be analyzed as a determiner. Since it does not inflect, the adjective *unnütz* 'superfluous' must exhibit strong inflection.

(260)   *viel*   *unnützes*   *Gerede*
         much[UNINFL] superfluous[ST] rumour

If, alternatively, *viel* is analyzed as a quantifying adjective, then again the adjective *unnütz* must be strongly inflected since no determiner is present. To avoid these otherwise identical analyses, we arbitrarily disallow invariant quantifying adjectives (by restricting the sublexical category INVAR-F to _cat = det/predet, in the definition of @DISTR-INFL).

We now turn to the remaining inflection tags, marking (i) gender, (ii) case, and (iii) number. We show selected lexicon entries of such tags.

Gender   The example entries of gender tags illustrate the analysis of underspecified tags in our implementation. Tags such as .MN (masculine, neuter) introduce negative constraints (e.g. ($\uparrow$GEND) $\tilde{}=$ fem) rather than disjunctive positive equations, enumerating the possible values (as in { ($\uparrow$GEND) = masc | ($\uparrow$GEND) = neut }).

```
.Masc       GEND-F xle @(GEND masc); ONLY.

.MN         GEND-F xle @(GEND-NO fem); ONLY.

GEND(_gend) = @(GEND_desig ^ _gend).


GEND-NO(_gend) = ~@(GEND_desig ^ _gend).


GEND_desig(_desig _gend) = (_desig GEND) = _gend.
```

This approach reduces the number of analyses of certain sentences that only differ with regard to gender. For instance, sentences containing the two-ways ambiguous pronoun *ihm* 'he[MASC]/it[NEUT]' would receive twice as many analyses otherwise.

Note, however, that in our approach, the f-structures of such sentences or phrases do not exhibit the feature GEND at all (since negative constraints are usually not displayed), cf. the f-structure analysis of (261).

(261)   *ihm*
         him[MASC]/it[NEUT]

```
"ihm"
```

$$
0 \begin{bmatrix} \text{PRED} & \text{'pro'} \\ \text{NTYPE} & [\text{NSYN pronoun}] \\ \text{CASE dat, NUM sg, PERS 3, PRON-FORM sie, PRON-TYPE pers} \end{bmatrix}
$$

A way of making the underspecification explicit is the use of features with values represented by so-called "closed sets" (Dalrymple and Kaplan 2000). The feature GEND then would have the complex value { masc neut }. This being a closed set, no further elements (such as fem) might be added.

Case   As with gender tags, there are underspecified case tags, e.g. .NGDA (which does not introduce any f-structure constraint).

```
.Nom        CASE-F xle @(CASE nom); ONLY.

.NGDA       CASE-F xle "noun, adj/quant"; ONLY.
```

Number   Number is specified by two different tags, .Sg or .Pl.

```
.Sg         NUM-F xle @(NUM sg); ONLY.
.Pl         NUM-F xle @(NUM pl); ONLY.
```

Up to now, we have discussed the distribution and inflection of quantifying expressions. All types of quantifying expressions (such as demonstratives or interrogatives) behave alike in this respect, hence they all make use of the general macro @DISTR-INFL.

We now turn to distinguishing properties of the different types of quantifying expressions, such as different stem categories DEM-S, INT-S, etc.

10.8.3   The Macros @QUANTart, @QUANTdemon, etc.

The general macro @QUANT-EXPRESSION covers all kinds of quantifying expressions in German: articles, demonstratives, possessives, indefinites, and interrogatives (represented by @QUANTart, @QUANTdemon, @QUANTposs, @QUANTindef, and @QUANTint, respectively) (cf. sec. 10.8.1).

The macros @QUANTart, @QUANTdemon, etc. cover the stem category, the part-of-speech category, and the sublexical categories of distribution and inflection (encoded by @DISTR-INFL), cf. the schematic tree below.

```
                      @QUANTart/demon/...
            ┌──────────────┼──────────────────────┐
          Stem       Part of Speech           @DISTR-INFL
                                           ┌─────────────┴────────┐
          ART-S            ART-T
          DEM-S            DEM-T        DISTR-ATTR/-PRO    INFL-F[_cat]
          INDEF-S          INDEF-T
           ...              ...
```
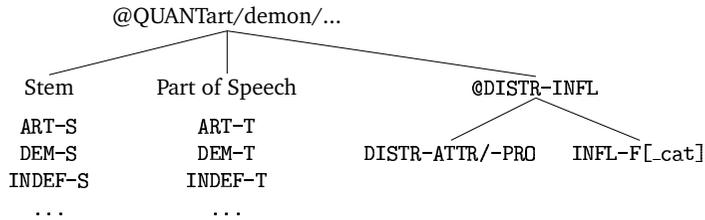
Basic differences between the different types of quantifying expressions are confined to the categories of stem and part of speech. In addition, certain quantifying expression allow for an alternative genitive form ("weak genitive"). These differences are addressed in this section.

(Note that this section focuses on differences that relate to sublexical rules and categories. Differences between the corresponding lexicon entries have been addressed above (cf. sec. 10.7).)

Stem categories   Each kind of quantifying expression is related to a specific stem category, e.g. @QUANTindef introduces INDEF-S, @QUANTdemon introduces DEM-S, etc.

As an example, cf. the definition of @QUANTindef. (For the category WEAKGEN-F, see below.)

```
QUANTindef(_cat) =
      @(INDEF-STEM _cat)
      ( WEAKGEN-F_BASE: _cat = det; )
      INDEF-T_BASE
      @(DISTR-INFL _cat).
```

The stem categories are covered by the parametrized macro @INDEF-STEM.

```
INDEF-STEM(_cat) =
      { INDEF-S_BASE[_cat] | INDEF-S_BASE }.
```

The parametrized category INDEF-S[_cat] encodes pure determiner (D) and pure adjectival (Aquant) readings, by an appropriate setting of the parameter _cat. For instance, the instantiated category INDEF-S[det] may dominate the lemma tag of the quantifying expression *irgendeine* 'someone', according to the specification in the lexicon entry of *irgendeine*.

```
irgendeine !INDEF-S[det] xle @(QUANT %stem) @(SPEC-COUNT);
          =PRON-S xle "indefinite pronoun"; ONLY.
```

As an alternative, the sublexical category INDEF-S provides the stem of those indefinites that allow for ambiguous inflection (D and Aquant). For instance, the ambiguous *einige* 'some' is associated with the category INDEF-S, cf. the lexicon entry below.

```
einige    !INDEF-S xle @(QUANT %stem)
                       @(NO-DEF) @(AMBIG-INFL); ONLY.
```

Hence, if @QUANTindef is called with _cat = det (which indicates that it has been called by the category D), the stem can either be represented by a lemma of category INDEF-S[det], which meets the restrictions on _cat (first disjunct). Or else, it is represented by an ambiguous lemma, of category INDEF-S (second disjunct), cf. the following figure.

```
      INDEF-S[det] ──────────────→ D
         INDEF-S   ───────╲   ╱
                           ╲ ╱
      INDEF-S[adj] ─────────╳──────→ Aquant
```
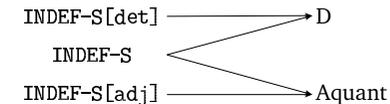
(Whether a quantifying expressions is classified as ambiguous or as a pure determiner or quantifying adjective depends on corpus data, as described above (cf. sec. 10.7.2).)

For other types of quantifying expression, we apply the same principle. (For articles and possessives, there is no need for an ambiguous category ART-S or POSS-S, respectively, since there are no ambiguous instances.)

Part-of-speech categories   Each type of quantifying expression is marked by its corresponding part-of-speech category. The sublexical macros accordingly expand to respective sublexical categories: e.g. @QUANTart introduces ART-T, @QUANTindef introduces INDEF-T, etc. These categories in turn dominate the morphological tags +ART, +INDEF, etc.

"Weak" genitive   With the exception of articles and possessives, all quantifying expressions allow for an alternative genitive form, e.g. *diesen* 'this[GEN]', marked by a special tag ^WEAKGEN by the morphology (cf. p. 226).

```
M-Input    diesen  ('this')
M-Output   diese ^WEAKGEN +DEM .Attr .MN .Gen .Sg .St
```

We associate these "weak" genitive readings with a dispreference mark, WeakGen.

```
^WEAKGEN  WEAKGEN-F xle WeakGen $ o::* "dispreferred"; ONLY.
```

Its sublexical category, WEAKGEN-F, is introduced optionally by the macros @QUANTindef, @QUANTdemon, etc. It is confined to determiner usage (_cat = det) (cf. p. 226).

```
QUANTdemon(_cat) =
      @(DEM-STEM _cat)
      ( WEAKGEN-F_BASE: _cat = det; )
      DEM-T_BASE
      @(DISTR-INFL _cat).
```

Certain possessives are capitalized in the polite form, e.g. *Dein* 'your[SG]'. These possessives are marked by a special tag ^CAP by the morphology. Moreover, additional tags mark features of the addressee, such as second person singular ('^.2 ^.Sg').

```
 M-Input     Dein   ('your')
 M-Output    ihre_attr ^.2 ^.Sg ^.MFN ^CAP +POSS .Attr .Masc .Nom .Sg
             .None
             ihre_attr ^.2 ^.Sg ^.MFN ^CAP +POSS .Attr .Neut .NA .Sg
             .None
```

The corresponding macro, @QUANTposs, optionally allows for these additional tags. The functional information associated with these tags is encoded by features CHECK _MORPH _CAPITAL and REFERENT (for the addressee), cf. the f-structure embedded under the function SPEC POSS in the analysis of *Dein* 'your'.

```
           "Dein"

        ┌                                                        ┐
        │CHECK  [_SPEC-TYPE [_COUNT +, _DEF +, _DET attr]]        │
        │                                                        │
        │          ┌                                          ┐  │
        │          │PRED       'pro'                          │  │
        │          │CHECK      [_MORPH [_CAPITAL +]]          │  │
        │SPEC  POSS│PRON-FORM  ihre                           │  │
        │          │       ┌REFERENT -2[NUM sg, PERS 2]  ┐    │  │
        │          │    -1 └                             ┘    │  │
        │0 CASE nom, GEND masc, NUM sg                           │
        └                                                        ┘
```

10.8.4   Summary

The macro @QUANT-EXPRESSION covers all kinds of quantifying expressions: articles, demonstratives, possessives, indefinites, and interrogatives.

Moreover, with the parameter _cat, @QUANT-EXPRESSION encodes the categorial status of these expressions: determiner (D) or quantifying adjectives (Aquant).

The parameter _cat percolates through a chain of macros, which connect the syntactic categories D and Aquant with the underlying morphological tag sequences (including the lemma), provided by morphology. That is, the parameter _cat restricts the co-occurrences of the syntactic categories D/Aquant and the following sublexical categories:

- The stem categories
  - ART-S
  - DEM-S[det], DEM-S
  - INDEF-S[det], INDEF-S[adj], INDEF-S
  - INT-S[det], INT-S[adj], INT-S
  - etc.

- The distributional categories, DISTR-ATTR and DISTR-PRO

- The inflection-type categories, INFL-F[det] and INFL-F[adj]

For instance, via the parameter _cat, the sublexical rules exclude co-occurrence of the stem category INDEF-S[det] and the inflection-type category INFL-F[adj].

As a valid example, compare the outline of the sublexical tree of *mehrere* 'some', analyzed as Aquant.

Note that the sublexical rules or the value of the parameter _cat do not preclude any morphological form of a quantifying expression (analyzed by the current version of the morphological component). That is, all sequences of morphological tags which represent the morphological analysis of a quantifying expression are captured by the sublexical rules.

@QUANT-EXPRESSION encodes various interdependencies between syntactic and morphological categories of a quantifying expression. In contrast, the feature INFL serves to restrict the inflection types of a quantifying expression and an adjacent adjective. The restrictions are encoded by means of different values of INFL and by defining and constraining equations involving INFL.

Our approach also accounts for multiple quantifying expressions. This is the topic of the next section.

10.9   D/Aquant: Multiple Quantifying Expressions

In the theoretical introduction, we mentioned that some quantifying expressions can occur simultaneously with certain others (sec. 76), as in (262), (263).

(262) *solch   ein   netter   Empfang*
       such[UNINFL] a[UNINFL] nice[ST] welcome

```
CS 1:          DP[std]
                  |
               DPx[std]
              /        \
          D[std]        NP
          /    \        /    \
   Dpre[std] D[std] AP[std,+infl] N[comm]
      |        |      |              |
    solch     ein  APx[std,+infl] Empfang
                        |
                    A[+infl]
                        |
                     netter
```

"solch ein netter Empfang"

```
PRED      'Empfang'
                 ⎡ PRED   'nett<[0:Empfang]>'           ⎤
                 ⎢ SUBJ   [0:Empfang]                   ⎥
                 ⎢        ⎡ AMORPH ⎡AHEAD base⎤       ⎤ ⎥
ADJUNCT  {       ⎢ CHECK  ⎢                          ⎥ ⎥ }
                 ⎢        ⎣ MORPH  ⎣CAPITAL -⎦       ⎦ ⎥
              69 ⎣ ADEGREE base_, ATYPE attributive    ⎦
          ⎡ NCONSTR   ⎡ADJ-ATTR +⎤                     ⎤
CHECK     ⎢                                            ⎥
          ⎣ SPEC-TYPE ⎣COUNT +, _DET attr⎦             ⎦
NSEM      [COMMON count]
NTYPE     [NSYN common]
          ⎡      ⎡ PRED      'eine'   ⎤                 ⎤
          ⎢ DET  ⎣ DET-TYPE indef    ⎦                 ⎥
SPEC      ⎢                                            ⎥
          ⎣ QUANT [PRED 'solch']                       ⎦
        0 CASE nom, GEND masc, INFL strong-adj, NUM sg, PERS 3
```

(263) *ein   solcher   netter   Empfang*
       a[UNINFL] such[ST] nice[ST] welcome
       'such a nice welcome'

```
CS 1:         DP[std]
                 |
              DPx[std]
             /    |    \
       D[std] Aquant[std]  NP
         |        |       /    \
        ein    solcher AP[std,+infl] N[comm]
                          |
                      APx[std,+infl] Empfang
                          |
                      A[+infl]
                          |
                       netter
```

"ein solcher netter Empfang"

```
PRED      'Empfang'
                 ⎡ PRED   'nett<[0:Empfang]>'           ⎤
                 ⎢ SUBJ   [0:Empfang]                   ⎥
                 ⎢        ⎡ AMORPH ⎡AHEAD base⎤       ⎤ ⎥
ADJUNCT  {       ⎢ CHECK  ⎢                          ⎥ ⎥ }
                 ⎢        ⎣ MORPH  ⎣CAPITAL -⎦       ⎦ ⎥
              91 ⎣ ADEGREE base_, ATYPE attributive    ⎦
          ⎡ NCONSTR   ⎡ADJ-ATTR +⎤                     ⎤
CHECK     ⎢                                            ⎥
          ⎣ SPEC-TYPE ⎣COUNT +, _DET attr⎦             ⎦
NSEM      [COMMON count]
NTYPE     [NSYN common]
          ⎡ AQUANT {-1[PRED 'solche']}                 ⎤
SPEC      ⎢      ⎡ PRED      'eine'   ⎤                 ⎥
          ⎣ DET  ⎣ DET-TYPE indef    ⎦                 ⎦
        0 CASE nom, GEND masc, INFL strong-adj, NUM sg, PERS 3
```

The inflectional patterns indicate that in the first example there are two determiners, whereas in the second example there is one determiner plus one quantifying adjective. (Note that the above f-structures are identical except for the feature INFL.) We describe both types of multiple quantifying expressions in the following sections.

An overview of all co-occurrences of inflection-types, as allowed by the c-structure rules and the constraints on INFL, is given in the table below. (Instances of these patterns are given below (cf. sec. 10.9.4).)

We make use of the XLE notation of regular expressions to denote the combinations of quantifying expressions and adjectives (cf. p. 52). (Note, howev-

er, that additional, lexical restrictions effectively exclude certain combinations, such as D[ST] D[UNINFL]; see below.)

(Remember that uninflected quantifying expressions do not project Aquants in our implementation to prohibit spurious ambiguities (cf. sec. 10.8.2.2).)

Determiners	Quantifying Adjectives	Attributive Adjectives
[D[ST]+ , D[UNINFL]*]	Aquant[WK]*	{ A[WK]	A[UNINFL] }*
D[UNINFL]*	Aquant[ST]*	{ A[ST]	A[UNINFL] }*

Table 10.5: Inflection of multiple quantifying expressions and adjectives

Only certain (probably semantically restricted) combinations of multiple de-terminers or determiners plus quantifying adjectives are grammatical in Ger-man.

To avoid massive overgeneration, only quantifying expressions that are lex-ically assigned a specific sublexical category, Dpre-S ("predeterminers"), may precede other determiners in our implementation (i.e. the class of predeter-miners is restricted c-structurally). In addition, many combinations are ruled out by f-structure restrictions.

The following sections address restrictions on (i) multiple determiners (sec. 10.9.1), (ii) co-occurrences of determiners and quantifying adjectives (sec. 10.9.2), and (iii) multiple quantifying adjectives (sec. 10.9.3).

10.9.1 Dpre + D

Quantifying expressions can consist of two adjacent determiners. Predetermin-ers (Dpre) are adjoined to determiners (D), as in (264).

