FR 4.7 Allgemeine Linguistik
Universität des Saarlandes

A declarative grammar formalism for dependency grammar

Diplomarbeit

Angefertigt unter der Leitung von
Prof. Dr. Manfred Pinkal,
Dr. Denys Duchier and Dr. Joachim Niehren

Ralph Debusmann

19. Juli 2004

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Saarbrücken, den 19. Juli 2004.

Ralph Debusmann

Abstract

Beginning with the groundbreaking work of Chomsky in the 1950s, syntactians have concentrated mostly on the English language. But English is not a typical natural language: in particular, its word order is very rigid, as opposed to most other languages which exhibit freer word order. The phrase structure-based approach employed for the analysis of English runs into severe problems when confronted with freer word order languages.

Aside from the mainstream, linguists in Eastern Europe and Japan have pursued an approach to syntax which seemed better suited for the analysis of freer word order languages: dependency grammar. The key asset of dependency grammar is that it allows for a clean separation of syntactic dependency and surface word order. Unfortunately, none of the frameworks for dependency grammar has really caught on. We suggest two reasons for their failure: a) many of the dependency-based frameworks lack proper formalization and, perhaps surprisingly, b) most of them lack a realistic and workable account of word order.

In this thesis, we try to remedy these problems in the setting of a constraint-based approach to dependency grammar based on (Duchier 1999). We present a new account of word order for dependency grammar couched in a declarative grammatical formalism called Topological Dependency Grammar (TDG). TDG allows to cleanly separate the two levels of syntactic dependency and surface word order, which greatly facilitates the conception of grammars for freer word order languages. In addition, we can efficiently parse with TDG grammars: using a reduction described in (Duchier 2000), we achieved an efficient parser implementation using modern constraint programming techniques.

Acknowledgements

I would like to thank my professor Manfred Pinkal for having created with professor Gert Smolka and the other members of the CHORUS team a great research environment. Here, I was not only given the opportunity to work within a very research environment but also the chance to work on a subject which, at first glance, looks a little 'off-topic' in a project mostly concerned with semantics.

I would like to thank my supervisors Denys Duchier and Joachim Niehren, who helped me getting interested in the subject in the first place and gave me excellent advice whenever I needed it. I would also like to thank Denys for reading through countless drafts and for staying up all night for doing so when the thesis grew larger.

I am deeply grateful to Geert-Jan Kruijff for reading through several drafts of this thesis and providing me with extensive and extremely helpful comments. Also, a big thank you goes to Alexander Koller for advice on a lot of issues, not only with respect to the thesis itself.

I would like to thank my fellows and colleagues at Saarland University for anything: Carsten Brockmann, Mario Cattaneo, Christopher Dege, Amit Dubey, Markus Egg, Katrin Erk, Gerd Fliedner, Malte Gabsdil, Tilman Jäger, Karel Oliva, Christian Korthals, Andrea Kowalski, Ivana Kruijff-Korbayova, CJ Rupp, Joachim Sauer, Christoph Stahl, Kristina Striegnitz and Stefan Thater. This list is not supposed to be exhaustive.

Finally, thanks to my family for all their support. Without them, of course, I wouldn't even be here, let alone be able to write a thesis about dependency grammar.

Contents

Chapter 1

Introduction

In this thesis, we develop a declarative grammar formalism for dependency grammar. Contrary to most other dependency-based proposals, our formalism also includes a concise account of surface word order. Our proposal makes use of a notion of tree flattening *to derive the licensed linearizations, similar to approaches in both the paradigm of phrase structure grammar (PSG) (e.g. Reape 1994) and dependency grammar (DG) (e.g. Kahane, Nasr & Rambow 1998). In addition to tree flattening, we employ the notion of* topological fields *(Höhle 1986) to achieve a finer-grained description of word order variation. The coupling of a flattened tree structure amenable to linearization and concepts from topological fields theory positions our approach very close to recent dependency-based proposals by Bröker (1999) and Gerdes & Kahane (2001). One important benefit over these and other approaches is, however, that our formalism was from the start informed by considerations of a computational nature. As a result, its formalization can be transformed into an efficient parser implementation using a reduction described in (Duchier 2000).*

1.1 Word order variation

Word order in English is rather fixed, or *rigid* in the sense of Steele (1978). It exhibits only a limited degree of *word order variation*, similar to artificial languages such as programming languages. For that reason it is sufficient to employ for the analysis of English devices which are also used for the interpretation of artificial languages, such as context-free grammar (CFG).

Contrary to English, most other natural languages exhibit fairly *free word order*.

Especially morphologically richer languages such as Finnish and the Slavic languages show a high degree of word order variation. We call them free word order languages. Morphologically less rich languages such as Dutch and German still exhibit a fair amount of word order variation but not quite as much. Following Steele (1978) again, we call them *mixed word order languages.*[1] Notice that we will often refer to mixed and free word order as *freer word order* from now on.

Traditional phrase structure grammars are not adequate for the analysis of freer word order languages: they cannot express generalizations about word order variation but need to enumerate the possible linearizations instead. If immediate dominance (ID) and linear precedence (LP) are separated as in Generalized Phrase Structure Grammar (GPSG) (Gazdar, Klein, Pullum & Sag 1985) or Head-driven Phrase Structure Grammar (HPSG) (Pollard & Sag 1994), these approaches still run into problems when confronted with long-distance dependencies and discontinuous constituents.

The 'failure' of phrase structure grammar to account for languages with freer word order has inspired grammarians in Europe and also in Japan to seek other ways for the syntactic analysis of these languages, dependency grammar being one of them. The main difference is, as Venneman (1977) puts it, that PSG follows a *horizontal organization principle*, whereas DG follows a *vertical organization principle.* PSGs divide sentences into substrings, while DGs distinguish between *heads* and *dependents* and define no constraints on linearization. Only through additional principles such as *projectivity* (Lecerf 1960), (Hays 1964), (Gaifman 1965), (Robinson 1970) can DGs account for restrictions on possible linearizations.

DG is not the only approach that enables to account for freer word order languages. For Categorial Grammar (CG), Bach (1981) introduced the *head-wrapping*-operation to deal with discontinous constructions, and Ades & Steedman (1982) proposed the *type-shifting*-operation, which lead to the theory of Combinatorial Categorial Grammar (CCG). Descendants of Tree Adjoining Grammar (TAG) (Joshi 1987) are also equipped with means for dealing with freer-than-rigid word order languages (e.g. Becker & Rambow 1994), and so are modern variants of HPSG (Reape 1994), (Müller 1999), (Kathol 1995), (Fouvry & Meurers 2000). However it is interesting to note that in their striving for extended generative power, these formalisms make more and more use of notions borrowed from DG, most notably, the head/dependent asymmetry.

At the same time, despite the obvious appeal of DG, existing DG frameworks

[1] There are no discrete steps in the transition from rigid word order languages such as English over mixed word order languages such as German and Dutch to free word order languages (Finnish, Slavic languages). For instance, although German and Dutch both have mixed word order, word order in German is freer than in Dutch.

have not really caught on: Dependency Unification Grammar (DUG) (Hellwig 1986), Functional Generative Description (FGD) (Sgall, Hajicova & Panevova 1986), Meaning Text Theory (MTT) (Mel'čuk 1988), Lexicase (Starosta 1988), Word Grammar (WG) (Hudson 1990), all of these have not made the breakthrough into the mainstream of linguistic research. Why? For two main reasons in our opinion: First, all of these frameworks lack proper formalization in some way or other, which severely constricts their applicability and verifiability. And, perhaps surprisingly, the second reason is the very lack of a realistic and workable account of word order.

Thus, DG has its appeal, but in general lacks both a clean formalization and a workable account of word order. Our starting point is the dependency grammar outlined in (Duchier 1999). Duchier's DG already is properly formalized, in a way that enables its axiomatization to be transformed into an efficient parser implementation. Still, like many of the above mentioned DG frameworks, it lacks a sophisticated account of word order.

What we want to contribute in this thesis is a proper formal account of word order for Duchier's (1999) dependency parser. We couch this account of word order in a declarative grammatical formalism which leads to a framework called *Topological Dependency Grammar* or TDG. Earlier versions of TDG have already been presented in (Duchier & Debusmann 2001), (Duchier 2001) and (Debusmann 2001). A prototype parser for TDG, implemented using techniques described in (Duchier 2000), demonstrates that the efficiency of the parser described in (Duchier 1999) can be carried over to the more sophisticated TDG grammar framework.

1.2 Dependency grammar

In this section, we give a brief overview of the theory of dependency grammar, starting out from its history.

History. Dependency grammar dates back to the middle ages (Covington 1984). Others even argue that it actually dates back to antiquity. Modern day dependency grammar is often credited to Tesnière (1959) and Lecerf (1960).

Heads and dependents. There is little agreement amongst the various approaches about what dependency grammar really is. Still, one commonality across all dependency-based theories is the distinction between *head* and *dependent*, similar (though not identical) to the distinction between *functor* and *argument* in

Categorial Grammar. Heads and their dependents are related by directed and typed *dependency relations* which are for the most part semantically motivated.

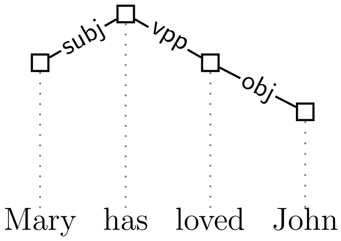
Dependency relations. The degree of the 'semantic' motivation of dependency relations varies considerably across dependency-based theories. While in some approaches (e.g. Word Grammar), dependency relations are embodied by grammatical roles such as *subject* and *object*, there are also approaches where dependency relations are identified with 'deep' argument roles such as *agent*, *patient* and *beneficiary*. Among the theories on the latter track are Prague School descendants such as FGD (Sgall et al. 1986) and Dependency Grammar Logic (DGL) (Kruijff 2001). Many theories also postulate relations that reflect considerations regarding syntactic structure, linearization, but also (e.g. Kruijff 2001) discourse and temporal structure.

Valency. *Valency* is the fundamental concept of all the theories in the paradigm of dependency grammar. It states what arguments (dependents) a word needs to be combined with in order to form a meaningful unit. Valency of words can thus be likened to the term's original use in chemistry, where it expresses the bonding requirements between chemical elements. An example is the word *loves*, which requires through its valency specification a subject and an object, or, using deep argument roles as dependency relations, an agent (the lover) and a patient (the one being loved).

Lexicalization. One of the key characteristics of most dependency-based approaches to grammar is that they are highly lexicalized. Already in traditional DG (Tesnière 1959), the valency of each word form is determined in the lexicon. This notion of lexicalized valency originates from DG, but has in the meantime been assimilated by most phrase structure-based and also CG-based approaches. Government and Binding theory (GB) (Chomsky 1986) and HPSG (Pollard & Sag 1994) for instance employ the similar concept of *subcategorization*.

Lexicalized DGs include DUG, MTT, Lexicase, WG, and, more recently, DACHS (Bröker 1999), DGL (Kruijff 2001) and of course TDG (this study). A few approaches to dependency grammar make use of rule-based valency specifications, e.g. the dependency grammar described in (Hays 1964) and (Gaifman 1965). The more recent approach of (Kahane et al. 1998) is also rule-based rather than lexicalized.

Dependency trees. A DG analysis is a set of dependency relations between pairs of words in a sentence, making up an analysis structure called *dependency tree*. Dependency trees are often drawn with vertical dotted lines called *projection edges* which connect nodes in the tree with the words that correspond to them. (1.1) shows an example dependency tree including projection edges.



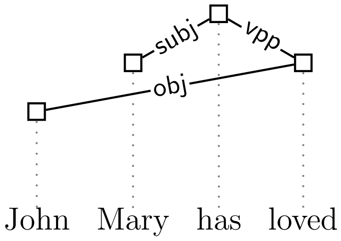
(1.1)

Here, *Mary* is the subject of the finite perfectizer *has*. *loved* is a past participle complement of *has* and *John* is the object of *loved*.

In many dependency-based theories, dependency trees are general graphs rather than trees. Word Grammar for instance makes use of cyclic analysis structures (e.g. for the analysis of relative clauses).