(264) *alle   diese   Fragen*
       all[ST] these[ST] questions

```
CS 1:      DP[std]
              |
            DPx[std]
            /        \
        D[std]        NP
        /    \         |
  Dpre[std] D[std]  N[comm]
      |        |       |
    alle     diese   Fragen
```
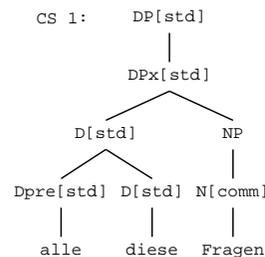
```
D[_type] -->
    ...
    | "predeterminer (added recursively)"
      "EX: all die vielen Leute
      (all these many people)"
      Dpre[_type]
      D[std]
      "first det determines DP type
      EX: welch ein Kind (what a child)"
    | ...
```

The type of the DP (standard, interrogative, or relative) is always deter-mined by the first quantifying expression, i.e. the predeterminer. For instance, an interrogative DP may consist of an interrogative predeterminer followed by an ordinary (standard) determiner (265).

(265) *welch   einem Unsinn*
       what[UNINFL] a[ST] nonsense
       'what nonsense'

```
CS 1:      DP[int]
              |
            DPx[int]
            /        \
        D[int]        NP
        /    \         |
  Dpre[int] D[std]  N[comm]
      |        |       |
    welch    einem   Unsinn
```
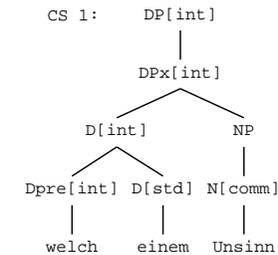
The predeterminer(s) are represented by all kinds of quantifying expressions: articles, possessives, demonstratives, indefinites (266), or interrogatives (267) (*was für* 'what' is analyzed as a multiword determiner). (For more examples (cf. sec. 10.9.4).)

(266) *alle   ihre   Gedanken*
       all[ST] her[ST] thoughts

(267) *was für   ein   netter   Empfang*
       [what for UNINFL] a[UNINFL] nice[ST] welcome
       'what a nice welcome'

To account for the different types of quantifying expressions, the sublexical rule of Dpre comprises a disjunction of all correspondings part-of-speech categories, such as ART-T, DEM-T, etc.

```
Dpre[_type] -->
        "listed in dp.lex.lfg"
        e: _type ~= rel;
        Dpre-S_BASE

      { "EX: ein jeder (everybody)"
        ART-T_BASE: _type = std;

      | "EX: ihr meistes Geld (most of her money)"
        @(PNG (^SPEC POSS REFERENT))
        ( CAP-DERIV-F_BASE: (^SPEC POSS) = ! )
        POSS-T_BASE: _type = std;

      | "EX: diese vielen Leute (these many people)"
        ( WEAKGEN-F_BASE )
        DEM-T_BASE: _type = std;

      | "EX: alle diese Leute (all these people)"
        ( WEAKGEN-F_BASE )
        INDEF-T_BASE: _type = std;

      | "EX: welch eine Frau (what a woman)"
        ( WEAKGEN-F_BASE )
        INT-T_BASE: _type = int;
      }

      @(DISTR-INFL predet).
```

Predeterminers are listed in the lexicon, to avoid overgeneration. They encode which determiners they want to combine with. There are two different types of predeterminer-determiner combination: (i) *alle/all* 'all' plus a determiner marking definiteness; (ii) certain predeterminers plus the indefinite article *eine* 'a'.

(i) *alle/all* 'all' plus a determiner marking definiteness The predeterminer *alle* 'all' obligatorily precedes definite specifiers. It thus calls a template @PLUS-DEF, requiring the presence of the feature CHECK _SPEC-TYPE _DEF, which encodes definiteness, cf. the entry of category Dpre-S. (For the template @SPEC-MASS (cf. sec. 10.11.1).)

```
alle      !INDEF-S xle @(QUANT %stem) @(SPEC-MASS)
                       @(NO-DEF) @(AMBIG-INFL);
          !Dpre-S xle @(QUANT %stem) @(SPEC-MASS)
                       @(PLUS-DEF); ONLY.

PLUS-DEF =
          ...
        | "predet + a quantifying expressions marked as definite"
          "EX: alle die Menschen, die vor dem Krieg flüchten, "
          "(all these people fleeing from the war)"
          "EX: alle diese Dinge (all these things)"
          "EX: alle meine Kollegen (all my colleagues)"
          (^CHECK _SPEC-TYPE _DEF)
        | ...
```

The definiteness feature CHECK _SPEC-TYPE _DEF is introduced by the definite article, possessives, and demonstratives. (In addition, prenominal genitives mark definiteness.)

They all make use of the template @SPEC-DEF to introduce the definiteness feature, cf. the example lexicon entry of the morphological tag .Def, which marks the definite article.

```
SPEC-DEF =
          (^CHECK _SPEC-TYPE _DEF) = +.

.Def      DISTR-ATTR xle @(DET-TYPE def) @(SPEC-DEF); ONLY.
```

As an example, cf. the f-structure analysis of (268).

(268)   *alle   diese   Fragen*
         all[ST] these[ST] questions

$$
\begin{array}{l}
\text{"alle diese Fragen"}\\
\left[
\begin{array}{ll}
\text{PRED} & \text{'Frage'}\\
\text{CHECK} & \left[\text{_SPEC-TYPE} \left[\text{_COUNT +, _DEF +}\right]\right]\\
\text{NTYPE} & \left[\text{NSYN common}\right]\\
\text{SPEC} & \left[\begin{array}{ll}\text{DEM} & \left[\text{PRED 'diese'}\right]\\ \text{QUANT} & \left[\text{PRED 'alle'}\right]\end{array}\right]\\
\text{GEND fem, INFL strong-det, NUM pl, PERS 3}
\end{array}
\right]_0
\end{array}
$$

alle 'all' and *beide* 'both' are both marked as indefinites by the morphology. In examples such as (269), their f-structures would therefore both end up under SPEC QUANT and would clash. Therefore, we treat *beide* as a cardinal and introduce it under the feature SPEC NUMBER.

(269)   *alle*   *beide*
        all[ST] both[ST]
        'both'

```
CS 1:      DP[std]
              |
           DPx[std]
              |
            D[std]
             / \
    Dpre[std] D[std]
        |        |
       alle    beide
```

```
"alle beide"

  ⎡PRED 'pro'                      ⎤
  ⎢      ⎡NUMBER ⎡PRED 'beide'⎤⎤   ⎥
  ⎢SPEC  ⎢                    ⎥⎥   ⎥
 0⎣      ⎣QUANT  ⎡PRED 'alle' ⎤⎦⎦  ⎦
```
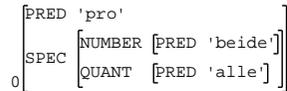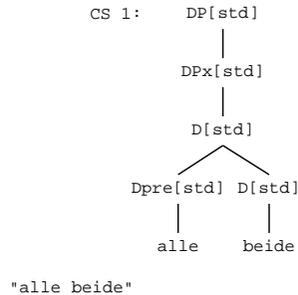
Predeterminers of category Dpre may also precede pronouns, cf. (270).

(270)   *all dessen*
        all this[GEN]
        'that all'

```
CS 1:      DP[std]
              |
           DPx[std]
              |
           PRON[std]
             / \
   Dpre[std] PRON[std]
       |         |
      all      dessen
```
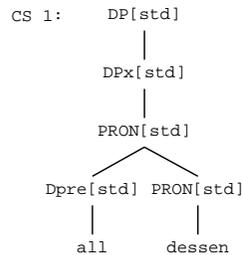
```
"all dessen"

  ⎡PRED  'pro'                                            ⎤
  ⎢CHECK ⎡_NMORPH ⎡_GENITIVE +⎤⎤                          ⎥
  ⎢NSEM  ⎡COMMON mass⎤                                    ⎥
  ⎢NTYPE ⎡NSYN pronoun⎤                                   ⎥
  ⎢SPEC  ⎡QUANT ⎡PRED 'all'⎤⎤                             ⎥
 0⎣CASE gen, NUM sg, PERS 3, PRON-FORM die, PRON-TYPE demon⎦
```
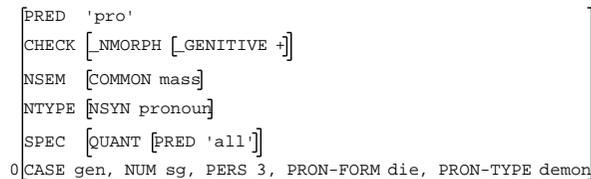
(ii) Certain predeterminers plus the indefinite article *eine* 'a'   The predeterminers *manch* 'some', *solch* 'such', *was für* 'what', and welch 'what' obligatorily precede the indefinite article. This is encoded by the template @PLUS-EINE, which constrains the feature SPEC DET PRED accordingly.

```
PLUS-EINE =
        (^SPEC {DET|NUMBER} PRED FN) =c eine.
```

The template @PLUS-EINE is called by the lexicon entries of the respective predeterminers, cf. the entry of *welch* 'what'.

```
welch    !INT-S xle @(INT %stem) @(SPEC-COUNT);
         !Dpre-S xle @(INT %stem) @(PLUS-EINE); ONLY.
```

As an example, see the f-structure of (271).

(271)   *welch*   *einem Unsinn*
        what[UNINFL] a[ST] nonsense
        'what nonsense'

```
"welch einem Unsinn"

  ⎡PRED  'Unsinn'                                      ⎤
  ⎢CHECK ⎡_SPEC-TYPE ⎡_COUNT +, _DET attr⎤⎤            ⎥
  ⎢NSEM  ⎡COMMON mass⎤                                 ⎥
  ⎢NTYPE ⎡NSYN common⎤                                 ⎥
  ⎢      ⎡    ⎡PRED     'eine' ⎤⎤                       ⎥
  ⎢      ⎢DET ⎢DET-TYPE indef  ⎥⎥                       ⎥
  ⎢SPEC  ⎢    ⎣                ⎦⎥                       ⎥
  ⎢      ⎣INT ⎡PRED 'welch'⎤   ⎦                        ⎥
 0⎣CASE dat, GEND masc, INFL strong-det, NUM sg, PERS 3 ⎦
```
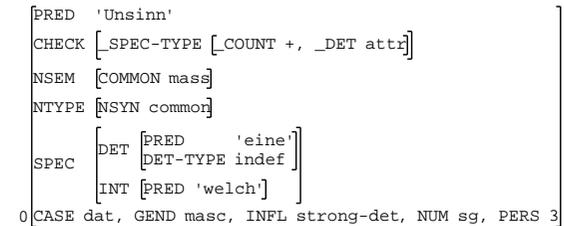
The template @PLUS-EINE alternatively constrains the feature SPEC NUMBER PRED, see above. This alternative captures pronominal use of *eine* 'one', as in (272).

(272)   *manch*   *einer*
        some[UNINFL] one[ST]
        'someone'

```
"manch einer"

  ⎡PRED 'pro'                      ⎤
  ⎢      ⎡NUMBER ⎡PRED 'eine' ⎤⎤   ⎥
  ⎢SPEC  ⎢                    ⎥⎥   ⎥
 0⎣      ⎣QUANT  ⎡PRED 'manch'⎤⎦⎦  ⎦
```
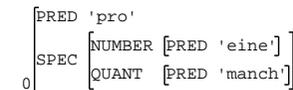
Note, finally, that our class of predeterminers is probably too restricted, in that it excludes examples such as (273) or (274).

(273) (?) *etliche   andere   jungen   Leute*
many[ST] other[ST] young[WK] people

(274) (?) *einige   wenige jungen   Leute*
some[ST] few[ST] young[WK] people
'some few young people'

10.9.2   D + Aquant

Multiple quantifying expressions can also consist of a (strong) determiner and a (weak) quantifying adjective. This type of co-occurrence is clearly less restricted than co-occurrence of multiple determiners, presented above.

This is reflected by our analysis. In contrast to predeterminers (which are to be listed in the lexicon), any determiner may precede a quantifying adjective in our implemenation. The Determiner (D) is a sister constituent of the quantifying adjective (Aquant), cf. (275). (Multiple quantifying adjectives are addressed below (cf. sec. 10.9.3).)

(275)   *die   beiden   Kinder*
the[ST] both[WK] children
'the two children'
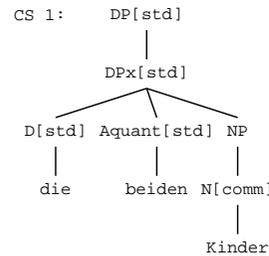
```
CS 1:       DP[std]
               |
            DPx[std]
            /    |    \
      D[std] Aquant[std] NP
        |        |        |
       die     beiden  N[comm]
                           |
                        Kinder
```

```
Dbar(_type) =
      ...

      "EX: der Mann (the man)"
      "EX: alle Leute (all people)"
      "EX: welche beiden Frauen (which two women)"
      D[_type]: @(RIGHT-SISTER);
```

```
      Aquant[std]*
      ...
```

The type of the DP is again determined by the first quantifying expression (as in the case of a predeterminer plus determiner). For instance, an interrogative DP may consist of an interrogative determiner followed by an ordinary (standard) quantifying adjective (276). (For more examples (cf. sec. 10.9.4).)

(276)   *welche   beiden   Kinder*
which[ST] both[WK] children
'which two children'

```
CS 1:       DP[int]
               |
            DPx[int]
            /    |    \
      D[int] Aquant[std] NP
        |        |        |
      welche   beiden  N[comm]
                           |
                        Kinder
```
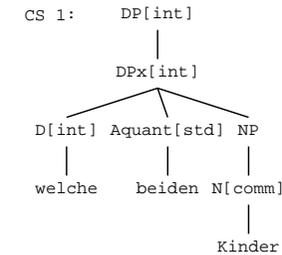
Note that the determiner which precedes the quantifying adjective may be complex itself, i.e. it may consist of a predeterminer and determiner, e.g. as in (277). (The f-structure analysis of multiple indefinites, such as *all* 'all' and *viele* 'many', is discussed below (cf. sec. 10.9.3).)

(277)   *all   die   vielen   Leute*
all[UNINFL] the[ST] many[WK] people
'all the numerous people '

```
CS 1:       DP[std]
               |
            DPx[std]
            /       |      \
      D[std]   Aquant[std]  NP
       / \         |         |
Dpre[std] D[std] vielen  N[comm]
   |       |                 |
  all     die              Leute
```
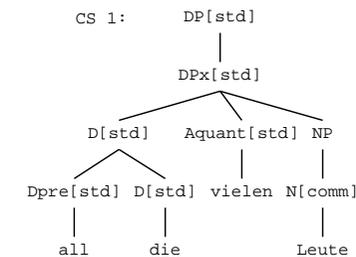
As presented above, in the case of multiple determiners (i.e., Dpre + D), the predeterminer imposes idiosyncratic restrictions on the following determiner. In contrast, in the case of D + Aquant co-occurrences the determiner does not restrict the following quantifying adjective.

However, certain quantifying adjectives impose idiosyncratic constraints on possible determiner-adjective combinations, e.g. a quantifying adjective may disallow definiteness marking, cf. (278). The restriction is encoded by the template @NO-DEF, which represents the opposite of @PLUS-DEF, by disallowing the definiteness feature CHECK _SPEC-TYPE _DEF.

(278)   *die   mehreren   Kinder
         the[ST] some[WK] children

```
mehrere    !INDEF-S[adj] xle @(QUANT %stem) @(NO-DEF); ONLY.


NO-DEF =
        ~(^CHECK _SPEC-TYPE _DEF).
```

Other examples of idiosyncratic restrictions are provided by *meiste* 'most', which requires definiteness marking (279), and *jede* 'each', which requires the presence of the indefinite article (280).

(279)   (see corpus example DP10.10, Appendix A)
        ihre   meisten   Nachbarn
        their[ST] most[WK] neighbours
        'most of their neighbours'

(280)   *eines jeden   Freundes*
        a[ST] each[WK] friend
        'of every friend'

jede 'each' normally inflects like a determiner, but if preceded by *eine* 'a' it is analyzed as an adjective.

Note that *jeden* 'each' in the above example could be analyzed as a strong determiner as well (= in the so-called "weak" genitive inflection variant (cf. p. 226)). In the corpus, we found no unambiguous instances of *jede* followed by an adjective. For the implementation, it is easier to assume that *jede* inflects like an adjective if preceded by *eine* 'a' (this seems to contradict the results for *jede* 'each' in the table above (cf. sec. 10.7); but note that in this table only unambiguous instances are taken into account).

Note that many D-Aquant combinations are ruled out due to feature clashes, e.g. if two indefinites are to be introduced as instances of SPEC QUANT (281).

(281)   *alle   sämtlichen Kinder*
          all[ST] all[WK]   children

Inspite of these restrictions, our implementation clearly still overgenerates. However, many of the combinations seem to be semantically restricted, hence they can be modelled only approximately in the syntax.

10.9.3   Multiple Aquants

Finally, we discuss multiple quantifying adjectives. Similarly to the co-occurrence of D and Aquant, there are no lexical restrictions on multiple quantifying adjectives.

The adjectives are analyzed as sister constituents (similar to attributive adjectives), cf. (282). The type of the DP is again determined by the first quantifying expression (as in the cases above).