Projectivity. The *projectivity condition* is another aspect which can be used to distinguish various formalisms in the dependency paradigm. It was introduced by Lecerf (1960). Hays (1964) and Gaifman (1965) utilized the notion in their attempt to relate DG to context-free grammar and found out that projective DG is weakly equivalent to CFG. The projectivity condition states that the set of nodes reachable by traversing downwards zero or more edges must form a contiguous sequence. Dependency trees which obey this condition are called *projective*, and *non-projective* if they do not.

In graphical terms, a violation of the projectivity condition typically leads to crossing edges in the dependency tree. (1.2) depicts such a non-projective dependency tree, where the edge from *loved* to *John* crosses the projection edges corresponding to *Mary* and *has*:



(1.2)

Compared with phrase structure grammar, the projectivity condition is the counterpart of the assumption that constituents are contiguous strings of words. Similar to PSG, projective DG is bereft of the possibility to decouple vertical from horizontal organization principles and is therefore deprived of the possibility to elegantly analyze variable word order languages. Even for rigid word order languages such as English, it brings about the exclusion of structures which seem linguistically adequate, such as the dependency tree shown in (1.2). For this reason, many DG researchers have looked for weaker restrictions than projectivity, this study included.

Levels of analysis. Dependency-based theories of grammar differ in the number of levels of analysis structures they postulate. Most theories stick to a *monostratal* approach using only one level of analysis (WG, Lexicase, DGL), whereas the most prominent *multistratal* DGs are Meaning Text Theory, which employs seven levels or *strata* of representation, and FGD.

1.3 Topological Fields Theory

Complementary to the short introduction to dependency grammar in the previous section, we now review the basic concepts of topological fields theory.

History. Topological fields theory is a traditional approach to German descriptive syntax reaching back, as Höhle (1986) shows, at least to Herling (1821) and Erdmann (1886). Only recently has topological fields theory become popular among PSG and DG grammarians: e.g. in his HPSG-based grammar of German, Kathol (1995) heavily relies on topological fields theory. The same holds for Gerdes & Kahane (2001) in the DG paradigm.

Topological fields. The basic idea of topological fields theory is to divide a sentence into contiguous substrings. Each of these substrings is assigned to a position called *topological field*. The inventory of topological fields commonly assumed includes *Vorfeld* (‘pre-field’), *linke Satzklammer* (‘left sentence bracket’), *Mittelfeld* (‘midfield’), *rechte Satzklammer* (‘right sentence bracket’) and *Nachfeld* (‘post-field’).

The structure imposed on a German sentence by topological fields theory is as follows: the left sentence bracket (‘(’) is the position of the finite verb, and there can be at most one constituent left of it in the Vorfeld. The disposition of material into the Vorfeld is often called *fronting*. Verbal dependents of the finite verb land in the right sentence bracket (‘)’), which is often referred to as the *verb cluster*. Any other dependents of the finite verb or of embedded verbs are positioned in the Mittelfeld, where their linear arrangement is quite arbitrary, giving rise to phenomena such as *scrambling* (Ross 1967). However, the order of the material in the Mittelfeld is not entirely arbitrary but subject to preferences (e.g. Uszkoreit 1987) stating e.g. that nominative nouns precede dative nouns and that dative nouns precede accusative nouns, the situation being different for pronouns. The Mittelfeld is surrounded or ‘bracketed’ by the left sentence bracket and the right sentence bracket. We give two example sentences analyzed in this fashion below:

Vorfeld	(	Mittelfeld	)	Nachfeld
Maria *Maria*	*hat* *has*	*dem Mann einen Korb* *the man a basket*	*gegeben.* *given.*	
Einen Korb gegeben *A basket given*	*hat* *has*	*Maria dem Mann* *Maria the man*		

The Nachfeld is the position for relative clauses which have been *extraposed*, i.e. dislocated to the right of the right sentence bracket, and also for subordinate clauses. In the examples below, the subordinate clause *dass er sie geliebt hat* and the extraposed relative clause *der sie geliebt hat* occur in the Nachfeld:

Vorfeld	(	Mittelfeld	)	Nachfeld
Maria *Maria*	*hat* *has*	*dem Mann nicht* *the man not*	*geglaubt* *believed*	*dass er sie geliebt hat.* *that he her loved has.*
Maria *Maria*	*hat* *has*	*dem Mann einen Korb* *the man a basket*	*gegeben* *given*	*der sie geliebt hat.* *who her loved has.*

Sentential patterns. Usually, a typology of three *sentential patterns* or *sentence types* is assumed for German, viz. *verb-first*, *verb-second* and *verb-final.* These names refer to the respective position of the finite verb. The examples given above were all verb-second sentences with the finite verb *hat* appearing in the left sentence bracket. Verb-second is by far the most common sentential pattern in German, being used for declarative sentences and constituent questions (also known as wh-questions).

In the case of polar questions (yes-no questions) and imperatives, the verb is not in verb-second but in initial position with the Vorfeld remaining empty. This sentential pattern is called verb-first:

Vorfeld	(	Mittelfeld	)	Nachfeld
	Hat	*Maria dem Mann einen Korb*	*gegeben?*	
	Has	*Maria the man a basket*	*given?*	
	Gib	*dem Mann einen Korb!*		
	Give	*the man a basket!*		

If the finite verb appears sentence-final, we observe the verb-final sentential pattern. Verb-final sentences are instantiated by embedded sentences such as subordinate and relative clauses. It is commonly assumed in adaptations of topological fields theory (e.g. Kathol 1995) that the Vorfeld is empty in verb-final sentences, and that the complementizer (in subordinate clauses) or the relative pronoun (in relative clauses) is in the position of the left sentence bracket:

Vorfeld	(	Mittelfeld	)	Nachfeld
	dass	*Maria dem Mann einen Korb*	*gegeben hat*	
	that	*Maria the man a basket*	*given has*	
	dem	*Maria einen Korb*	*gegeben hat*	
	whom	*Maria a basket*	*given has*	

Notice that matrix clauses can only be verb-first or verb-second sentences. The verb-final sentential pattern is restricted to subordinate clauses and relative clauses.

Up to this point, we have only discussed the relevance of topological fields theory for German. However, the theory also applies for other languages. Kathol for instance extends his account of linearization-based syntax using topological fields theory to apply also to Dutch, Scandinavian languages and Yiddish (Kathol 1995), (Kathol 2000). Penn (1999) assumes the existence of topological fields for his analysis of Serbo-Croatian. He identifies topological fields with domain objects denoting a particular region over which to state linear precedence constraints. Kruijff (2001) goes several steps further: he makes use of generalized notions from topological fields theory in order to tackle a number of VX, XV and SVO languages[2], using a language typology based on work by Greenberg (1966) and Hawkins (1983).

1.4 Topological Dependency Grammar

The grammar formalism introduced in this thesis, TDG, makes use of two orthogonal yet mutually constraining tree structures:

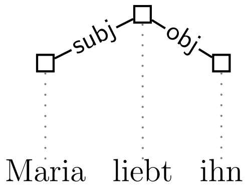
[2] "VX" stands for verb-first, "XV" for verb-final and "SVO" for subject-verb-object order.

- the syntactic dependency tree or ID tree
- the topological dependency tree or LP tree

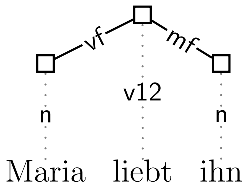
The non-ordered *syntactic dependency tree* (ID tree) conveys syntactic dependency or immediate dominance information, whereas the ordered and projective *topological dependency tree* (LP tree) conveys information regarding linear precedence. Edges in ID trees are labeled by grammatical roles such as subject and object, while edges in LP trees are labeled by topological fields such as Vorfeld and Mittelfeld. An *ID analysis* consists of an ID tree and a *lexical assignment* of lexical entries to nodes. An *LP analysis* consists of an LP tree, a lexical assignment, a total order on the set of nodes and a node label assignment.

The two trees can be seen as a realization of Curry's (1961) call for a distinction between *tectogrammatical structure* and *phenogrammatical structure*. The tectogrammatical structure expressed by the ID tree can be utilized to compositionally construct semantic meaning, and the phenogrammatical structure (LP tree) is concerned with the overt linear realization of words in a string.

A TDG ID/LP analysis consists of both an ID analysis and an LP analysis sharing the same lexical assignment. At the heart of the well-formedness conditions characterizing the admissibility of both analyses lies a lexicalized notion of valency: each word states how many possible dependents of a particular kind it can have. For instance the word *liebt* (*loves*) requires a subject and an object on the ID analysis level (1.3). On the LP analysis level (1.4), it requires its arguments to land either in its Vorfeld or the Mittelfeld. Below, we depict an ID/LP analysis of the sentence *Maria liebt ihn* (*Maria loves him*):



(1.3)



(1.4)

The ID tree in (1.3) is to be interpreted as follows: The root node is the finite verb *liebt*. *liebt* has two dependents, viz. the subject (subj) *Maria* and the object (obj) *ihn*. In the LP tree in (1.4), the root is again *liebt*. *liebt* has two dependents: *Maria* is in the Vorfeld (vf) of *liebt* and *ihn* in its Mittelfeld (mf).[3]

Although we concentrate on the application of TDG for German in this study, we claim that the grammar formalism is applicable to other languages as well. We envisage that other Germanic languages such as Dutch will be rather easy to handle, as well as 'simple' (with respect to word order) languages such as English. Actually, an implementation of TDG has been used in combination with a small English grammar in a software project at the Department of Computational Linguistics (Gabsdil, Koller & Striegnitz 2001). It should also be possible to handle non-Germanic languages with TDG, e.g. following Kruijff's (2001) language typology.

1.5 Overview

The structure of this thesis is as follows. Chapter 2 introduces the notion of an ID analysis. An ID analysis consists of an ID tree and a lexical assignment, where ID trees are traditional non-ordered dependency trees. TDG's concept of an ID analysis is based on the dependency grammar suggested in (Duchier 1999) but does not include any constraints on word order.

In chapter 3, we turn to the question of how to derive word order from non-ordered ID analyses. We give an overview of previously suggested theories tackling that topic, both in the paradigm of PSG and of DG. We review the groundbreaking work of Reape (1994) on word order domains in HPSG and point out its defects. Then, we show how Kathol (1995) goes beyond Reape's theory and how Bröker (1999) transfers the notion of word order domains to DG. We conclude the chapter with a theory recently suggested by Gerdes & Kahane (2001), which is in many respects surprisingly similar to TDG.

In chapter 4, we introduce the notion of an LP analysis, equipping TDG with a theory of word order. A TDG LP analysis consists of an LP tree, a lexical assignment, a total order on the set of nodes and a node label assignment. LP analyses describe topological structures in the spirit of topological fields theory.

Chapter 5 brings together the ID and the LP analyses using the notion of an ID/LP analysis. An ID/LP analysis is a pair of an ID and an LP analysis. Both share the same lexical assignment and the LP analysis includes a total order on the set of nodes and a node label assignment. ID and LP analyses are related to each other through the 'climbing' mechanism. We characterize licensed climbings by

[3] The LP tree also includes *node labels* (n and v12) on the projection edges. We postpone the introduction of node labels to chapter 4.

well-formedness conditions called ID/LP principles.

The TDG ID/LP lexicon is the theme of chapter 6. Since TDG is highly lexicalized, it is convenient to express linguistic generalizations in the lexicon. Therefore, we arrange lexical types in a lexical type hierarchy, and obtain subtypes by lexical inheritance.

In chapter 7, we present a TDG grammar fragment for German, thereby demonstrating the expressive power of the framework. The fragment covers all sentential patterns, i.e. verb-first, verb-second and verb-final. It handles subordinate clauses, relative sentences, prepositional phrases, adverbial modification and more.

We lay out in chapter 8 how the grammar fragment can elegantly account for several notorious phenomena in German syntax. These phenomena include scrambling, VP-dislocation (including extraposition, intraposition and fronting) and auxiliary flip. In addition, it also covers relative clause extraposition and pied piping. Chapter 8 is partly based on (Duchier & Debusmann 2001).

Chapter 9 provides a precise formalization of the notions of a TDG grammar and the ID, LP, and ID/LP principles.

In chapter 10, we outline our prototype TDG parser implementation. The parser is a reduction of the formalization presented in chapter 9 into a constraint program in the Mozart-Oz programming language (*Mozart* 1998) using techniques described in (Duchier 2000).