(282)   *wieviele   andere   Kinder*
         how_many[ST] other[ST] children
         'how many other children'
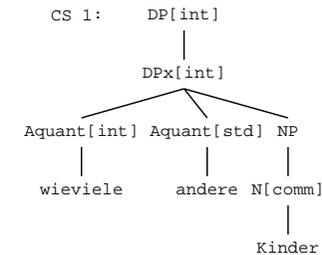
```
CS 1:      DP[int]
               |
            DPx[int]
          /     |     \
Aquant[int] Aquant[std] NP
     |           |       |
  wieviele    andere  N[comm]
                         |
                       Kinder
```

```
Dbar(_type) =
    ...
     "EX: wieviele Frauen (how many women)"
     "EX: einige wenige Frauen (some few women)"
     Aquant[_type]
     Aquant[std]*
    ...
```

Most instances of multiple quantifying adjectives consist of multiple indefinites. These are uniformly represented by the feature SPEC QUANT, which incurs f-structure clashes.

Therefore, selected indefinites, such as *viele* 'many' and *wenige* 'few', are represented by the (set-valued) feature SPEC AQUANT ("adjectival QUANT"). Hence, they may cooccur with other indefinites, cf. the f-structure analysis of (283).

(283)　*einige　　wenige　andere　　Kinder*
　　　　some[ST] few[ST] other[ST] children
　　　　'some few other children'
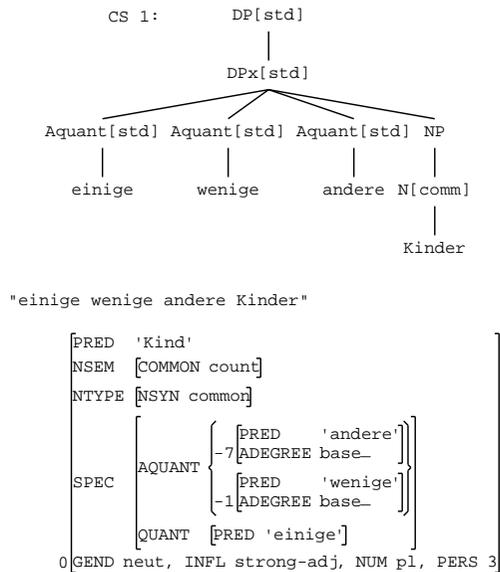
```
                          DP[std]
            CS 1:            |
                          DPx[std]

          Aquant[std] Aquant[std] Aquant[std] NP
               |           |           |       |
             einige     wenige      andere N[comm]
                                              |
                                            Kinder
```

```
      "einige wenige andere Kinder"

            ⎡PRED  'Kind'                              ⎤
            ⎢NSEM  [COMMON count]                      ⎥
            ⎢NTYPE [NSYN common]                       ⎥
            ⎢           ⎧    ⎡PRED    'andere'⎤⎫       ⎥
            ⎢           ⎪ -7 ⎣ADEGREE base_  ⎦⎪        ⎥
            ⎢      AQUANT⎨                     ⎬        ⎥
            ⎢SPEC       ⎪    ⎡PRED    'wenige'⎤⎪        ⎥
            ⎢           ⎩ -1 ⎣ADEGREE base_  ⎦⎭        ⎥
            ⎢      QUANT [PRED 'einige']               ⎥
            0⎣GEND neut, INFL strong-adj, NUM pl, PERS 3⎦
```

```
andere     !INDEF-S xle "special: multiple indefinites"
                        "EX: einige wenige andere "
                        "(some few others)"
                        @(AQUANT %stem) @(AMBIG-INFL); ETC.

AQUANT(_stem) =
        (!PRED) = '_stem'
        ! $ (^SPEC AQUANT).
```

Cardinals provide further examples of multiple quantifying expressions. Analyzed as category Acard, they can combine with diverse kinds of determiners (284), (285), (286).

(284)　*die　　　zwei*
　　　　the[ST] two

```
                          DP[std]
            CS 1:            |
                          DPx[std]
                             |
                        D[std]  NP
                           |     |
                         die   Acard
                                 |
                               zwei
```

```
      "die zwei"

            ⎡PRED  'pro'                                       ⎤
            ⎢CHECK [_SPEC-TYPE [_COUNT +, _DEF +, _DET attr]]   ⎥
            ⎢NTYPE [NSYN pronoun]                              ⎥
            ⎢       ⎡    ⎡PRED    'die'⎤                ⎤       ⎥
            ⎢       ⎢DET ⎣DET-TYPE def ⎦                ⎥       ⎥
            ⎢SPEC   ⎢       ⎡PRED      'zwei'⎤          ⎥       ⎥
            ⎢       ⎣NUMBER ⎣NUMBER-TYPE card⎦          ⎦       ⎥
            0⎣INFL strong-det, NUM pl, PERS 3, PRON-TYPE null⎦
```
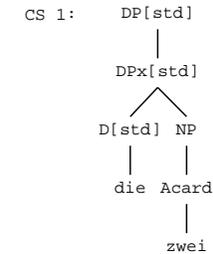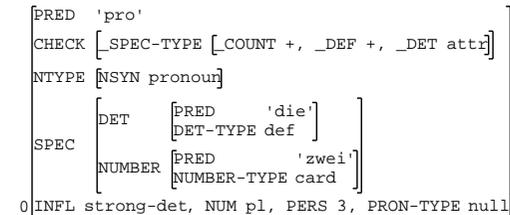
(285)　*alle　　drei Kinder*
　　　　all[ST] three children

(286)　*keine　vier Kinder*
　　　　no[ST] four children

The assumption that certain quantifying adjectives may project a set-valued feature (namely, SPEC AQUANT) hints at the close relationship between these quantifying expressions and ordinary adjectives (which project the set-valued feature ADJUNCT).

Modification by special adverbs and particles constitutes a further property common to certain quantifying expressions and ordinary adjectives. For instance, the adverb *sehr* 'very' may precede adjectives or quantifying expressions, such as *wenig* 'few', cf. (287).

(287)　*sehr wenig Menschen*
　　　　very few　people

Moreover, similar to adjectives, *wenig* 'few' can be modified by comparison particles, e.g. *so* 'so', which may trigger comparison phrases, as in (288).

(288) *So wenig Menschen kamen, daß Hans ging.*
so few people came that H. went
'The number of people who came was so low that Hans left.'

In our analysis, the comparison phrase (*daß Hans ging*) is subcategorized by the comparison particle (*so*) and functions as a special type of oblique, OBL-COMPAR, cf. the f-structure below.

```
"So wenig Menschen kamen, daß Hans ging."
        ⎡PRED    'kommen<[17:Mensch]>'                                    ⎤
        ⎢        ⎡PRED 'Mensch'                                           ⎥
        ⎢        ⎢        ⎡PRED    'wenig'                              ⎤ ⎥
        ⎢        ⎢        ⎢PRED        'so<[104:dass]>'                 ⎥ ⎥
        ⎢SUBJ    ⎢        ⎢            ⎡PRED   'dass<[128:gehen]>'    ⎤ ⎥ ⎥
        ⎢        ⎢SPEC QUANT⎢COMPAR   ⎢           ⎡PRED   'gehen<[112:Hans]>'⎤⎥ ⎥ ⎥
        ⎢        ⎢        ⎢        ⎢OBL-COMPAR⎢OBJ        ⎢SUBJ 112[PRED 'Hans']⎥⎥⎥ ⎥ ⎥
        ⎢     17 ⎣      2⎣      0⎣        104⎣  128⎣                ⎦⎦⎦ ⎦ ⎥
        ⎢                                                                ⎥
        ⎣57 TOPIC  [17:Mensch]                                          ⎦
```
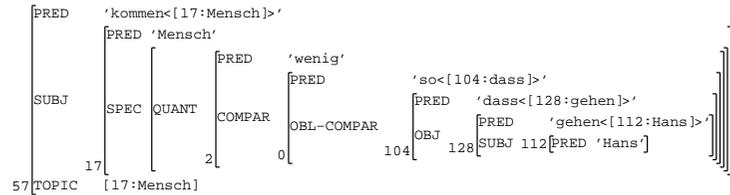
Both types of modifications (by adverbs such as *sehr* 'very' or by comparison particles) is restricted to quantifying expressions that are marked by the feature ADEGREE. This feature is introduced by the template @SPEC-ADEGREE, cf. the lexicon entry of *wenig* 'few'.

```
wenig    !INDEF-S xle @(QUANT %stem)
                     @(SPEC-ADEGREE QUANT base_);
         =ADV-S xle;
         =ADVadj *;
         ONLY.
```

```
SPEC-ADEGREE(_desig _type) =
         "specifiers that can be modified by ADVadj,
         which in turn checks for ADEGREE feature"
         (^SPEC _desig ADEGREE) = _type.
```

10.9.4 Examples

We complete the section on multiple determiners by example phrases, illustrating co-occurrences of (i) Dpre + D, (ii) D + Aquant, (iii) Aquant + Aquant, (iv) D + Aquant + Aquant, and (v) Dpre + D + Aquant. (Constructions that involve 3 quantifying expressions are marginal.)

All example phrases instantiate one of the two inflection patterns below. (For reasons of transparency, we ignore invariant adjectives here.)

```
[ D[ST]+ , D[UNINFL]* ]   Aquant[WK]*  A[WK]*

D[UNINFL]*                Aquant[ST]*  A[ST]*
```

(ia) Dpre + D: *alle/all* 'all' plus definiteness

Dpre[ST]	D[ST]	A[WK]	N
alle	*die*	*alten*	*Menschen*
all	the	old	people
'all these old people'			
alle	*diese*	*dummen*	*Fragen*
all	these	stupid	questions
'all these stupid questions'			
alle	*beide*	*vorgebrachten*	*Einwände*
all	both	made	objections
'both objections brought forward'			

Dpre[UNINFL]	D[ST]	A[WK]	N
all	*diese*	*dummen*	*Fragen*
all	these	stupid	questions
'all these stupid questions'			
all	*ihre*	*dummen*	*Gedanken*
all	her	stupid	thoughts
'all her stupid thoughts'			

In a similar construction, the predeterminer *all* 'all' precedes a pronoun, cf. (289).

(289) *all derer*
all these
'these all'

```
CS 1:    DP[std]
            |
         DPx[std]
            |
         PRON[std]
           /    \
   Dpre[std]   PRON[std]
        |          |
       all       derer
```
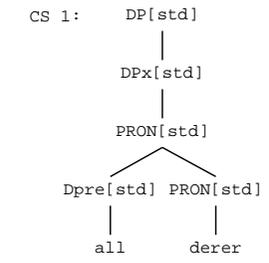
```
"all derer"

      ⎡PRED  'pro'                                                              ⎤
      ⎢CHECK  [_NMORPH [_GENITIVE +]]                                          ⎥
      ⎢                                                                        ⎥
      ⎢NSEM   [COMMON mass]                                                    ⎥
      ⎢NTYPE  [NSYN pronoun]                                                   ⎥
      ⎢SPEC   [QUANT [PRED 'all']]                                            ⎥
    0 ⎣CASE gen, GEND fem, NUM sg, PERS 3, PRON-FORM die, PRON-TYPE demon⎦
```

(ib) Dpre + D: predeterminer plus *eine* 'a'

Dpre[UNINFL]	D[ST]	A[ST]	N
manch	*einem*	*guten*	*Freund*
some	a	good	friend
'some good friend'			
solch	*einem*	*guten*	*Freund*
such	a	good	friend
'some good friend'			
was für	*einem*	*alten*	*Mann*
what for	a	old	man
'what an old man'			

Dpre[UNINFL]	D[UNINFL]	A[ST]	N
manch	*ein*	*guter*	*Freund*
some	a	good	friend
'some good friend'			
solch	*ein*	*netter*	*Empfang*
such	a	nice	welcome
'such a nice welcome'			
welch	*ein*	*netter*	*Empfang*
which	a	nice	welcome
'what a nice welcome'			

(ii) D + Aquant

D[ST]	Aquant[WK]	A[WK]	N
das	*meiste*	*gestohlene*	*Geld*
the	most	stolen	money
'most of the stolen money'			
ihre	*meisten*	*direkten*	*Nachbarn*
her	most	direct	neighbours
'most of her direct neighbours'			
die	*vielen/wenigen*	*dummen*	*Fragen*
the	many/few	stupid	questions
'the many/few stupid questions'			
das	*eine*	*große*	*Lager*
the	one	big	camp
'the one big camp'			
das	*selbe*	*nette*	*Kind*
the	same	nice	child
'the same nice child'			
alle	*beiden*	*vorgebrachten*	*Einwände*
all	both	made	objections
'both objections brought forward'			
welche	*beiden*	*vorgebrachten*	*Einwände*
which	both	made	objections
'which two objections brought forward'			
eines	*jeden*	*großen*	*Luxushotels*
a	each	big	luxury_hotel
'any big luxury hotel'			

D[UNINFL]	Aquant[ST]	A[ST]	N
ein	*solcher*	*dummer*	*Fehler*
a	such	stupid	error
'such a stupid error'			
kein	*solches*	*altes*	*Geld*
no	such	old	money
'no such old money'			
ihr	*meistes*	*gestohlenes*	*Geld*
her	most	stolen	money
'most of her stolen money'			

The following examples are instances of D + Aquant as well, cf. (290) and (291).

(290)   *manch*   *einer*
            some[UNINFL] one[ST]
            'someone'

(291)   *was für*   *einer*
            what for[UNINFL] one[ST]
            'what person'

However, this usage of *eine* 'one' is special in that *eine* is obligatorily "intransitive", i.e. the (optional) NP sister must be omitted, cf. (292).

(292)   *manch   *einer*   *Hund*
            some[UNINFL] one[ST] dog

This restriction is encoded by a c-structure constraint, similar to the (opposite) constraint of obligatorily transitive determiners (cf. sec. 10.10).

(iii) Aquant + Aquant

Aquant[ST]	Aquant[ST]	A[ST]	N
einige	*wenige*	*junge*	*Leute*
some	few	young	people
'some few young people'			
etliche	*andere*	*junge*	*Leute*
many	other	young	people
'many other young people'			

(iv) D + Aquant + Aquant

D[ST]	Aquant[WK]	Aquant[WK]	A[WK]	N
keine	*solchen*	*vielen*	*jungen*	*Leute*
no	such	many	young	people
'no such numerous young people'				
seine	*etlichen*	*anderen*	*neuen*	*Autos*
his	many	other	new	cars
'his numerous other new cars'				

D[UNINFL]	Aquant[ST]	Aquant[ST]	A[WK]	N
kein	*anderes*	*solches*	*verrücktes*	*Unterfangen*
no	other	such	crazy	enterprise
'no other similar crazy enterprise'				
kein	*solches*	*anderes*	*verrücktes*	*Unterfangen*
no	such	other	crazy	enterprise
'no other similar crazy enterprise'				

(v) Dpre + D + Aquant

Dpre[UNINFL/ST]	D[ST]	Aquant[WK]	A[WK]	N
all	*die*	*vielen*	*jungen*	*Leute*
all	the	many	young	people
'all the numerous young people'				
alle	*diese*	*vielen*	*jungen*	*Leute*
all	these	many	young	people
'all these numerous young people'				

Note that our grammar overgenerates and, e.g., allows for (293).

(293)   *welch   *ein*   *mancher dummer Fehler*
            which[UNINFL] a[UNINFL] some[ST] stupid[ST] error

10.10   D/Aquant: Transitive Determiners

As we have seen, some determiners cannot be used "intransitively", i.e. they never project a DP on their own, cf. (294), (295).

(294)   **Manch kommt.*
            some   comes

(295)   **Ein kommt.*
            a   comes

These determiners are marked by .Attr, e.g. *manch* 'some', or by .Def or .Indef (for articles).

Transitivity of .Attr determiners is encoded as a c-structure constraint, by requiring the D node to have a right sister. This is effected by the constraint (∗RIGHT_SISTER) attached to the D node in the @Dbar macro, via the template @RIGHT-SISTER.

```
Dbar(_type) =
     ...

       "EX: der Mann (the man)"
       "EX: alle Leute (all people)"
       "EX: welche beiden Frauen (which two women)"
       D[_type]: @(RIGHT-SISTER);
       Aquant[std]*
     ...



RIGHT-SISTER =
       "'attributive' determiners, want right sister
       (NP or Aquant)"
       @(IF (^CHECK _SPEC-TYPE _DET) = attr
       (* RIGHT_SISTER) ).
```

This analysis correctly excludes examples like (296) or (297).

(296)   *ein
         a

(297)   *manch
          some

The c-structure constraint is triggered by the feature [CHECK _SPEC-TYPE _DET attr]. This feature is introduced in the macro @DISTR-INFL, if and only if the quantifying expression in question is marked by .Attr (or .Def/.Indef). (Only these tags are assigned the sublexical category DISTR-ATTR.)

```
DISTR-INFL(_cat) =
     ...
       "_cat = det/predet"
       DISTR-ATTR_BASE
       e: { _cat = det
            (^CHECK _SPEC-TYPE _DET) = attr
          | "special disjunct for predeterminers:
            don't introduce CHECK feature here
            – predeterminers marked by .Attr automatically have
            right sister (namely D)
            – CHECK feature of predet would unify with right
            sister D, which would then need a right sister
            itself; this would exclude:"
```

```
            "EX: Manch einer kam. (Some people came.)
            (since 'einer' would need right sister due to
            requirements of 'manch')"
            _cat = predet };

       | "_cat = det/predet/adj"
         DISTR-PRO_BASE
       }
     ...
```

Note that the requirement for a sister to the right of D need not be fulfilled by an c-structure complement of category NP, as in (298), or (299).