We conclude the thesis in chapter 11. Here, we also point out several ideas for future research.

Appendix A outlines how we handle a number of phenomena which for simplicity we do not cover in the grammar fragment of chapter 7, but which are still covered in the implementation.

Chapter 2

ID analyses

This chapter introduces the notion of an immediate dominance (ID) analysis. An ID analysis consists of a non-ordered traditional dependency tree called ID tree and a lexical assignment. The well-formedness conditions for ID analyses are based on the axiomatization of the dependency grammar suggested in (Duchier 1999) and comprise general and lexicalized principles. We do not yet formalize the well-formedness conditions in great detail: we refer to chapter 9 for a more in-depth account.

2.1 ID trees

Similar to Hudson's Word Grammar (Hudson 1990), ID trees (also called *syntactic dependency trees*) relate heads and dependents by grammatical roles in the tradition of Systemic-Functional Linguistics (SFL) (Halliday 1961), (Halliday 1994). Grammatical roles include e.g. subject, object, past participle and adverb. ID tree edges are labeled by abbreviations of these grammatical roles, e.g. subj for subject, obj for object, vpp for past participle and adv for adverb. Nodes in the ID tree correspond 1:1 to words in the analyzed string.[1] Hence for a sentence of length n, there are precisely n nodes in the corresponding ID tree.[2] In (2.2) below, we give

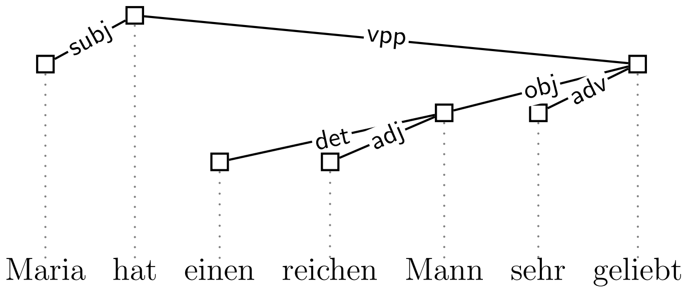
[1] Because of their 1:1-correspondence, we will often identify nodes by words.

[2] We represent *all* the words of the analyzed string in the ID tree, contrary to approaches like FGD (Sgall et al. 1986). FGD focuses only on 'autosemantic' words in its dependency tree representation, leaving out for instance function words.

an example ID tree of the sentence glossed in (2.1).

Maria	hat	einen	reichen	Mann	sehr	geliebt.
Maria	has	a	rich	man	a lot	loved.

"Maria loved a rich man very much." (2.1)



(2.2)

Here, *Maria* is the subject of the perfectizer *hat*, and *geliebt* its past participle complement. *Mann* is the object of *geliebt*, *einen* the determiner and *reichen* an adjective of *Mann*. Finally, *sehr* is an adverb of *geliebt*.

We represent nodes by boxes, connected to each other by downwardly-directed labeled edges. In the picture, each node is connected to its corresponding word by a dotted vertical projection edge. We call a mother in an ID tree *syntactic head* and a daughter *syntactic dependent*. Notice that because ID trees are non-ordered trees, we can pick an arbitrary linear arrangement for displaying purposes.

2.2 ID principles

HPSG (Pollard & Sag 1994) is a prototypical constraint-based grammar formalism. HPSG states the semantics of its constraints or *principles* with respect to typed feature structures. A *sign* is a typed feature structure describing a linguistic analysis, and it is licensed if it satisfies all constraints. In TDG, we also employ a set of principles to characterize the well-formedness conditions for ID analyses. But whereas HPSG states its principles with respect to typed feature structures, we state the principles for licensed ID analyses with respect to finite labeled graphs under a lexical assignment.

What is an ID analysis? An ID analysis $(V, E_{\text{ID}}, \varepsilon)$ consists of a finite set of nodes V, a finite set of edges $E_{\text{ID}} \subseteq V \times V \times \mathcal{R}$ (where $\mathcal{R}$ is a finite set of edge labels[3]) and a lexical assignment ε. We call $(V, E_{\text{ID}}, \varepsilon)$ an *attributed graph*. From the set of attributed graphs, we are only interested in those attributed graphs which are

[3] In the context of ID trees, the set of edge labels is the set of grammatical roles.

valid ID analyses. A valid ID analysis must satisfy the set of ID principles which consist of *general principles* and *lexicalized principles*. The only general principle is the *treeness principle* which requires that every ID analysis be a tree. The set of lexicalized principles contains the *accepted edge labels principle*, the *valency principle* and *edge constraints*. The accepted edge labels principle restricts the label of a node's incoming edge, and the valency principle poses restrictions on a node's outgoing edges. Edge constraints are formulated with respect to lexical attributes and can be used to capture notions such as agreement[4].

We write w for a node in the set of nodes V, and ρ for an edge label in the set of edge labels $\mathcal{R}$. An edge (w, w', ρ) from w to w' labeled with ρ is written $w{-}\rho{\rightarrow}w'$, and we call w' a ρ-dependent of w. We also say that w' *has role* ρ.

2.2.1 General principles

Treeness principle

Principle 2.1 *Each analysis is a tree.*

The treeness principle characterizes trees by the following three *treeness conditions*:

1. Each node has at most one incoming edge.
2. There is precisely one node (the root) with no incoming edge.
3. There are no cycles.

An ID analysis $(V, E_{\text{ID}}, \varepsilon)$ is well-formed under the treeness principle only if all of these conditions hold.

2.2.2 Lexicalized principles

Lexicalized principles are defined with respect to a lexicon $(\mathcal{E}, \mathcal{A})$ consisting of a finite set $\mathcal{E}$ of lexical entries and a finite set $\mathcal{A}$ of lexical attributes. A lexical entry is a pair (s, e_t) of a string s from the set *Strs* of strings[5] and a lexical type e_t from the set of lexical types $\mathcal{E}_t$:

$$\mathcal{E} \quad \subseteq \quad \mathit{Strs} \times \mathcal{E}_t \tag{2.3}$$

[4] We demonstrate how to capture agreement using edge constraints in appendix A.

[5] In this study, we consider the set of *Strs* strings to denote fully inflected word forms.

A lexical attribute $\alpha \in \mathcal{A} : \mathcal{E}_t \to D_\alpha$ maps lexical types to values in some domain D_α. We introduce the function $\varepsilon : V \to \mathcal{E}$ called *lexical assignment* which assigns to each node a lexical entry. Overloading α, we introduce the function $\alpha : V \to D_\alpha$ defined as:

$$\alpha(w) \quad = \quad \alpha(\pi_2(\varepsilon(w))) \tag{2.4}$$

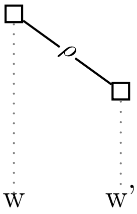
where $\pi_2(\varepsilon(w))$ stands for the second projection of $\varepsilon(w)$, i.e. $\pi_2((s, e_t)) = e_t$, and $\alpha(w)$ denotes the value of lexical attribute α at node w.

We posit the set $\mathcal{A}_{\text{ID}} = \{\mathsf{labels}_{\text{ID}}, \mathsf{valency}_{\text{ID}}\}$ of lexical attributes called *ID attributes.* $\mathsf{labels}_{\text{ID}}$ is used in the formulation of the accepted labels principle, and $\mathsf{valency}_{\text{ID}}$ in the formulation of the valency principle.

Accepted edge labels principle

Principle 2.2 *Each node must accept the label of its incoming edge if any.*

Except for the root, each node in the ID tree has one *incoming edge.* We show an example in (2.5) below, where the incoming edge of node w' is labeled with grammatical role ρ:



(2.5)

The accepted edge labels principle restricts the licensed labels for a node's incoming edge: using the lexical attribute $\mathsf{labels}_{\text{ID}} : \mathcal{E}_t \to 2^{\mathcal{R}}$ we assign to each lexical type a set of licensed edge labels. We call the value of the $\mathsf{labels}_{\text{ID}}$-attribute *accepted roles.* An edge from node w to node w' labeled with ρ is only licensed if ρ is in the set of accepted edge labels of w':

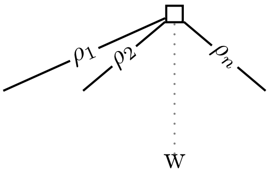
$$w{-}\rho{\to}w' \in E_{\text{ID}} \quad \Rightarrow \quad \rho \in \mathsf{labels}_{\text{ID}}(w') \tag{2.6}$$

An ID analysis $(V, E_{\text{ID}}, \varepsilon)$ is well-formed under the accepted edge labels principle only if (2.6) holds for all edges $w{-}\rho{\to}w' \in E_{\text{ID}}$.

Valency principle

Principle 2.3 *Each node's outgoing edges must precisely fulfill the node's valency.*

Each node w has a number of outgoing edges labeled with roles $\rho_1, \rho_2, \ldots, \rho_n$:[6]



(2.7)

Symmetrically to the accepted edge labels principle, the valency principle poses restrictions on a node's *outgoing edges*. We define the valency principle with respect to a language of *role valency specifications* $v(\mathcal{R})$. A role valency specification $\rho' \in v(\mathcal{R})$ is defined by the following abstract syntax:

$$\rho' \quad ::= \quad \rho \mid \rho? \mid \rho* \quad \text{for } \rho \in \mathcal{R} \tag{2.8}$$

We pose the function $\mathsf{valency}_{\text{ID}} : \mathcal{E}_t \to 2^{v(\mathcal{R})}$ mapping lexical types to sets of role valency specifications. In the following, we will often refer to the value of the $\mathsf{valency}_{\text{ID}}$-attribute simply as *role valency*. Writing $\rho(w)$ for the set of dependents of node w with role ρ and $|\rho(w)|$ for its cardinality, we formalize the valency principle as follows:

$$\begin{array}{rcl} \rho \in \mathsf{valency}_{\text{ID}}(w) & \Rightarrow & |\rho(w)| = 1 \\ \rho? \in \mathsf{valency}_{\text{ID}}(w) & \Rightarrow & |\rho(w)| \leq 1 \\ \rho* \in \mathsf{valency}_{\text{ID}}(w) & \Rightarrow & |\rho(w)| \geq 0 \\ \text{otherwise} & \Rightarrow & |\rho(w)| = 0 \end{array} \tag{2.9}$$

Intuitively, ρ indicates a required dependent with role ρ, $\rho?$ an optional dependent and $\rho*$ an arbitrary number of dependents of role ρ.

An ID analysis $(V, E_{\text{ID}}, \varepsilon)$ is well-formed under the valency principle only if (2.9) holds for all nodes $w \in V$.

Role valency can be likened to *subcategorization*: for instance a finite intransitive verb subcategorizes for a required subj-dependent, and a finite transitive verb for both a required subj- and a required obj-dependent. We also capture *modification* with the role valency-concept: e.g. to license an arbitrary number of adjectives which may modify a noun we state that a noun may have an arbitrary number of dependents with role adj, i.e. the noun's role valency includes the role valency specification adj$*$.

Edge constraints

Principle 2.4 *Each edge must be licensed by the corresponding edge constraint.*

[6] Notice that the roles $\rho_1, \rho_2, \ldots, \rho_n$ need not necessarily be pairwise distinct.

Edge constraints are a family (Γ_ρ) of binary predicates indexed by grammatical roles ρ. We call edge constraints for ID analyses *ID edge constraints*. Edge constraints are stated by making reference to lexical attributes.[7] An edge $w{-}\rho{\rightarrow}w'$ is only licensed if the corresponding edge principle $\Gamma_\rho(w, w')$ is satisfied:

$$w{-}\rho{\rightarrow}w' \in E_{\text{ID}} \quad \Rightarrow \quad \Gamma_\rho(w, w') \tag{2.10}$$

An ID analysis $(V, E_{\text{ID}}, \varepsilon)$ is well-formed under all edge constraints only if (2.10) holds for all $w, w' \in V$ and $\rho \in \mathcal{R}$.