(298)   *ein Mann*
        a   man
(299)   *ein rotes*
        a   red
        'a red one'
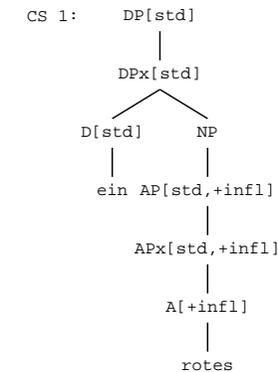
Alternatively, an Aquant sister, e.g. *mancher* 'some' as in (300) or (301), may represent the right sister of the .Attr expression, building a complex quantifying determiner.

(300)   *ein mancher Freund*
        a   some   friend
        'some friend'
(301)   *ein mancher*
        a   some
        'some'

One could consider an alternative encoding of (obligatory) transitivity, namely by stating a corresponding c-structure constraint in the lexicon entry of the tag .Attr, as sketched in the following entry.

```
.Attr DISTR-ATTR xle (* MOTHER MOTHER RIGHT_SISTER); ONLY.
```

'*' refers to the .Attr tag itself, '* MOTHER' refers to the sublexical category DISTR-ATTR, and '* MOTHER MOTHER' to the syntactic category D, which directly dominates DISTR-ATTR. Hence, the c-structure constraint requires that the category D have a right sister, e.g. NP.

```
         D
         |
     DISTR-ATTR
         |
       .Attr
```
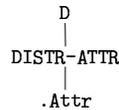
However, a problem of this approach are complex determiners. A node Dpre and a node D are again dominated by a D node, as shown below.

```
         D
        / \
     Dpre   D
```
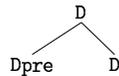
It is only for the upper D node that we require the presence of a right sister (the lower D node never has a right sister constituent). Due to this uncertainty of (depth of) embedding, the constraint is best stated in the @Dbar macro rather than in the lexicon entry of .Attr or in the D rule.

For similar reasons, the equation ($\uparrow$CHECK _SPEC-TYPE _DET) = attr is introduced in the macro @DISTR-INFL rather than the lexicon entry of .Attr: Since all features of Dpre and D unify, it would not be transparent any more whether this equation has been introduced by Dpre or D—but surely, we do not want to require the upper D node to have a right sister if only Dpre is marked by .Attr.

Therefore, the equation ($\uparrow$CHECK _SPEC-TYPE _DET) = attr is best introduced if (i) the quantifying expression in question is marked by .Attr (or .Def/.Indef), i.e. if the distributional sublexical category is DISTR-ATTR, and (ii) the expression functions as a determiner and not as a predeterminer (this is the sole purpose of the parameter _cat = predet).

Note that the c-structure constraint on D to have a right sister works correctly because postnominal modifiers (which are covered by the macro @DPpost) are attached

at DP level, and not within NP. If @DPpost constituents were attached at the NP level, they would qualify as right sisters of D, and we could not exclude examples like (302).

(302) *ein mit Auto
        a   with car

We now turn to aspects of the relation quantifying expressions—head noun.

10.11 D/Aquant: Restrictions on the Head Noun

Certain quantifying expressions impose restrictions on their head noun. This section addresses two (related) restrictions, (i) restrictions on the noun's granularity (count vs. mass noun), and (ii) restrictions on the number of the noun.

10.11.1 Count vs. Mass Noun

In German as well as in English, singular count nouns normally need a determiner to be grammatical, compare (303) and (304).

(303)   *Hans kauft ein Auto.*
         Hans buys a   car.

(304)   **Hans kauft Auto.*
         Hans buys car.

We call *eine* 'a' in the above example a "licensing" determiner, which allows singular count nouns to occur. The question is now whether the remaining quantifying expressions in German may also function as licensing determiners (in the above sense). In fact, this depends on the (type of) specific quantifying expression.

Expressions licensing singular count nouns The following types of quantifying expressions may precede singular count nouns.

- Articles (305)

  (305)   *ein Hund*
           a   dog

- Possessives (including prenominal genitives, in the SPEC-DP position) (306)

  (306)   *mein Hund*
           my   dog

- Most demonstratives (307)

    (307)   *dieser Hund*
             this   dog

- Cardinals (308)

    (308)   *1 Hund*
             1 dog

In our implementations, the respective expressions introduce a specific feature (↑CHECK ␣SPEC-TYPE ␣COUNT) = +, via the template @SPEC-COUNT. See the example entry of the part-of-speech tag marking articles, +ART.

```
+ART        ART-T xle @(SPEC-COUNT); ONLY.

SPEC-COUNT =
        (^CHECK ␣SPEC-TYPE ␣COUNT) = +.
```

The feature [CHECK ␣SPEC-TYPE ␣COUNT +] is required by all singular count nouns via the template @COUNT-NOUN. A count noun then either has to be plural, in which case it does not need any specifier. Or it is singular and accompanied by a licensing specifier (i.e., marked by the feature CHECK ␣SPEC-TYPE ␣COUNT).

```
COUNT-NOUN =
        @(NTYPE common)
        @(NSEM COMMON count)
        @(IF (^NUM) = sg
        @(CHECK-SPEC ^)
        ).


CHECK-SPEC(␣desig) =
        ...

        { "either accompanied by some licensing determiner"
          (␣desig CHECK ␣SPEC-TYPE ␣COUNT)
        ...
```

A singular count noun may occur without a licensing specifier in special contexts, e.g. embedded in a PP, as in (309). Such cases are covered by further disjuncts of the template @CHECK-SPEC (not shown here).

  (309)   *Hunde ohne   Maulkorb*
           dogs   without muzzle

Besides the above expressions, the possessive *ihre* 'her', certain interrogatives (e.g. *welche* 'which'), and certain indefinites (e.g. *irgendeine* 'someone') license singular count nouns. These are all listed in the lexicon, calling the template @SPEC-COUNT.

```
irgendeine !INDEF-S[det] xle @(QUANT %stem) @(SPEC-COUNT);
              =PRON-S xle "indefinite pronoun"; ONLY.
```

Expressions disallowing singular count nouns   On the other hand, there are certain indefinites that disallow singular count nouns, compare (310) (*Hund* 'dog' is a count noun) and (311) (*Unheil* 'disaster' is a mass/abstract noun).

(310)   **aller Hund*
          all   dog

(311)   *alles Unheil*
          all   disaster

Whenever these indefinites occur in the singular form, they introduce the equation (↑NSEM COMMON) = mass, via the template @SPEC-MASS.

```
alle        !INDEF-S xle @(QUANT %stem) @(SPEC-MASS)
                         @(NO-DEF) @(AMBIG-INFL);
            !Dpre-S xle @(QUANT %stem) @(SPEC-MASS)
                         @(PLUS-DEF); ONLY.

SPEC-MASS =
        @(IF (^NUM) = sg
        (^NSEM COMMON) = mass
        ).
```

The feature [NSEM COMMON mass] marks mass nouns, [NSEM COMMON count] marks count nouns. Hence, a singular count noun cannot cooccur with indefinites such as *alle* 'all', cf. the feature clash in the analysis of (312).

(312)   **all Hund*
          all dog

$$
\begin{bmatrix}
\text{PRED} & \text{'Hund'} \\
\text{CHECK} & \begin{bmatrix} \text{\_SPEC-TYPE} & [\text{\_DET attr}] \end{bmatrix} \\
\text{NSEM} & \begin{bmatrix} \text{COMMON} & \begin{bmatrix} = \begin{matrix} \text{count} \\ \text{mass} \end{matrix} \end{bmatrix} \end{bmatrix} \\
\text{NTYPE} & [\text{GRAIN common}] \\
\text{SPEC} & \begin{bmatrix} \text{QUANT} & [\text{PRED 'all'}] \end{bmatrix} \\
\text{GEND masc, NUM sg, PERS 3}
\end{bmatrix}
$$

Note, however, that our noun lexicon focuses on mass and measure nouns. Hence, only few nouns are currently marked as count nouns.

Our analysis also accounts for data that involve complex quantifying expressions. Suppose an indefinite like *all* 'all' (which disallows singular count nouns) builds a complex determiner with a licensing specifier such as *der* 'the': *all der* 'all the'. Now, singular count nouns are still ungrammatical, cf. (313).

(313)   *all der Hund
         all the dog

The same holds for *viele* 'much', compare (314), (315).

(314)   *vieles Kind
         much child

(315)   *das viele Kind
         the much child

Expressions that are indifferent to singular count nouns   A third type of quantifying expression is indifferent with regard to count nouns. They neither license (316) nor disallow them (317) (if combined with another specifier).

(316)   *anderer Hund
         other   dog

(317)   *der andere Hund*
         the other   dog

In our implementation, these expressions just do not introduce any constraints on the features CHECK _SPEC-TYPE or NSEM.

The restrictions on the head noun that have been presented in this section clearly depend on the semantics of the quantifying expressions. The distinction between mass and count nouns refer to the granularity of the nouns. A related property is the number (singular, plural) of the head noun. The following section addresses the question whether quantifying expression impose restrictions on the number of their head noun.

10.11.2   Restrictions on Number

Many uninflected quantifying expression occur most naturally with head nouns in singular form, e.g. *manch* 'some', as in (318) and *viel* 'much' (319). (Some authors assume that uninflected *viel* 'much' can only be singular, e.g. Pafel 1994.)

(318)   *manch Zeitgenosse*
         some   contemporary

(319)   *viel   Unsinn*
         much nonsense

However, corpus queries reveal that these expressions may also occur with plural head nouns, cf. (320), (321), (322), (323), (324), (325), (326).

(320)   (see corpus example DP10.11, Appendix A)
         von   manch anderen Zeitgenossen
         from some   other   contemporaries

(321)   (see corpus example DP10.12, Appendix A)
         von solch taktischen Gesichtspunkten
         by   such tactical   aspects

(322)   (see corpus example DP10.13, Appendix A)
         mit viel   Vorschußlorbeeren
         with much premature_praise

(323)   (see corpus example DP10.14, Appendix A)
         zu welch   groben Mißdeutungen
         to which coarse misinterpretations

(324)   (see corpus example DP10.15, Appendix A)
         nur sehr wenig geeignete Standorte
         only very few   suitable sites

(325)   (see corpus example DP10.16, Appendix A)
         zuviel   Pausen
         too_many pauses

(326)   (see corpus example DP10.17, Appendix A)
         zuwenig Duschen
         too_few showers

An exception is *etwas* 'some', which only allows for singular head nouns, compare (327) and (328).

(327)   *etwas Milch*
        some milk

(328)   **etwas Weine*
        some wines

This restriction is encoded by the template @NUM in the lexicon entry of *etwas* 'some'.

```
etwas      !INDEF-S xle @(QUANT %stem) @(SPEC-MASS) @(NUM sg)
                        @(SEM-ETWAS (^SPEC QUANT) a_little)
                        { "allows for modification by 'so'"
                          "EX: so etwas Dummes "
                          "(such a stupid thing)"
                          @(SPEC-ADEGREE QUANT base_)
                          (^SPEC QUANT COMPAR PRED FN) =c so };
           =PRON-S xle;
           =ADV-S xle; ONLY.
```

    Note that *etwas* 'something/a little' is ambiguous. (i) *etwas* meaning 'something' is a real pronoun (of category PRON); examples such as (329) are instances of pronoun appositives (cf. sec. 10.2.4). In this construction, only nominalized adjectives can follow the pronoun.

(329)   *etwas*   *Nettes*
        something nice

(ii) In examples such as (330), *etwas* can only be read as 'a little'.

(330)   *etwas Geld*
        some money

Since these readings are easily confounded, *etwas* introduces the constraint (↑SEM-ETWAS) = a_little if used as a determiner (of category INDEF-S) and (↑SEM-ETWAS) = something if used as a pronoun (of category PRON).

    This chapter addressed core aspects of the DP's c-structure, i.e. all constituents that belong to the DP projection. The focus of this chapter was primarily on the analysis of the head of DP. The head is realized by quantifying expressions of category D and Aquant, depending on the inflection of adjacent attributive adjectives.

Chapter 11

Summary and Overview Tables

Contents

This chapter serves as an overview of the phenomena that are covered by the DP implementation. It also outlines the basic c-structure and f-structure properties of the phenomena in question. Finally, there are overview tables of the morphology tags and the morphology-syntax interface.

11.1 Phenomena

Our implementation covers many important aspects of the German DP. It accounts for various types of so-called "quantifying expressions": articles, possessives, demonstratives, indefinites, and interrogatives. It also includes multiple/complex quantifying expressions. Moreover, our analysis accounts for the varying inflection types (D vs. Aquant, depending on the inflection of adjacent attributive adjectives) that many quantifying expressions exhibit.

Furthermore, the implementation accounts for the analysis of pronouns and various postnominal modifiers and arguments. Due to space limitations, however, the documentation of these constructions could not be included.

11.2 Basic Properties

(i) Basic c-structure of the DP
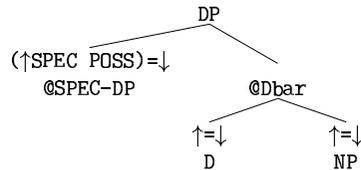
(ii) Positions of (multiple/complex) quantifying expressions
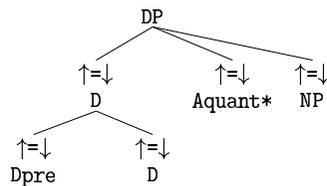
(iii) Postnominal constituents
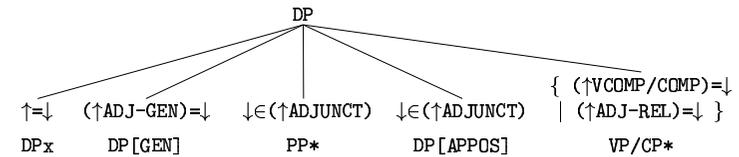
11.3 Quantifying Expressions

11.3.1 Morphology-Syntax Interface

The following table shows the relation between (i) the morphological part-of-speech tag of a quantifying expression, (ii) its sublexical stem category, and (iii) the f-structure feature under which the quantifying expression will end up.

	POS Tag	Sublexical Cat.	Function
Article	+ART	ART-S	SPEC DET
Possessive	+POSS	POSS-S	SPEC POSS
Demonstrative	+DEM	DEM-S	SPEC DEM
Indefinite	+INDEF	INDEF-S	SPEC QUANT
			(SPEC AQUANT)
			(SPEC NUMBER)
Interrogative	+WPRO	INT-S	SPEC INT

Table 11.1: Quantifying expressions: POS tags, stem category, and function

11.3.2 POS and Distribution Tags

The following tables list all quantifying expressions comprised in the current version of our morphology. For each type of quantifying expression, the tables provide (i) the part-of-speech (POS) tag, (ii) the distributional tag, and (iii) an exhaustive list of all instances of the respective type.

POS Tag	Distribution	Instances
+ART	(.Def)	*die*
	(.Indef)	*eine*

Table 11.2: Morphology tags for articles

POS Tag	Distribution	Instances
+POSS	.Attr	*ihre*
	.Pro	*ihre, ihrige*

Table 11.3: Morphology tags for possessives

POS Tag	Distribution	Instances
+DEM	.Pro	*diejenige, diese, dieselbe, ebendiese, jene, selbe, selbige*
	.Subst	*die*

Table 11.4: Morphology tags for demonstratives

POS Tag	Distribution	Instances
+INDEF	.Attr	*all, etwas, irgendeine, keine, manch, so, solch*
	.Pro	*alle, allerlei, andere, anderlei, beide, beiderlei, bißchen, dergleichen, dreierlei, ebensoviel, ebensoviele, ebensowenige, eine, einerlei, einige, erstere, etliche, etwelche, irgendwelche, jede, jedwede, jegliche, keinerlei, lauter, letztere, manche, mancherlei, mehr, mehrere, mehrerlei, meiste, paar, reichlich, sämtliche, solche, solcherlei, sonstwelche, soviel, viel, viele, vielerlei, wenig, wenige, weniger, zuviel, zuviele, zuwenig, zuwenige, zweierlei*
	.Subst	*beide, etwas, ihresgleichen, irgendetwas, irgendjemand, irgendwas, irgendwer, jedermann, jemand, keine, man, nichts, niemand, nix, sonstjemand, sonstwas, sonstwer, unsereine, unsereins, was, welche*

Table 11.5: Morphology tags for indefinites

POS Tag	Distribution	Instances
+WPRO	.Attr	*was (für), welch*
	.Pro	*welche, wieviel, wieviele*
	.Subst	*was, wer*

Table 11.6: Morphology tags for interrogatives

11.4   Nominal Inflection: Morphology Tags

The following table displays all tags encoding nominal inflection, including underspecified tags, as provided by our current morphological analyzer. The middle column lists the morphological tags, the last column presents the meaning

of the tags. For underspecified tags, the last column lists the single tags that the underspecified tags correspond to.

	Morphology Tag	Meaning
Person	.1	first person
	.2	second person
	.3	third person
	.13	.1 .3
Gender	.Masc	masculine
	.Fem	feminine
	.Neut	neuter
	.MFN	.Masc .Fem .Neut
	.MN	.Masc .Neut
Number	.Sg	singular
	.Pl	plural
Case	.Nom	nominative
	.Gen	genitive
	.Dat	dative
	.Acc	accusative
	.NGDA	.Nom .Gen .Dat .Acc
	.NA	.Nom .Acc
	.GD	.Gen .Dat
	.DA	.Dat .Acc
Inflection type	.St	strong
	.Wk	weak
	.None	uninflected
	.Invar	invariant

Table 11.7: Morphology tags marking nominal inflection

Appendix A

DP Corpus Data

Corpus Examples of Chapter 7

DP7.1   Durch seine höchst riskante und triftige Äquilibristik von historisch-politischem Ernst und spielerisch-ästhetischer Selbstreflexion stellt es **vieles Gegenwärtige** in den Schatten .