2.3 Examples

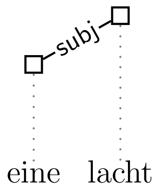
We present in Figure 2.1 an example lexicon including the indefinite determiner *eine* (*a*), the adjectives *hübsche* and *kleine* (*pretty* and *little*), the common noun *Frau* (*woman*) and the intransitive finite verb *lacht* (*laughs*). We define the lexical assignment ε as a straightforward mapping of nodes to their corresponding lexical entries: for instance $\varepsilon(eine)$ maps the node corresponding to *eine* to the appropriate lexical entry topmost in Figure 2.1:

$$\varepsilon(eine) \quad = \quad (eine \quad , \quad \left[\begin{array}{rcl} \mathsf{labels}_{\text{ID}} & : & \{\mathsf{det}\} \\ \mathsf{valency}_{\text{ID}} & : & \emptyset \end{array} \right]) \tag{2.11}$$

In the following, we use the lexicon to demonstrate how the accepted edge labels principle and the valency principle lead to the exclusion of ungrammatical sentences and the inclusion of grammatical sentences.

2.3.1 Accepted edge labels principle violation

The ID tree below violates the accepted edge labels principle:



(2.12)

[7] The abstract syntax of the contraint language in which edge constraints can be stated is presented in chapter 9.

$$(\textit{eine} \quad , \left[\begin{array}{lcl} \mathsf{labels}_{\mathrm{ID}} & : & \{\mathsf{det}\} \\ \mathsf{valency}_{\mathrm{ID}} & : & \emptyset \end{array}\right])$$

$$(\textit{hübsche} \quad \left[\begin{array}{lcl} \mathsf{labels}_{\mathrm{ID}} & : & \{\mathsf{adj}\} \\ \mathsf{valency}_{\mathrm{ID}} & : & \emptyset \end{array}\right])$$

$$(\textit{kleine} \quad , \left[\begin{array}{lcl} \mathsf{labels}_{\mathrm{ID}} & : & \{\mathsf{adj}\} \\ \mathsf{valency}_{\mathrm{ID}} & : & \emptyset \end{array}\right])$$

$$(\textit{Frau} \quad , \left[\begin{array}{lcl} \mathsf{labels}_{\mathrm{ID}} & : & \{\mathsf{subj}\} \\ \mathsf{valency}_{\mathrm{ID}} & : & \{\mathsf{det}, \mathsf{adj}*\} \end{array}\right])$$

$$(\textit{lacht} \quad , \left[\begin{array}{lcl} \mathsf{labels}_{\mathrm{ID}} & : & \emptyset \\ \mathsf{valency}_{\mathrm{ID}} & : & \{\mathsf{subj}\} \end{array}\right])$$

Figure 2.1: Example lexicon

By the lexical assignment, the set of accepted edge labels of the determiner *eine* includes only det but not subj. Thus, the accepted edge labels constraint instantiated below is violated:

$$\textit{lacht}\text{—}\mathsf{subj}\rightarrow\textit{eine} \quad \Rightarrow \quad \mathsf{subj} \in \mathsf{labels}_{\mathrm{ID}}(\textit{eine}) \tag{2.13}$$

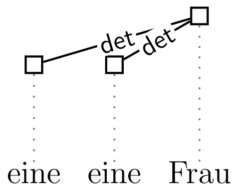
because $\mathsf{subj} \notin \mathsf{labels}_{\mathrm{ID}}(\textit{eine}) = \{\mathsf{det}\}$.

2.3.2 Valency principle violation

The ID tree shown in (2.14) violates the valency principle because *Frau* requires a determiner by the lexical assignment, but lacks one in the tree:

(2.14)

The ID tree below also violates the valency principle since *Frau* requires only one determiner but gets two:



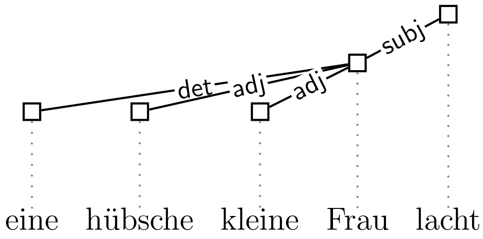
(2.15)

Both trees violate the instantiation of the valency principle shown below:

$$\mathsf{det} \in \mathsf{valency}_{\text{ID}}(\mathit{Frau}) \quad \Rightarrow \quad |\mathsf{det}(\mathit{Frau})| = 1 \tag{2.16}$$

2.3.3 Well-formed ID tree

The following ID tree is well-formed:



(2.17)

Here, *Frau* is modified by the two adjectives *hübsche* and *kleine*.

2.4 Summary

We introduced the basic concepts behind the notion of an immediate dominance (ID) analysis. An ID analysis consists of a non-ordered traditional dependency tree and a lexical assignment. The well-formedness conditions for ID analyses are defined using general principles and lexicalized principles.

ID analyses are only one part of our grammar formalism, and they are not at all concerned with surface word order. In the next chapter, we will show how previously suggested theories incorporate constraints on surface word order. After that, we propose our own account of word order in chapter 4.

Chapter 3

Approaches to linear precedence

This chapter contains an overview of several theories which stand close to our approach in the paradigms of PSG and of DG. What all of these theories have in common is the fundamental idea of dissociating syntactic structure from surface structure. All of them posit two dissociated yet interacting levels of representation, one characterizing relations of immediate dominance (ID), and the other relations of linear precedence (LP). The set of discussed theories includes (Reape 1994) and (Kathol 1995) (using HPSG), and (Bröker 1999) and (Gerdes & Kahane 2001) (using DG).

3.1 Reape

Mike Reape suggested the theory of *word order domains* formulated in an HPSG-setting (Reape 1994) which has paved the way for many modern HPSG grammars for freer word order languages such as German. For instance Müller (1999) and Kathol (1995) adapt Reape's word order domains in their HPSG grammars for German, and other adaptations of Reape's idea can be found in (Penn 1999) and (Fouvry & Meurers 2000).