DP7.2   Der Profi beherrscht die Szene , es wird Leistung produziert , wohingegen der Amateur , der (Sport-)Liebhaber , auf den **alle die schönen Definitionen und Bonmots** noch zutreffen mögen (" körperliche Tätigkeit um ihrer selbst willen " , " nicht der Sieg , die Teilnahme ist wichtig " usw.) , ganz unten rangiert , in den A- , B- und C-Klasse-Vereinen .

DP7.3   " Wenn wir nicht so anfangen zu bauen wie hier " , sagte er , " dann sind **alle unsere schönen Sprüche** von der CO 2-Reduzierung und vom Klimaschutz Makulatur .

DP7.4   Die beiden großen politischen Lager müssen zusammenwirken . **Das eine große Lager** war - wie ich auch immer gesagt habe - zu keiner Zeit bereit , eine wirkliche praktische Lösung ohne Grundgesetzänderung zu vollziehen .

DP7.5   Nach den ersten Gewittergüssen durften sie dann ihre nassen Matratzen **gegen trockene** austauschen und sich später wie Oumar Guirassy aus Metallgittern und Plastikbahnen einen behelfsmäßigen Schutz bauen .

DP7.6   Manche der auf dem Platz Kampierenden hätten doch Wohnungen gehabt , sagt er , sie wollten sich jetzt nur **bessere** erzwingen .

DP7.7 **Gerade um der Debatte in der landwirtschaftlichen Fachwelt willen** erläutert Dr. Volker Eggermann , Leiter des Friedberger Veterinäramtes , das bisher Einmalige an diesem Akt der (Schweine-)-Völkerverständigung .

DP7.8 **Um der Gesundheit der 8000 Einwohner willen** zwangen der Gemeinderat und Fürst Wolfgang Ernst III. schon 1831 die 28 Mitglieder der Metzegerzunft , die Hof- und Hausschlachtung aufzugeben und im Marstall neben dem Schloß , wo heute die Hochschule für Gestaltung ist , ein Schlachthaus zu errichten .

DP7.9 **Der einseitigen Ausrichtung der Produktionen wegen** gibt es in der Region kaum Ersatzarbeitsplätze .

Corpus Examples of Chapter 10

DP10.1 Wenn **Babys Herz** abrupt aussetzt

DP10.2 Doch die Mühlheimer trafen auch nur 54 Mal in **des Gegners Netz** , was nicht zuletzt an den Verletzungen von Goalgetter Michael Blank lag .

DP10.3 Nur wenige Mitglieder fortgeschrittenen Alters treffen sich noch jeden ersten Freitag im Monat im Haus Nied **zum gemütlichem Beisammensein** .

DP10.4 Bei **jedem mißglücktem Dribbling** werde ich ausgepfiffen .

DP10.5 Vor **diesem wirtschaftlichem Hintergrund** , so Telschow , " verzichten wir nicht auf höhere Mieten " .

DP10.6 Die verlorene Partie gegen Brasilien schmerzte nicht sonderlich , da Angola die DBB-Auswahl schon zuvor mit **jenem spektakulärem Triumph** über den gescheiterten Gastgeber Spanien ins Viertelfinale gehievt hatte .

DP10.7 Die Frau , die laut **mehrerer ärztlichen Atteste** an Asthma leidet, erreichte vor der 1. Kammer des Gerichts unter Vorsitz von Richter Jürgen Schuldt einen Vergleich , der sie zufriedenstellt :

DP10.8 [...] ein zweideutiges Mißverständnis , das selbst im heutigen Japan noch **einiges verschämte Kichern** verursachen dürfte .

DP10.9 Dort beginnt nämlich die Tournee von vier Rockbands aus Offenbach und **anderer hessischen Jugendzentren** während der Herbstferien .

DP10.10 Den Celebis ging es zu Hause besser als **ihren meisten Nachbarn** .

DP10.11 Was von **manch anderen Zeitgenossen** allerdings nicht gerade zu behaupten wäre .

DP10.12 Doch nicht nur die Auswahl , sondern auch der Zeitpunkt der Ernennung des Kandidaten für das Amt des Vize-Präsidenten werden in den USA traditionell von **solch taktischen Gesichtspunkten** bestimmt .

DP10.13 Das in Dressurreiter-Kreisen mit **viel Vorschußlorbeeren** bedachte neue Pferd im Stall Rothenberger/Schumacher , die in Belgien zu Beginn des Jahres erworbene Stute Bo , ist nach Angaben des Trainers in ihrer Entwicklung noch nicht so weit , um schon jetzt eine Alternative für Andiamo und Ideaal darstellen zu können .

DP10.14 Ist es nicht fatal , zu **welch groben Mißdeutungen** solche Nachlässigkeiten einen aufmerksamen , aber einfältigen Leser verleiten können ?

DP10.15 Für Betriebsstrukturen wie die unsere gibt es nur **sehr wenig geeignete Standorte** .

DP10.16 **Zuviel Ballverluste** im Mittelfeld leistete sich Stefan Sebastian und **zuviel Pausen** legte der Ex-Rot-Weiße Christian Peukert ein , dessen Pässe allerdings noch immer allererste Güte sind .

DP10.17 Es gibt **zuwenig Duschen** und der Eintritt ist zu teuer .

Appendix B

List of DP Tables

Appendix C

Index of DP Features, OT Marks, etc.

This index lists occurrences of features, OT marks, rules/macros, and templates within the example DP documentation (chapters 7–11). Page numbers of important occurrences are underlined.

As described in section 5.4.5, the index lists all references to features and OT marks in the text (thus ignoring occurrences within the cited grammar code). In contrast, rules/macros and templates are listed whenever their definition is cited (thus ignoring textual references to rules/macros and templates). Rules/macros and templates are sorted according to the phrasal category they adhere to; e.g. the template 'CASE' is listed under the headword 'MISC', since the template is defined within the file 'misc.tmpl.lfg'.

Appendix D

DP Grammar Code

Contents

D.1   Rules and Macros

```
STANDARD GERMAN RULES (1.0)

"#################################
DP core rules
#################################"

DP[_type $ {std int rel}] -->
        ( e: _type = std;
         ADVdp: @(ADJUNCT) )

        DPx[_type]: @(DP-COMPLETE);


        ( "optional constituents may follow DP"
         @(DPpost) ) .

DPx[_type] -->
        { "full DPs: verb in third person"
          e: @(PERS 3);

          { "no SPEC-DP, no D head"
            NP: _type = std;

          | "with SPEC-DP, no D head
            EX: Karls Hund (Karl's dog)"
            @(SPEC-DP _type)
            NP

          | "with D head, no SPEC-DP
            (use macro to keep c-structure flat)
            EX: ein Hund (a dog)"
            e: _type ~= rel;
            @(Dbar _type)
          }

        | "pronouns"
          PRON[_type]

          ( "optional constituents may follow pronoun"
           "EX: jemand Nettes aus Stuttgart"
           "(someone nice from Stuttgart)"
           @(PRON-APPOS) )
        }.
```

```
SPEC-DP(_type) =
        "prenominal genitives"
        e: @(SPEC-COUNT)
           @(SPEC-DEF);

        { "certain pronouns 'dessen/deren/wessen'"
          PRONspec[_type]: @(PRENOM-GENITIVE);

        | "proper nouns"
          NAMEP: _type = std
                 @(PRENOM-GENITIVE);

          ( APOSTROPHE: (^SPEC POSS) = !
                        "EX: Hans' Auto (Hans's car)"
                        "EX: *Paris' Bahnhöfe"
                        "(the stations of Paris)"
                        { (^SPEC POSS NSEM PROPER) =c first_name
                        | (^SPEC POSS NSEM PROPER) =c last_name };
          )

        | "certain title nouns (experimental)
          EX: Omas Auto (granny's car)"
          N[title]: _type = std
                    @(PRENOM-GENITIVE);
        }.


Dbar(_type) =
        "(use macro to keep c-structure flat)"
        {
          "EX: der Mann (the man)"
          "EX: alle Leute (all people)"
          "EX: welche beiden Frauen (which two women)"
          D[_type]: @(RIGHT-SISTER);
          Aquant[std]*

        | "EX: wieviele Frauen (how many women)"
          "EX: einige wenige Frauen (some few women)"
          Aquant[_type]
          Aquant[std]*
        }
```

```
      { NP
      | "empty head noun
        EX: viele der Gäste (many of the guests)"
        e: @(NULL ^)
          @(IF ~(^INFL) "for invariant determiners: no OBJ2"
          ~(OBJ2 ^) "EX: *Er hilft allerlei."
          );
      } .


"#################################
Sublexical rules
#################################"

D[_type] -->
      {
        @(QUANT-EXPRESSION det _type)

      | "predeterminer (added recursively)"
        "EX: all die vielen Leute
         (all these many people)"
        Dpre[_type]
        D[std]
        "first det determines DP type
        EX: welch ein Kind (what a child)"

      | "modifiers"
        "EX: sehr viel (very much)"
        e: _type = std;
        @(QUANT-MOD)
        D[_type]
      }.


Aquant[_type] -->
      {
        @(QUANT-EXPRESSION adj _type)

      | "modifiers"
        e: _type = std;
        @(QUANT-MOD)
        Aquant[_type]
      }.
```

```
Dpre[_type] -->
        "listed in dp.lex.lfg"
        e: _type ~= rel;
        Dpre-S_BASE

        { "EX: ein jeder (everybody)"
          ART-T_BASE: _type = std;

        | "EX: ihr meistes Geld (most of her money)"
          @(PNG (^SPEC POSS REFERENT))
          ( CAP-DERIV-F_BASE: (^SPEC POSS) = ! )
          POSS-T_BASE: _type = std;

        | "EX: diese vielen Leute (these many people)"
          ( WEAKGEN-F_BASE )
          DEM-T_BASE: _type = std;

        | "EX: alle diese Leute (all these people)"
          ( WEAKGEN-F_BASE )
          INDEF-T_BASE: _type = std;

        | "EX: welch eine Frau (what a woman)"
          ( WEAKGEN-F_BASE )
          INT-T_BASE: _type = int;
        }

        @(DISTR-INFL predet).


QUANT-EXPRESSION(_cat _type) =
        "used by quants (D and Aquant)"
        { "articles (inserted under DET)"
          e: _type = std;
          @(QUANTart _cat)

        | "demonstratives (inserted under DEM)"
          e: _type = std;
          @(QUANTdemon _cat)

        | "possessives (inserted under POSS)"
          e: _type = std;
          @(QUANTposs _cat)

        | "indefinites (inserted under QUANT)"
          e: _type = std;
```

```
        @(QUANTindef _cat)


    | "interrogatives (inserted under INT)"
        e: _type = int;
        @(QUANTint _cat)
    }.


D-INC[_type] -->
        "incorporated determiners
        EX: zum (to_the)"
        e: _type = std;
        DELIM_BASE
        @(QUANTart det).

QUANTart(_cat) =
        ART-S_BASE: _cat = det;
        ART-T_BASE
        @(DISTR-INFL _cat).

QUANTdemon(_cat) =
        @(DEM-STEM _cat)
        ( WEAKGEN-F_BASE: _cat = det; )
        DEM-T_BASE
        @(DISTR-INFL _cat).

QUANTposs(_cat) =
        POSS-S_BASE[_cat] "there is no general POSS-S category"
        @(PNG (^SPEC POSS REFERENT)) "features of addressee"
        ( CAP-DERIV-F_BASE: (^SPEC POSS) = ! )
        POSS-T_BASE
        @(DISTR-INFL _cat).


QUANTindef(_cat) =
        @(INDEF-STEM _cat)
        ( WEAKGEN-F_BASE: _cat = det; )
        INDEF-T_BASE
        @(DISTR-INFL _cat).

QUANTint(_cat) =
        @(INT-STEM _cat)
        ( WEAKGEN-F_BASE: _cat = det; )
        INT-T_BASE
        @(DISTR-INFL _cat).
```

```
"stem categories"
DEM-STEM(_cat) =
        { DEM-S_BASE[_cat] | DEM-S_BASE }.
INT-STEM(_cat) =
        { INT-S_BASE[_cat] | INT-S_BASE }.
INDEF-STEM(_cat) =
        { INDEF-S_BASE[_cat] | INDEF-S_BASE }.


DISTR-INFL(_cat) =
        "distributional and inflectional features"

        { "_cat = det/predet"
          DISTR-ATTR_BASE
          e: { _cat = det
               (^CHECK _SPEC-TYPE _DET) = attr
             | "special disjunct for predeterminers:
               don't introduce CHECK feature here
               - predeterminers marked by .Attr automatically have
               right sister (namely D)
               - CHECK feature of predet would unify with right
               sister D, which would then need a right sister
               itself; this would exclude:"
               "EX: Manch einer kam. (Some people came.)
               (since 'einer' would need right sister due to
               requirements of 'manch')"
               _cat = predet };

        | "_cat = det/predet/adj"
          DISTR-PRO_BASE
        }


        { "gender, case, number"
          @(GCN)
          "inflection type"
          { INFL-F_BASE[det]: { _cat = det
                              | _cat = predet };
          | INFL-F_BASE[adj]: _cat = adj; }

        | "invariant"
          INVAR-F_BASE: { _cat = det
                        | _cat = predet };
        }.
```

```
GCN =
      GEND-F_BASE
      CASE-F_BASE
      NUM-F_BASE.


"pronouns"
PRON[_type] -->
        { "personal pronouns"
          e: _type = std;
          PRON-S_BASE
          ( "EX: Sie (polite: you)"
           CAP-DERIV-F_BASE )
          PRON-PERS-T_BASE
          { PRON-PERS-F_BASE
            PERS-F_BASE
            @(NGCI)
          | "EX: einander (each other)"
            PRON-REC-F_BASE
            INVAR-F_BASE
          }

        | "reflexive and expletive pronouns
          (without PRED value, listed in lexicon)
          EX: es (it)
          EX: mich (me)"
          e: _type = std;
          PRON-NOPRED-S_BASE
          PRON-PERS-T_BASE
          PRON-NOPRED-F_BASE
          PERS-F_BASE
          @(NGCI)

        | "quantifying pronouns, marked by .Subst
          (_type is determined by PRON-T[_type], i.e.
          morphologically marked by POS tag)"
          "EX: jemand (someone)"
          "predicative: EX: Er ist jemand, der gerne lacht."
          "(He is someone who likes to laugh.)"
          e: @(PERS 3);
          PRON-S_BASE
          ( "EX: deinesgleichen (people like you)"
           @(PNG (^REFERENT)) "features of addressee" )
```

```
          ( CAP-DERIV-F_BASE )
          PRON-T_BASE[_type]
          DISTR-SUBST_BASE
          { @(GCNI)
          | INVAR-F_BASE }

        | "EX: all dessen ((of) all that)"
          Dpre[_type]: _type = std;
          PRON[std]

        | e: _type = std;
          @(PRON-MOD)
          PRON[_type]
        }.


PRON-INC[_type] -->
        "incorporated pronouns
        EX: deretwegen (because of her/them)"
        DELIM_BASE
        { "personal pronouns"
          e: _type = std;
          PRON-S_BASE
          PRON-PERS-T_BASE
          { PRON-PERS-F_BASE
            PERS-F_BASE
            @(NGCI)
          | PRON-REC-F_BASE
            INVAR-F_BASE
          }

        | "quantifying pronouns, marked by .Subst"
          PRON-S_BASE
          PRON-T_BASE[_type]
          DISTR-SUBST_BASE: @(PERS 3);
          { @(GCNI)
          | INVAR-F_BASE }
        }.

GCNI =
      GEND-F_BASE
      CASE-F_BASE
      NUM-F_BASE
      INFL-F_BASE.
```

```
NGCI =
      NUM-F_BASE
      GEND-F_BASE
      CASE-F_BASE
      INFL-F_BASE.

PNG(_desig) =
      PERS-F_BASE: _desig = !;
      NUM-F_BASE: _desig = !;
      GEND-F_BASE: _desig = !.


"###################################
DP Periphery
###################################"

PRON-APPOS =
      { "appositive to quant pronouns"
        NP: @(APP)
            (^CHECK _LEX _PRON _APP) "marked in lexicon"
            (!PRED-RESTR) "only nominalized adjectives"
            { "either full agreement"
              "EX: niemandem Geringerem (nobody less than)"
              "EX: nichts Neues (nothing new)"
              @(CNG-AGR)
            | "or neuter nom"
              "EX: von niemandem Geringeres (of nobody less than)"
              (^NUM) = (!NUM)
              (!CASE) = nom
              (!GEND) = neut
              "EX: *von nichts Neues (of nothing new)"
              (^GEND) ~= neut
            };

      | "appositive to personal pronouns"
        e: PersAppos $ o::* "dispreferred (STOPPOINT)";
        NP: @(APP)
            (^PRON-TYPE) =c pers
            { @(PERS-NO 3)
              @(CNG-AGR) "full agreement"
            | "or else: polite form 3rd plural"
              "EX: *er Armer (he poor)"
              "EX: Sie Idiot (you idiot)"
              @(PERS 3)
              (^CHECK _MORPH _CAPITAL)
```

```
              "and no number agreement"
              (^CASE) = (!CASE)
              (^GEND) = (!GEND)
            }
            "EX: *meiner Idioten (I idiot (genitive))"
            @(CASE-NO gen)
            (!NTYPE NSYN) ~= proper "no names"

            { "optional: weak inflection in plural"
              "EX: wir Deutschen (us Germans)
              (vs. EX: wir Deutsche (us Germans))"
              @(NUM pl)
              @(PERS-NO 3)
              { "check if adjective is present"
                (!CHECK _MORPH _DERIV _ADJ)
              | (!CHECK _NCONSTR _ADJ-ATTR) }
              (!INFL) = strong-det
            };
      }.


DPpost =
      [
        ( { "genitive DP"
            "EX: das Auto Karls (Karl's car)"
            DP[std]: @(DPpost-GENITIVE)
          | "alternatively von-PP"
            "EX: das Auto von Karl (Karl's car)"
            PPgen[std]: (^ADJ-GEN) = !;
          } )

        [ "at most 2 PPs"
          { "EX: die Fahrt nach Bonn (the trip to Bonn)"
            PP[std]: @(DPpost-PP);
          | "EX: nettere Leute als Hans
            (people nicer than Hans)"
            PPcompar: @(DPpost-COMPAR); } ]#O#2

        ( "appositives"
          "EX: Karl, Lehrer, (Karl, a teacher,)"
          DP-APPOS: @(ADJUNCT)
                    @(ADJUNCT-TYPE appositive)
                    @(IF (!NTYPE)
                    @(PREFER-AGR ^ !));)
```

```
  [ COMMA
   { CPdep[std]: (^COMP) = !
                  "EX: die Tatsache, daß es regnet,"
                  "(the fact that it is raining)"

   | CPdep[int]: (^COMP) = !
                  "EX: die Frage, wer gewonnen hat, "
                  "(the question who won)";
   | VPinf: (^VCOMP) = !
               "EX: der Versuch, zu gewinnen,"
               "(the attempt to win)"
               "EX: sein Wunsch, respektiert zu werden,"
               "(his wish to be respected)"
   | CPdep[rel]: @(DPpost-REL)
                  "EX: der Mann, der dort steht,"
                  "(the man standing over there)";
   }
   COMMA ]#0#1

  ] - e
  "DPpost may not be empty to allow for constraints on pronouns"
  e: (^PRON-TYPE) ~= rel
    (^PRON-TYPE) ~= refl
    (^PRON-TYPE) ~= inh-refl_.


DP-APPOS -->
    "punctuations surround appositive plus optional adverbs
    note: need extra node DP-APPOS to record form of punctuation"
    { COMMA: (^PAREN-FORM) = comma;
      @(DP-APPOS-HEAD) "appositive"
      COMMA

    | LEFT-PAREN: (^PAREN-FORM) = paren;
      @(DP-APPOS-HEAD) "appositive"
      RIGHT-PAREN
    }.


DP-APPOS-HEAD =
    "optional adverb"
    "EX: Kinder, darunter auch Babys,
     (children, among them babies as well,)"
    ( ADVapp: @(ADJUNCT) )
```

```
  { "EX: Hans, Lehrer aus Stuttgart,
     (Hans, teacher from Stuttgart,)"
    "EX: Hans, der beste Lehrer aus Stuttgart,
     (Hans, the best teacher from Stuttgart,)"
    DP[std]
    e: DPAppositive $ o::*; "dispreferred"

  | "EX: Karl Schmidt (64)"
    CARD }.

----
```

D.2   Templates

```
STANDARD GERMAN TEMPLATES (1.0)

"#################################
DP core templates
#################################"

DP-COMPLETE =
        "(i) determiner features"
        @(COMPLETE (^SPEC INT))
        @(COMPLETE (^SPEC POSS))
        @(COMPLETE (^SPEC QUANT))
        @(COMPLETE (^SPEC AQUANT))
        @(COMPLETE (^SPEC NUMBER))

        @(COMPLETE (^CHECK _NMORPH _GENITIVE))

        "might be introduced by contracted dets, e.g. 'zum':
        @(COMPLETE (^SPEC DET))
        @(COMPLETE (^CHECK _SPEC-TYPE _COUNT))
        @(COMPLETE (^CHECK _SPEC-TYPE _DEF))
        @(COMPLETE (^CHECK _SPEC-TYPE _DET))
        @(COMPLETE (^INFL))"

        "(ii) pronoun features"
        @(COMPLETE (^PRON-TYPE))
        @(COMPLETE (^PRON-FORM))
        @(COMPLETE (^CHECK _MORPH _CAPITAL))
        @(COMPLETE (^CHECK _LEX _PRON _APP))

        "(iii) noun features"
        @(COMPLETE (^NTYPE))
        @(COMPLETE (^CHECK _MORPH _DERIV _ADJ)) "nominalized adjs"
        @(COMPLETE (^CHECK _NCONSTR _MEASURE))
        @(COMPLETE (^CHECK _NCONSTR _ADJ-ATTR)) "attributive adjs"
        @(COMPLETE (^MOD))

        "partly assigned in rules, e.g.
        - NSEM COMMON = mass in N-APPOS-MEASURE
        - NSEM PROPER = last_name in NAME-APPOSITIVE
        @(COMPLETE (^NSEM COMMON))
        @(COMPLETE (^NSEM PROPER))" .
```

```
PRENOM-GENITIVE =
        (^SPEC POSS) = !
        "EX: Karls Hund (Karl's dog)"
        @(CASE_desig ! gen)
        "must be morphologically marked:"
        "EX: *Karl Hund (Karl dog)".


RIGHT-SISTER =
        "'attributive' determiners, want right sister
        (NP or Aquant)"
        @(IF (^CHECK _SPEC-TYPE _DET) = attr
        (* RIGHT_SISTER) ).

RIGHT-SISTER-CARD =
        "'ein' as CARD, wants right sister
        (NP or Aquant)"
        @(IF (^CHECK _SPEC-TYPE _DET) = eine_card
        (* RIGHT_SISTER) ).

CHECK-SPEC(_desig) =
        "called by singular count nouns"

        { "either accompanied by some licensing determiner"
          (_desig CHECK _SPEC-TYPE _COUNT)

        | "else:"
          ~(_desig CHECK _SPEC-TYPE _COUNT)
          { "embedded by preposition"
            "EX: Hunde ohne Maulkorb (dogs without muzzle)"
            ((OBJ _desig) PTYPE)

          | "embedded in appositive"
            "EX: Lehrer Schmidt kommt. (Teacher Schmidt is coming.)"
            (APP $ _desig)
          | "EX: Schmidt, Lehrer aus Stuttgart, kommt."
            "(Schmidt, a teacher from Stuttgart, is coming.)"
            (_desig ADJUNCT-TYPE) =c appositive

          | "embedded in predicative (restricted to copula)"
            "EX: Hans ist Lehrer. (Hans is a teacher.)"
            (_desig XCOMP-TYPE) =c copular

          | "embedded in imperative"
            "EX: Scanner kalibrieren (Calibrate the scanner)"
```

```
             { ((OBJ _desig) STMT-TYPE) =c imperative
             | ((OBJ2 _desig) STMT-TYPE) =c imperative }
             TechnicalConstr $ o::*
          }
        }.


INFL-NOMINAL(_infl) =
        "for det/adj inflection"
        (^INFL) = _infl.


CNG-AGR =
        "case, number and gender agreement
        called in pronoun appositive rule"
        (^CASE) = (!CASE)
        (^NUM) = (!NUM)
        (^GEND) = (!GEND).


"#################################
DPpost templates
#################################"

DPpost-GENITIVE =
        "EX: der wichtigste der oben beschriebenen Schritte
         (the most important one of the steps described above)
        EX: keines der Kinder (none of the children)
        EX: welches der Kinder (which of the children)"
        (^ADJ-GEN) = !
        @(CASE_desig ! gen)
        "EX: *wer der Kinder (who [the children GEN])"
        (^PRON-TYPE) ~= int
        @(IF (^PRON-TYPE)
        (!SPEC) "EX: *der kleiner (this small[GEN,PL])"
        ).

DPpost-PP =
        (!PRED FN) ~= von "treat von-PP as PPgen instead"

        "note: currently only PP adjuncts can occur here
        (since the current lexicon does not contain nouns
        subcategorizing for PP obliques)"
        @(ADJUNCT)
        PPAdjunct $ o::* "dispreferred"
```

```
        @(NO-POST-CIRCUM !) "only prepositions here"

        "most directional PPs are actually subcategorized,
        hence no proper nouns allowed here"
        "EX: die Fahrt nach Bonn (the trip to Bonn)"
        "EX: der Brief an den Vater (the letter to the father)"
        @(IF (^NTYPE NSYN) = proper
        (!OBJ CASE) ~= acc
        )

        @(IF (^PRON-TYPE) = pers
        "EX: wir aus Athen (us from Athens)"
        PersWithPP $ o::* "dispreferred"
        ).

DPpost-COMPAR =
        { "EX: nettere Leute als Hans"
          "(people nicer than Hans)"
          (^ADJ-COMPAR COMPAR OBL-COMPAR) = !
        | "EX: ein Besserer als ich"
          "(someone better than me)"
          "EX: jemand Besseres als ich"
          "(someone better than me)"
          (^ (APP $) PRED-RESTR COMPAR OBL-COMPAR) = !
        | "EX: mehr Leute als gestern"
          "{E: more people than yesterday"
          (^SPEC {QUANT|AQUANT} COMPAR OBL-COMPAR) = ! }.

DPpost-REL =
        (^ADJ-REL) = !
        (^NUM) = (!PRON-REL NUM)
        (^GEND) = (!PRON-REL GEND).


"#################################
Quantifying expressions
(lexicon templates)
#################################"

DET(_stem) =
        (^SPEC DET PRED) = '_stem'.

DET-TYPE(_type) =
        (^SPEC DET DET-TYPE) = _type.
```

```
POSS(_stem) =
        (^SPEC POSS PRED) = 'pro'
        (^SPEC POSS PRON-FORM) = _stem.


DEM(_stem) =
        (^SPEC DEM PRED) = '_stem'.


QUANT(_stem) =
        (^SPEC QUANT PRED) = '_stem'.


AQUANT(_stem) =
        (!PRED) = '_stem'
        ! $ (^SPEC AQUANT).


INT(_stem) =
        (^SPEC INT PRED) = '_stem'.


NUMBER(_stem) =
        (^SPEC NUMBER PRED) = '_stem'.


SPEC-NUMBER-TYPE(_type) =
        (^SPEC NUMBER NUMBER-TYPE) = _type.


AMBIG-INFL =
        @(IF (^INFL) = strong-det
        QuantAsDet $ o::* "preferred"
        ).


SPEC-COUNT =
        (^CHECK _SPEC-TYPE _COUNT) = +.


SPEC-MASS =
        @(IF (^NUM) = sg
        (^NSEM COMMON) = mass
        ).


SPEC-DEF =
        (^CHECK _SPEC-TYPE _DEF) = +.


SPEC-ADEGREE(_desig _type) =
        "specifiers that can be modified by ADVadj,
        which in turn checks for ADEGREE feature"
        (^SPEC _desig ADEGREE) = _type.
```

```
EINE-INDEF-CONDITION =
        { "(i) 'eine' preceded by a determiner marking definiteness
          (= 'eine' with strong or weak inflection)"
          "EX: der eine (this person)"
          "EX: die einen Bürger (one part of the citizens)"
          "EX: mein einer Wagen (one of my cars)"
          @(PLUS-DEF)

        | "(ii) 'eine' not preceded by a definite determiner
          then obligatorily empty NP"
          "EX: einer (someone)"
          "EX: manch einer (someone)"
          @(NO-DEF)
          @(NO-DET)
          ~(* MOTHER MOTHER RIGHT_SISTER)
          "EX: *einer rote (someone red)"
          "EX: *manch einer Mensch (someone person)"
          "EX: *einer Mensch (someone person)"
        }.


POSS-CONDITION =
        "POSS: 'meines' and 'meiniges'"
        "EX: meines, das dort steht, (mine, which is over there)"
        "EX: das meine (mine)"

        "head noun always empty (similar to EINE-INDEF-CONDITION)"
        "EX: *das meine Buch (the my book)"
        "EX: *das meine rote (the my red)"
        "EX: das meinige (mine)"
        "EX: *das meinige Buch (the my book)"
        "EX: *meinige (mine)"

        ~(* MOTHER MOTHER RIGHT_SISTER) "empty NP".


PLUS-DEF =
        { "predet + a quantifying expressions marked as definite"
          "EX: alle die Menschen, die vor dem Krieg flüchten,"
          "(all these people fleeing from the war)"
          "EX: alle diese Dinge (all these things)"
          "EX: alle meine Kollegen (all my colleagues)"
          (^CHECK _SPEC-TYPE _DEF)

        | "predet + a demonstrative pronoun"
          "EX: all derer (of all those)"
          (^PRON-TYPE) =c demon
```

```
          | "predet + 'beide'"
            "EX: alle beide (both)
            (special: 'beide' is SPEC NUMBER here because
            'alle' occupies SPEC QUANT)"
            (^SPEC NUMBER PRED FN) =c beide

          | "predet + pronoun 'beide'"
            "EX: alles beides (both)"
            (^PRON-FORM) =c beide
          }.

PLUS-EINE =
          (^SPEC {DET|NUMBER} PRED FN) =c eine.

NO-DEF =
          ~(^CHECK _SPEC-TYPE _DEF).

NO-DET =
          ~(^SPEC DET DET-TYPE).

SEM-ETWAS(_desig _sem) =
          (_desig SEM-ETWAS) = _sem.

"##################################
Pronouns
(lexicon templates)
#################################"

PRON(_stem) =
          @(PRED pro)
          @(PRON-FORM _stem).

PRON-NOPRED(_stem) =
          ~(^PRED)
          @(PRON-FORM _stem)
          NoPred $ o::* "preferred".

PRON-FORM(_stem) =
          (^PRON-FORM) = _stem.

PRON-FORM_desig(_desig _stem) =
          (_desig PRON-FORM) = _stem.
```

```
PRON-TYPE(_type) =
          @(PRON-TYPE_desig ^ _type).

PRON-TYPE_desig(_desig _type) =
          (_desig PRON-TYPE) = _type
          "feature checked in subcat templates"
          (_desig NTYPE NSYN) = pronoun

          "experimental
          { pronouns that scramble easily (in the Wackernagel position)
           'light pronouns'
           TODO: add 'das' as (optional) light pronoun?
           { _type = pers
           | _type = refl
           | _type = inh-refl_
           | _type = expl }
           (^CHECK _LEX _PRON _LIGHT) = +

          | _type ~= pers
            _type ~= refl
            _type ~= inh-refl_
            _type ~= expl
          }".


SUBJ-REFL-AGR =
          "forces PERS and NUM agreement between subject and reflexives"
          "EX: Er wäscht sich. (He washes himself.)
          (better check agreement here than in subcat templates,
          covers semantic reflexives as well)"
          { ( (OBJ ^) SUBJ) = %REF
          | ( (OBJ2 ^) SUBJ) = %REF
          | ( (OBL OBJ ^) SUBJ) = %REF
          | ( (OBLsem OBJ ^) SUBJ) = %REF
          | ( (ADJUNCT $ OBJ ^) SUBJ) = %REF }

          (%REF PERS) = (^PERS)
          (%REF NUM) = (^NUM)

          "EX: Sich zu waschen, ist wichtig. (It is important to wash.)
          only singular analysis (to avoid ambiguities)"
          { (%REF NUM) =c pl
          | (%REF NUM) = sg }.
```

```
PRON-EXPL(_stem) =
        "expletive EX: Es regnet. (It is raining.)"
        @(PRON-NOPRED _stem)
        @(PRON-TYPE expl_)
        @(NUM sg) @(PERS 3) "for verb agreement"
        @(GEND neut)
        @(CASE-NO gen) @(CASE-NO dat) "for dummy objects".


PRONquant-APPOS-MASC(_stem) =
        @(PRONquant-APPOS _stem)
        @(GEND masc) @(NUM sg).

PRONquant-APPOS-NEUT(_stem) =
        @(PRONquant-APPOS _stem)
        @(GEND neut) @(NUM sg).

PRONquant-APPOS(_stem) =
        @(PRED pro)
        @(PRON-FORM _stem)
        (^CHECK _LEX _PRON _APP) = +.



NULL(_desig) =
        "for null referents (= empty head of adjectives/quants)"
        { (_desig PRED) = 'pro'
        | "possibly predicative"
          (_desig PRED) = 'pro<(_desig SUBJ)>'
          (_desig XCOMP-TYPE)
        }
        @(PRON-TYPE_desig _desig null).

NULL-NOT-PREDIC(_desig) =
        (_desig PRED) = 'pro'
        @(PRON-TYPE_desig _desig null).


"#################################
Called by verb subcat templates
#################################"

PRON-ES(_desig) =
        "check for pronoun 'es'"
        (_desig PRON-FORM) =c sie
```

```
        (_desig PRON-TYPE) =c pers
        (_desig GEND) = neut
        (_desig PERS) = 3
        (_desig NUM) = sg
        (_desig CASE) ~= gen
        (_desig CASE) ~= dat.


CORR-RESTR(_desig _type) =
        "check for correlative 'es', which functions as the object
        (or subject in passivized sentences),
        clause functions as appositive to 'es'
        (similar to pronominal adverbs, cf. @OBL-RESTR)
        EX: Ihm wurde es überlassen, ob er kommen wollte.
         (It was left to him (to decide) whether he wanted to come.)"
        @(PRON-ES _desig)

        "feature checked by appositive clause"
        (_desig CHECK _VLEX _APP-CLAUSE) = _type

        "check for presence and correct form of appositive clause"
        (_desig APP-CLAUSE)
        @(CHECK-FORM (_desig APP-CLAUSE) _type).


----
```

D.3   Lexicon Entries

```
STANDARD GERMAN LEXICON (1.0)

"#################################
Quantifying expressions
#################################"

"#################################
Articles
#################################"

eine_art  !ART-S xle "article: uninflected in nom.sg.masc/neut"
                     @(DET eine)
                     EinAsDet $ o::* "preferred"
                     "to disprefer ditransitive analysis of"
                     "EX: Er schreibt einen Unsinn."
                     "(He writes a nonsense.)"; ONLY.


eine_indef !INDEF-S[adj] xle "always fully inflected 'eine', functions
                             as SPEC NUMBER to allow for multiple dets"
                             "EX: die einen Bürger"
                             "(one part of the citizens)"
                             "EX: Einer kam. (One person came.)"
                             "EX: manch einer (someone)"
                             @(NUMBER eine)
                             @(EINE-INDEF-CONDITION); ONLY.


eine_card !Dcard-S xle "same forms as the article, replaces '1'"
                       "EX: ca. eine Stunde (about one hour)"
                       "EX: ein Liter Wasser (one liter of water)"
                       @(NUMBER eine)

                       (^CHECK _SPEC-TYPE _DET) = eine_card
                       "feature corresponding to _DET = attr"

                       EinAsCard $ o::* "STOPPOINT
                       (to avoid double analyses for 'ein Mann'
                       (note: STOPPOINT often excludes correct reading,
                       e.g. EX: Mein Pulli ist eine Nummer zu klein."
                       "(My sweater is too small by one size.)"
                       ~(^SPEC POSS NTYPE)
                       "EX: *Karls ein Hund (Karl's a/one dog)"; ONLY.
```

```
einen_hom !V-S xle "all forms of the verb 'einen' that
                    are homonymous with the indefinite article
                    get special lemma 'einen_hom': ein/eine/einen
                    -> disprefer these forms (for efficiency reasons)"
                    @(DPnom-DPacc %stem) @(AUX-HABEN)
                    EinenAsVerb $ o::* "NOGOOD"; ONLY.


die_art    !ART-S xle @(DET die) DieAsDet $ o::* "preferred"; ONLY.


"entry for incorporated articles as in 'zum' (to_the)"
die        !ART-S xle "EX: zum Auto (to the car)"
                      @(DET die);
           !PRON-S xle "EX: deretwegen (because of her/them)"
                       @(PRON die);
           ONLY.

"#################################
Possessives
#################################"

ihre_attr !POSS-S[det] xle "determiner: EX: mein Buch (my book)
                           (same inflection as article 'ein')"
                           @(POSS ihre) @(SPEC-COUNT); ONLY.

ihre_pro  !POSS-S[adj] xle "does not allow for overt head noun
                           (similar to 'eine' as indefinite)"
                           "EX: das meine (mine)"
                           @(POSS ihre) @(POSS-CONDITION); ONLY.


ihrige    !POSS-S[adj] xle "similar to ihre_pro:
                           does not allow for head noun"
                           @(POSS %stem) @(POSS-CONDITION); ONLY.


"#################################
Demonstratives
#################################"

selbe     !DEM-S xle "EX: in selbem Umfang (to the same extent)"
                     "EX: die selbe Sprache (the same language)
                     (note: other DEM are determiners only)"
                     @(DEM %stem) @(AMBIG-INFL); ONLY.
```

```
"###################################
Interrogatives
###################################"

was       X xle "ignore adverbial and .Attr reading from morphology,
                which are provided for the 'was fuer' construction
                (for interrogative and relative 'was':
                see was_wpro and was_rel entries)"; ONLY.


was' für  !D[int] * "EX: was für Leute (what kind of people)"
                    "EX: was für einer (what kind of person)"
                    @(INT %stem)
                    (^CHECK _SPEC-TYPE _DET) = attr
                    @(MWE);
          !Dpre[int] * "EX: was für ein Mann (what a man)"
                       @(INT %stem) @(PLUS-EINE) @(MWE);
          ONLY.


was_wpro  PRON-S xle @(PRON was); ETC.


welch     !INT-S xle @(INT %stem) @(SPEC-COUNT);
          !Dpre-S xle @(INT %stem) @(PLUS-EINE); ONLY.

welche_wpro !INT-S xle @(INT welche) @(SPEC-COUNT) @(AMBIG-INFL); ONLY.

"###################################
Indefinites
###################################"

all       !INDEF-S xle @(QUANT %stem) @(SPEC-MASS);
          !Dpre-S xle "EX: all diese Fragen"
                      "(all those questions)"
                      @(QUANT %stem) @(SPEC-MASS) @(PLUS-DEF); ONLY.

alle      !INDEF-S xle @(QUANT %stem) @(SPEC-MASS)
                       @(NO-DEF) @(AMBIG-INFL);
          !Dpre-S xle @(QUANT %stem) @(SPEC-MASS)
                      @(PLUS-DEF); ONLY.

andere    !INDEF-S xle "special: multiple indefinites"
                       "EX: einige wenige andere"
                       "(some few others)"
                       @(AQUANT %stem) @(AMBIG-INFL); ETC.
```

```
beide     !INDEF-S xle "EX: alle beide (both)
                        special – like cardinals"
                       @(NUMBER %stem) @(AMBIG-INFL);
          =PRON-S xle "EX: alles beides (both)
                       ('beides' = .Subst)"; ONLY.


etwas     !INDEF-S xle @(QUANT %stem) @(SPEC-MASS) @(NUM sg)
                       @(SEM-ETWAS (^SPEC QUANT) a_little)
                       { "allows for modification by 'so'"
                         "EX: so etwas Dummes"
                         "(such a stupid thing)"
                         @(SPEC-ADEGREE QUANT base_)
                         (^SPEC QUANT COMPAR PRED FN) =c so };
          =PRON-S xle;
          =ADV-S xle; ONLY.


einige    !INDEF-S xle @(QUANT %stem)
                       @(NO-DEF) @(AMBIG-INFL); ONLY.

etliche   !INDEF-S xle @(QUANT %stem)
                       @(NO-DEF) @(AMBIG-INFL); ONLY.



irgendeine !INDEF-S[det] xle @(QUANT %stem) @(SPEC-COUNT);
           =PRON-S xle "indefinite pronoun"; ONLY.

jede      !INDEF-S[det] xle @(QUANT %stem) @(SPEC-COUNT);
          !INDEF-S[adj] xle @(QUANT %stem) @(SPEC-COUNT)
                            @(PLUS-EINE); ONLY.

jedwede   !INDEF-S xle @(QUANT %stem) @(SPEC-COUNT)
                       @(AMBIG-INFL); ONLY.

keine     !INDEF-S[det] xle @(QUANT %stem) @(SPEC-COUNT);
          =PRON-S xle "indefinite pronoun"; ONLY.

manch     !INDEF-S xle "EX: manch Wissenschaftler"
                       "(some researcher)"
                       "EX: manch einer (someone)"
                       @(QUANT %stem) @(SPEC-COUNT);
```

```
          !Dpre-S xle "EX: manch ein Wissenschaftler"
                      "(some researcher)"
                      @(QUANT %stem) @(PLUS-EINE); ONLY.


manche    !INDEF-S xle @(QUANT %stem) @(SPEC-COUNT)
                      @(NO-DEF) @(AMBIG-INFL)
                      "EX: ein mancher (someone)"; ONLY.


"TODO: '30 Prozent mehr Geld' (similar to adjectives like 'alt')"
mehr      !INDEF-S xle "EX: mehr gestohlenes Geld"
                      "(more stolen money)"
                      @(QUANT %stem)
                      @(SPEC-ADEGREE QUANT comparative_) "'viel mehr'"
                      { "optionally: subcategorize for OBL-COMPAR"
                        (!PRED) = '%stem<(!OBL-COMPAR)>' "like .Comp"
                        (^SPEC QUANT COMPAR) = !
                        "EX: mehr Leute als erwartet"
                        "(more people than expected)
                        note: two PRED values are introduced here,
                        SPEC QUANT PRED and SPEC QUANT COMPAR PRED.
                        The second one facilitates the treatment of
                        OBL-COMPAR (since 'mehr' then be treated like
                        comparative adjectives, similar to constraints
                        introduced by the tag .Comp"
                      };
          =PARTadj *;
          =ADV-S xle;
          ONLY.



mehrere   !INDEF-S[adj] xle @(QUANT %stem) @(NO-DEF); ONLY.


meiste    !INDEF-S xle @(QUANT %stem) @(SPEC-MASS) @(PLUS-DEF)
                      "EX: bei weitem die meisten Leute"
                      "(most people by far)"
                      "EX: die bei weitem meisten Leute"
                      "(most people by far)"
                      @(SPEC-ADEGREE QUANT superlative); ONLY.


reichlich !INDEF-S xle @(QUANT %stem) @(SPEC-MASS)
                      "special feature that allows for modification
                      by certain adverbs EX: besonders reichlich"
                      "(especially ample)"
                      @(SPEC-ADEGREE QUANT base_); ETC.
```

```
solch     =INDEF-S xle "EX: solch Unsinn (such nonsense)"
                      @(QUANT %stem) @(SPEC-MASS);
          !Dpre-S xle @(QUANT %stem) @(PLUS-EINE);
          =ADV-S xle; ONLY.


solche    !INDEF-S xle "EX: kein solches Geld (no such money)"
                      @(AQUANT %stem); ONLY.


so        !INDEF-S xle "EX: so einer (such a person)"
                      @(QUANT %stem) @(SPEC-MASS)
                      SoAsDet $ o::* "dispreferred";
          =ADV-S xle;
          =A[-infl] *;
          =PARTadj *; ONLY.


viele     !INDEF-S xle "AQUANT: for multiple determiners"
                      "EX: all die vielen Leute"
                      "(all those many people)"
                      @(AQUANT %stem)
                      @(SPEC-MASS) @(AMBIG-INFL)
                      @(SPEC-ADEGREE AQUANT base_); ONLY.


viel      !INDEF-S xle @(QUANT %stem)
                      @(SPEC-ADEGREE QUANT base_);
          =ADV-S xle;
          =ADVadj *;
          ONLY.


wenige    !INDEF-S xle "for multiple determiners"
                      "EX: einige wenige (some few)"
                      @(AQUANT %stem)
                      @(SPEC-MASS) @(AMBIG-INFL)
                      @(SPEC-ADEGREE AQUANT base_); ONLY.


wenig     !INDEF-S xle @(QUANT %stem)
                      @(SPEC-ADEGREE QUANT base_);
          =ADV-S xle;
          =ADVadj *;
          ONLY.


weniger   !INDEF-S xle "EX: weniger Leute (less people)"
                      @(QUANT %stem)
                      QuantAsDet $ o::* "preferred (otherwise
                      lemma 'wenige' would win always)"
                      @(SPEC-ADEGREE QUANT comparative_)
```

```
                        "EX: viel weniger (much less)"
                        { "optionally: subcategorize for OBL-COMPAR"
                          (!PRED) = '%stem<(!OBL-COMPAR)>' "like .Comp"
                          (^SPEC QUANT COMPAR) = ! };
              =PARTadj *;
              =ADV-S xle;
              ONLY.


ein' bißchen D[std] * @(QUANT %stem) @(MWE); ETC "ETC: adverb".
ein' bisschen D[std] * @(QUANT %stem) @(MWE); ETC "ETC: adverb".
ein' wenig D[std] * @(QUANT %stem) @(MWE); ETC "ETC: adverb".
ein' paar D[std] * @(QUANT %stem) @(NUM pl) @(MWE); ONLY.


genug    D[std] * @(QUANT %stem) @(SPEC-MASS); ETC.
genügend  D[std] * @(QUANT %stem) @(SPEC-MASS); ETC.


----
STANDARD GERMAN LEXICON (1.0)

"#################################
Pronouns
#################################"

"#################################
Relative pronouns
#################################"

die_rel   !PRON-S xle @(PRON die)
                      DieAsRelPron $ o::* "preferred"
                      "prefer relative pronoun to (full form)
                      demonstrative pronoun in:"
                      "EX: Hans sieht die Frau, die kommt."
                      "(Hans sees the woman who is coming.)"; ETC.

was_rel   !PRON-S xle @(PRON was); ETC.

welche_rel !PRON-S xle @(PRON welche)
                      WelcheAsRelPron $ o::* "dispreferred"; ETC.
```

```
"#################################
Interrogative pronouns
#################################"

wer       !PRON-S xle "interrogative pron, assign masculine
                      for anaphora and left dislocated gender agreement"
                      @(PRON %stem) @(GEND masc); ETC.


wessen    !PRONspec[int] * @(PRON wer) @(PRON-TYPE int)
                      @(GEND masc) @(NUM sg) @(CASE gen)
                      (^CHECK _NMORPH _GENITIVE) = +; ETC.


"#################################
Demonstrative pronouns
#################################"

die_dem   !PRON-S xle @(PRON die)
                      DieAsDemPron $ o::* "treat lemma as NOGOOD,
                      frequent forms handled by full form entries"; ETC.


"full form entries for demonstrative 'die'"
die       !PRON[std] * @(PRON die) @(PRON-TYPE demon) @(PERS 3)
                      { @(GEND fem) @(NUM sg)
                        @(CASE-NO gen) @(CASE-NO dat)
                      | @(NUM pl) @(CASE-NO gen) @(CASE-NO dat) }; ETC.


das       !PRON[std] * @(PRON die) @(PRON-TYPE demon) @(PERS 3)
                      @(GEND neut) @(NUM sg)
                      @(CASE-NO gen) @(CASE-NO dat);
              !PREDPes * "special: 'das' used as PREDP: may occur as
                      the first constituent in the Mittelfeld"
                      "EX: weil das Kinder sind"
                      "(because these are children)"
                      @(SUBJ_subc pro)
                      @(PRON-FORM das) @(PRON-TYPE pers);
              ETC.

der       !PRON[std] * @(PRON die) @(PRON-TYPE demon) @(PERS 3)
                      @(GEND masc) @(NUM sg) @(CASE nom); ETC.
```

```
dessen    !PRON[std] * @(PRON die) @(PRON-TYPE demon) @(PERS 3)
                       @(GEND-NO fem) @(NUM sg)
                       @(CASE gen)
                       (^CHECK _NMORPH _GENITIVE) = +;
          !PRONspec[std] * @(PRON die) @(PRON-TYPE demon) @(PERS 3)
                       @(GEND-NO fem) @(NUM sg)
                       @(CASE gen)
                       (^CHECK _NMORPH _GENITIVE) = +;
          !PRONspec[rel] * @(PRON die) @(PRON-TYPE rel) @(PERS 3)
                       @(GEND-NO fem) @(NUM sg)
                       @(CASE gen)
                       (^CHECK _NMORPH _GENITIVE) = +;
          ETC.

deren     !PRON[std] * @(PRON die) @(PRON-TYPE demon) @(PERS 3)
                       @(CASE gen)
                       (^CHECK _NMORPH _GENITIVE) = +
                       { @(GEND fem) @(NUM sg) | @(NUM pl) };
          !PRONspec[std] * @(PRON die) @(PRON-TYPE demon)
                       @(PERS 3) @(CASE gen)
                       (^CHECK _NMORPH _GENITIVE) = +
                       { @(GEND fem) @(NUM sg) | @(NUM pl) };
          !PRONspec[rel] * @(PRON die) @(PRON-TYPE rel)
                       @(PERS 3) @(CASE gen)
                       (^CHECK _NMORPH _GENITIVE) = +
                       { @(GEND fem) @(NUM sg) | @(NUM pl) };
          ETC.

derer     !PRON[std] * @(PRON die) @(PRON-TYPE demon) @(PERS 3)
                       @(CASE gen)
                       (^CHECK _NMORPH _GENITIVE) = +
                       { @(GEND fem) @(NUM sg) | @(NUM pl) };
          ETC.


"#################################
Indefinite Pronouns
#################################"

das' alles !PRON[std] * "(should probably be analyzed as
                       quantifier-floating)"
                       @(PRON %stem) @(PRON-TYPE indef)
                       @(NUM sg) @(GEND neut)
                       @(CASE-NO gen) @(CASE-NO dat) @(MWE); ETC.
```

```
dies' alles !PRON[std] * @(PRON %stem) @(PRON-TYPE indef)
                       @(NUM sg) @(GEND neut)
                       @(CASE-NO gen) @(CASE-NO dat) @(MWE); ETC.

welche_indef !PRON-S xle @(PRON welche)
                       WelcheasQuant $ o::* "dispreferred"; ETC.


"#################################
Pronouns + Appositives
#################################"

etwas     !PRON-S xle @(PRONquant-APPOS-NEUT %stem)
                       @(SEM-ETWAS ^ something); ETC.

irgendetwas !PRON-S xle @(PRONquant-APPOS-NEUT %stem); ETC.
irgendwas !PRON-S xle @(PRONquant-APPOS-NEUT %stem); ETC.
nichts    !PRON-S xle @(PRONquant-APPOS-NEUT %stem); ETC.
sonstwas  !PRON-S xle @(PRONquant-APPOS-NEUT %stem); ETC.

was_indef !PRON-S xle @(PRONquant-APPOS-NEUT was)
                       WasAsQuant $ o::* "dispreferred"; ETC.


jemand    !PRON-S xle @(PRONquant-APPOS-MASC %stem); ETC.
irgendjemand !PRON-S xle @(PRONquant-APPOS-MASC %stem); ETC.
irgendwer !PRON-S xle @(PRONquant-APPOS-MASC %stem); ETC.
niemand   !PRON-S xle @(PRONquant-APPOS-MASC %stem); ETC.
sonstjemand !PRON-S xle @(PRONquant-APPOS-MASC %stem); ETC.


"#################################
NOPRED pronouns
#################################"

"note: the ordinary pronoun 'es' is covered by the
entry of 'sie' (= the lemma form of 'es')"
es        !PRON[std] * "special uses of 'es' -> want [PRON-FORM es]
                       (rather than [PRON-FORM sie])"
                       "EX: Es regnet. (It is raining.)"
                       @(PRON-EXPL %stem);
          !VORF-ES * "EX: Es wird getanzt. (People are dancing.)"
                       @(PRON-EXPL %stem);
          !PREDPes * "special: 'es' used as PREDP: may occur as
                       the first constituent in the Mittelfeld"
```

```
              "EX: weil es Kinder sind"
              "(because they are children)"
              @(SUBJ_subc pro)
              @(PRON-FORM es) @(PRON-TYPE pers);
        ONLY.

sie     !PRON-NOPRED-S xle @(PRON-NOPRED _stem); ETC.

----
```

Part IV

Chapter 12

Conclusion

In this dissertation, we discussed the importance of maintainability and documentation in grammar development. A modular and transparent design of the grammar and detailed documentation are prerequisites for various grammar applications, for further grammar modifications (e.g. with respect to performance or coverage), and for reusability of the grammar code in general.

Modularity We argued that a grammar which implements a linguistic theory ought to encode linguistic generalizations explicitly. By "explicitly" we mean that generalizations, i.e. (statements about) properties that are common to different constructions, are to be encoded by so-called "grammar modules". A grammar module consists of a coherent piece of code, which encodes such common properties and, in this sense, represents a functional unit. In a modularized grammar, all constructions that share a certain property make use of the same grammar module to encode this property.

Grammar development environments, such as XLE, provide various means of abstraction to modularize the grammar code, e.g. macros and templates.

Maintainability The use of grammar modules is important for two reasons. First, a grammar that models a linguistic theory ought to represent linguistic generalizations in an appropriate way.

Second, a modularized grammar favours maintainability of the code. If a certain functionality of the grammar is to be modified, the grammar writer ideally only has to modify the code within the module encoding that functionality. Hence, if the analysis of a certain phenomenon is modified, all constructions that adhere to the same principles are affected as well, automatically.

Transparency In addition, a modularized grammar favours maintainability by an increase in code transparency. To illustrate this we introduced two notions of transparency, "intensional" and "extensional" transparency.

331

Intensional transparency of grammar code means that the characteristic, defining properties of a construction are encoded by means of suitable modules (macros/templates), i.e in terms of generalizing definitions. Hence, properties that are characteristic of different constructions are encoded by the same module. Grammar modules thus make the intentions of the grammar writer transparent. They allow for distinguishing between intended (motivated) and arbitrary similarities of pieces of code.

Extensional transparency means that linguistic generalizations are stated "extensionally", i.e. the grammar writer does not make use of abstraction means to assemble pieces of code that belong together. Generalizations thus remain implicit, which makes the grammar code error-prone.

The criterion of maintainability clearly favours intensional over extensional transparency. However, we showed that the use of macros and templates may result in a decreased readability, at least for casual users of the grammar. The decreased readability results from grammar-specific properties that distinguish a grammar implementation from other software projects.

Grammar-specific properties   We argued that a grammar differs from other software in two (related) aspects. First, the grammar code itself represents important information in that it encodes linguistic analyses. That means, besides the grammar writer, other people, e.g. linguists or users of the grammar, are interested in the details of the implementation.

Second, similar to software modules, grammar modules represent functional units in that they encode linguistic generalizations. Yet, in contrast to software modules, certain of the grammar modules are not black boxes because their internal structure and content is relevant to the outside. For instance, the functionality of a certain rule cannot be really understood without the definitions of all the dependent modules (i.e. all macros and templates that are called by this rule).

Moreover, grammar modules are often structured hierarchically. That is, the grammar user/writer has to follow long chains of macro and template calls and look up the respective definitions.

Hence, the consequent use of modules results in a decreased readability of the code. However, we argued that this can be compensated for by a special documentation technique.

Grammar documentation   User-friendly documentation needs to be independent of the structure of the grammar code. In this dissertation, we presented such a type of documentation. The proposed documentation technique, based on XML and XSLT, is flexible enough to provide a transparent picture of the grammar modules, despite their interactive nature.

Our documentation technique moreover supports grammar development in various aspects, e.g. by providing different types of indices or testsuites that are generated on the basis of all the example sentences in the documentation. This allows the grammar writer to easily compare the phenomena described in the documentation and the actual coverage of the grammar implementation.

Documentation maintainability   Due to the complexity of a grammar implementation, the documentation represents a complex object as well. Hence, maintainability of the documenation becomes an important issue.

We discussed some techniques that could be applied in the maintenance of our XML-based documentation. However, it is clear that only a user-friendly documentation tool can guarantee consistency and up-to-dateness of the documentation. (After any grammar modification, the grammar writer ought to revise and update all parts of the documentation that may be affected by the grammar modification.)

We believe that—as a prerequisite both for grammar development and reusability—grammar development platforms ought to include special tools for grammar documentation.

Outlook   Our XML-based documentation technique is a very powerful means that can be exploited to support the difficult task of grammar (and documentation) development in various further ways.

For instance, the grammar code can be "translated" to a pure XML document, i.e. each atomic element of the code (syntactic categories such as DP, f-structure designators, e.g. ↑, XCOMP, *) is marked by a tag. This markup can be used in various ways.

- The grammar code can be displayed with refined highlighting, e.g. c-structure and f-structure elements can be printed in different colours. This improves the transparency and readability of the code.

- The grammar code can be mapped to a representation that uses annotated trees instead of rules. This may result in a better understanding of the code. (Note, however, that the mapping to the annotated-tree representation is not trivial, since c-structure rules make use of regular expressions.)

- The markup allows for various types of links between different parts of the code. For instance, each template call can be linked to the definition of the template (and vice versa). Each feature can be linked to its declaration in the feature declaration section, etc.

● The markup also allows for further links between the documentation and the grammar code. For instance, each reference to a template in the documentation can be linked to the definition of the template in the code.

Implementation of the German DP   Finally, we presented the implementation and documentation of the German DP in the German Pargram grammar. We hope that our implementation illustrates the consequent use of macros and templates to encode linguistic generalizations, e.g. to encode common properties of determiners and quantifying adjectives.

Our documentation of the German DP motivates the implemented analysis and explains all details of the implementation at code-level. It further includes extensive corpus studies of the implemented phenomena. At the same time, it presents a linguistic introduction to the German DP and, moreover, proposes a formalization within the framework of LFG.

Our implementation of the German DP illustrates typical problems that grammar writers are confronted with. We hope that this dissertation made clear the importance of modularity, transparency, and documentation in the implementation of a grammar.

Bibliography

Abney, S. (1987). *The English Noun Phrase in its Sentential Aspect*. PhD thesis, MIT, Cambridge, MA.

Becker, T. (2001). DMOR: Handbuch. Technical report, IMS, University of Stuttgart, Germany.

Beesley, K. R. and Karttunen, L. (2003). *Finite-State Morphology: Xerox Tools and Techniques*. Studies in Computational Linguistics. Stanford, CA: CSLI Publications.

Berman, J. (2001). *Topics in the Clausal Syntax of German*, vol. 7(2) of *AIMS (Arbeitspapiere des Instituts für Maschinelle Sprachverarbeitung)*. University of Stuttgart.

Berman, J. and Frank, A. (1996). *Deutsche und französische Syntax im Formalismus der LFG*. No. 344 in Linguistische Arbeiten. Tübingen: Niemeyer.

Bhatt, C. (1990). *Die syntaktische Struktur der Nominalphrase im Deutschen*, vol. 38 of *Studien zur deutschen Grammatik*. Tübingen: Narr.

Börjars, K. (1998). Clitics, Affixes, and Parallel Correspondences. In M. Butt and T. H. King, eds., *Proceedings of the LFG98 Conference*, Brisbane, Australia. CSLI Online Proceedings. `http://csli-publications.stanford.edu/LFG/3`.

Börjars, K., Payne, J., and Chisarik, E. (1999). On the Junstification for Functional Categories in LFG. In M. Butt and T. H. King, eds., *Proceedings of the LFG99 Conference*, Manchester, UK. CSLI Online Proceedings. `http://csli-publications.stanford.edu/LFG/4`.

Bresnan, J. (2001). *Lexical-Functional Syntax*, vol. 16 of *Textbooks in Linguistics*. Oxford, UK: Blackwell.

Bröker, N. and Dipper, S. (1999). Zur Konstruktion von Lexika für die maschinelle syntaktische Analyse. In J. Gippert and P. Olivier, eds., *Multilinguale Corpora — Codierung, Strukturierung, Analyse. Proceedings of 11. Jahrestagung der GLDV*, 159–168, Frankfurt/Main. Praha: Enigma Corporation.

Butt, M., Niño, M.-E., and Segond, F. (1996). Multilingual Processing of Aux- iliaries in LFG. In D. Gibbon, ed., *Natural Language Processing and Speech Technology: Results of the 3rd KONVENS Conference*, 111–122, Bielefeld, Ger- many. Berlin/New York: Mouton de Gruyter.

Butt, M., King, T. H., Niño, M.-E., and Segond, F. (1999). *A Grammar Writer's Cookbook*. No. 95 in CSLI Lecture Notes. Stanford, CA: CSLI.

Butt, M., Dyvik, H., King, T. H., Masuichi, H., and Rohrer, C. (2002). The Parallel Grammar Project. In *Proceedings of the COLING-2002 Workshop on Grammar Engineering and Evaluation*, 1–7, Taipei, Taiwan.

Christ, O., Schulze, B. M., Hofmann, A., and König, E. (1999). The IMS Corpus Workbench: Corpus Query Processor (CQP) User's Manual. Technical report, IMS, University of Stuttgart, Germany.

Copestake, A. (2002). *Implementing Typed Feature Structure Grammars*. No. 110 in CSLI Lecture Notes. Stanford, CA: CSLI Publications.

Dalrymple, M. (2001). *Lexical Functional Grammar*, vol. 34 of *Syntax and Se- mantics*. New York et al.: Academic Press.

Dalrymple, M. and Kaplan, R. (2000). Feature Indeterminacy and Feature Res- olution. *Language*, **76**(4), 759–798.

Dalrymple, M., Kaplan, R., Maxwell, J., and Zaenen, A., eds. (1995). *Formal Issues in Lexical-Functional Grammar*. No. 47 in CSLI Lecture Notes. Stanford, CA: CSLI.

Dipper, S. (2000). Grammar-based Corpus Annotation. In A. Abeillé, T. Brants, and H. Uszkoreit, eds., *Proceedings of the COLING-2000 Workshop on Linguis- tically Interpreted Corpora LINC-2000*, 56–64, Luxembourg.

Doran, C., Egedi, D., Hockey, B. A., Srinivas, B., and Zaidel, M. (1994). XTAG System — A Wide Coverage Grammar for English. In *Proceedings of COLING- 94*, 922–928, Kyoto, Japan.

Drosdowski, G., ed. (1995). *DUDEN. Grammatik der deutschen Gegenwarts- sprache*. Mannheim et al.: Dudenverlag, fifth edition.

Eckle-Kohler, J. (1999). *Linguistisches Wissen zur automatischen Lexikon-Akqui- sition aus deutschen Textkorpora*. Berlin: Logos.

Erbach, G. (1992). Tools for Grammar Engineering. In *Proceedings of ANLP-92*, 243–244, Trento, Italy.

Erbach, G. and Uszkoreit, H. (1990). Grammar Engineering: Problems and Prospects. CLAUS Report No. 1. Report on the Saarbrücken Grammar En- gineering Workshop, University of the Saarland, Germany.

Falk, Y. N. (2001). *Lexical-Functional Grammar. An Introduction to Parallel Constraint-Based Syntax*. No. 126 in CSLI Lecture Notes. Stanford, CA: CSLI.

Frank, A. and Zaenen, A. (2002). Tense in LFG: Syntax and Morphology. In H. Kamp and U. Reyle, eds., *How We Say WHEN it Happens: Contributions to the Theory of Temporal Referene in Natural Language*, no. 455 in Linguistische Arbeiten, 17–52. Tübingen: Niemeyer.

Frank, A., King, T. H., Kuhn, J., and Maxwell, J. (2001). Optimality Theory Style Constraint Ranking in Large-Scale LFG Grammars. In P. Sells, ed., *Formal and Empirical Issues in Optimality Theoretic Syntax*, Studies in Constraint-Based Lexicalism, 367–397. Stanford, CA: CSLI Publications.

Gallmann, P. (1996). Die Steuerung der Flexion in der DP. *Linguistische Berichte*, **164**, 283–314.

Gallmann, P. and Lindauer, T. (1994). Funktionale Kategorien in Nominalphra- sen. *Beiträge zur Geschichte der Deutschen Sprache und Literatur (PBB)*, **116**, 1–27.

Haider, H. (1988). Die Struktur der deutschen Nominalphrase. *Zeitschrift für Sprachwissenschaft*, **7**(1), 32–59.

Harold, E. R. and Means, W. S. (2002). *XML in a Nutshell. A Desktop Quick Reference*. Sebastopol, CA: O'Reilly., second edition.

Helbig, G. and Buscha, J. (1993). *Deutsche Grammatik. Ein Handbuch für den Ausländerunterricht*. Leipzig et al.: Langenscheidt/Enzyklopädie, 15th edi- tion.

Kaplan, R. and Maxwell, J. (1988). Constituent Coordination in Lexical- Functional Grammar. In *Proceedings of COLING-88*, vol. 1, 303–305, Bu- dapest, Hungary. Reprinted in Dalrymple et al. 1995, pp.199–210.

Kaplan, R. and Newman, P. (1997). Lexical Resource Reconciliation in the Xerox Linguistic Environment. In D. Estival, A. Lavelli, K. Netter, and F. Pianesi, eds., *Proceedings of the ACL/EACL-98 Workshop on Computational Environments for Grammar Development and Linguistic Engineering*, 54–61, Madrid, Spain.

Kuhn, J. (1999). Towards a Simple Architecture for the Structure-Function Mapping. In M. Butt and T. H. King, eds., *Proceedings of the LFG99 Conference*, Manchester, UK. CSLI Online Proceedings. `http://csli-publications. stanford.edu/LFG/4`.

Lezius, W. (2002). *Ein Werkzeug zur Suche auf syntaktisch annotierten Textkorpora*, vol. 8(4) of *AIMS (Arbeitspapiere des Instituts für Maschinelle Sprachverarbeitung)*. University of Stuttgart.

Lezius, W., Dipper, S., and Fitschen, A. (2000). IMSLex – Representing Morphological and Syntactical Information in a Relational Database. In U. Heid, S. Evert, E. Lehmann, and C. Rohrer, eds., *Proceedings of the 9th EURALEX International Congress*, 133–139, Stuttgart, Germany.

McConnell, S. (1993). *Code Complete. A Practical Handbook of Software Construction*. Redmond, WA: Microsoft Press.

Müller, S. (1999). *Deutsche Syntax deklarativ. Head-Driven Phrase Structure Grammar für das Deutsche*. No. 34 in Linguistische Arbeiten. Tübingen: Niemeyer.

Netter, K. (1994). Towards a Theory of Functional Heads: German Nominal Phrases. In J. Nerbonne, K. Netter, and C. Pollard, eds., *German in Head-Driven Phrase Structure Grammar*, no. 46 in CSLI Lecture Notes, 297–340. Stanford, CA: CSLI.

Olsen, S. (1991). Die deutsche Nominalphrase als "Determinansphrase". In S. Olsen and G. Fanselow, eds., *DET, COMP und INFL. Zur Syntax funktionaler Kategorien und grammatischer Funktionen*, vol. 263 of *Linguistische Arbeiten*, 35–56. Tübingen: Niemeyer.

Pafel, J. (1994). Zur syntaktischen Struktur nominaler Quantoren. *Zeitschrift für Sprachwissenschaft*, **13**(2), 236–275.

Pollard, C. and Sag, I. (1994). *Head-Driven Phrase Structure Grammar*. Studies in Contemporary Linguistics. Standford, CA: CSLI/Chicago, IL: Chicago University Press.

Schiller, A., Teufel, S., Stöckert, C., and Thielen, C. (1999). Guidelines für das Tagging deutscher Textcorpora mit STTS. Technical report, IMS, University of Stuttgart, Germany, and SfS, University of Tübingen, Germany.

Schmid, H. (1994). Probabilistic Part-of-Speech Tagging Using Decision Trees. In *International Conference on New Methods in Language Processing*, 44–49, Manchester, UK.

Skut, W., Krenn, B., Brants, T., and Uszkoreit, H. (1997). An Annotation Scheme for Free Word Order Languages. In *Proceedings of ANLP-97*, 88–95, Washington, DC. Association for Computational Linguistics.

Tidwell, D. (2001). *XSLT*. Sebastopol, CA: O'Reilly.