Reape's basic idea is to dissociate structures dealing with immediate dominance (ID) from those dealing with linear precedence (LP). Reape's terminology is however different: he distinguishes a level of syntax (ID) and a level of word order domains[1] (LP). In our presentation of Reape's work, we use the terms ID and LP. Each analysis is represented by two tree structures instead of one: an ID tree and

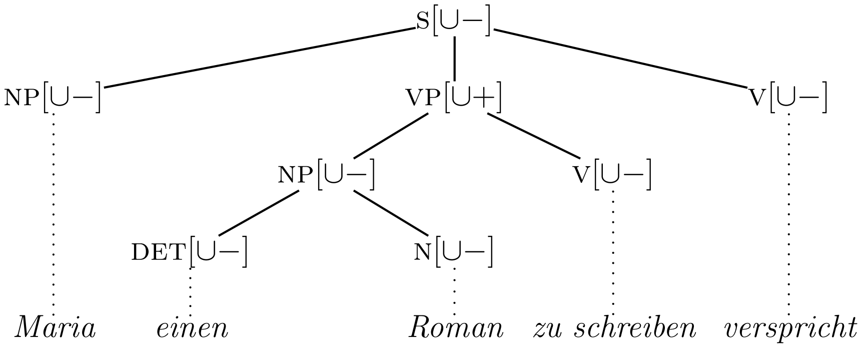
[1] Reape's word order domains are a technical notion: his word order domains do not correspond directly to fields in topological fields theory.

an LP tree. The ID tree is an non-ordered phrase structure analysis, and the LP tree is a flattened and ordered phrase structure analysis. As an example, consider the following German sentence:

(dass)	einen	Roman	Maria	zu	schreiben	verspricht	
(that)	a	novel(acc)	Maria	to	write	promises	(3.1)

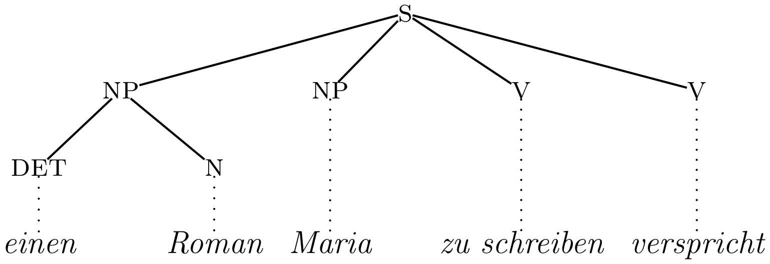
"(that) Maria promises to write a novel"

The corresponding ID tree analysis is shown here:



(3.2)

And here is the LP tree analysis of the example sentence:



(3.3)

The word order domain of a node in the LP tree is the sequence of its daughters. E.g. in LP tree (3.3), the word order domain of the S-node equals the sequence $\langle \mathrm{NP}, \mathrm{NP}, \mathrm{V}, \mathrm{V} \rangle$ of the two NPs *einen Roman* and *Maria* and the two verbs *zu schreiben* and *verspricht*. The word order domain of the node corresponding to the NP *einen Roman* is the sequence $\langle \mathrm{DET}, \mathrm{N} \rangle$ of the determiner *einen* and the noun *Roman*. Reape constrains the order of elements in a word order domain by *linear precedence rules* like $\mathrm{DET} \prec \mathrm{N}$, which requires that determiners precede nouns and $\mathrm{NP} \prec \mathrm{V}$, which requires that NPs precede verbs.

The LP tree is obtained from the ID tree by a process of flattening. This flattening is controlled by the unioned-attribute[2]: if a node is [∪+], it 'contributes' its daughters

[2] We write unioned = + as [∪+] and unioned = − as [∪−].

to the domain of its mother. In our example, the VP-node is [$\cup+$], and it therefore contributes its daughters (NP and V) to the domain of the S-node. If a node is [$\cup-$], the node contributes itself to the domain of its mother. In the example, the NP *einen Roman* is [$\cup-$] and hence contributes itself (and not its daughters) into the S-node's domain. Reape employs the notion of *language-specific principles* to determine the value of the unioned-attribute.

3.1.1 Formalization

We proceed with a HPSG-independent formalization of Reape's word order domains. An ID tree is a tree $(V_{\text{ID}}, E_{\text{ID}})$, where V_{ID} is a finite set of ID nodes and $E_{\text{ID}} \subseteq V_{\text{ID}} \times V_{\text{ID}}$ a finite set of edges. In Reape's approach, ID trees are phrase structure trees, encoded in HPSG using the DTRS-feature (representing the daughters in the syntactic derivation). Since ID trees are non-ordered, DTRS is set- rather than list-valued. In our formulation, we regard this feature as a function $\mathsf{dtrs}_{\text{ID}} : V_{\text{ID}} \to 2^{V_{\text{ID}}}$ from ID nodes to sets of ID nodes.

An LP tree is a tree $(V_{\text{LP}}, E_{\text{LP}})$. V_{LP} is a finite set of LP nodes and $E_{\text{LP}} \subseteq V_{\text{LP}} \times V_{\text{LP}}$ a finite set of LP edges. Because LP trees are ordered, the function $\mathsf{dtrs}_{\text{LP}} : V_{\text{LP}} \to V_{\text{LP}}^*$ maps LP nodes to *sequences* of LP nodes. By flattening, the set of LP nodes is a subset of the set of ID nodes: $V_{\text{LP}} \subseteq V_{\text{ID}}$.

ID and LP trees are related to each other by the function $\mathsf{dom} : V_{\text{ID}} \to V_{\text{LP}}^*$ from ID nodes to sequences of LP nodes. The value of dom at node $v \in V_{\text{ID}}$ is called the word order domain of v. The licensed values of dom depend on the two functions $\cup^*$ (for *sequence union*) and $\mathsf{contrib}$ (for *contributions*), presented in the following.

Sequence union

Sequence union is a function $\cup^*$, mapping pairs of LP node sequences to sets of LP node sequences:

$$\cup^* : V_{\text{LP}}^* \times V_{\text{LP}}^* \to 2^{V_{\text{LP}}^*} \tag{3.4}$$

Here is the definition of $\cup^*$: for two sequences A_1 and A_2, $A_1 \cup^* A_2$ denotes the set of sequences A which satisfy the following conditions:

1. A contains all elements in A_1 and A_2 (and none other).
2. The respective order of elements in A_1 and A_2 is preserved in A.

We write $\langle a_1, \dots, a_n \rangle$ for a sequence of the elements $a_1, \dots, a_n$. As an example, let $A_1 = \langle a, b \rangle$ and $A_2 = \langle c, d \rangle$. Then, $A_1 \cup^* A_2$ is equal to:

$$\begin{array}{rl} \langle a,b \rangle \cup^* \langle c,d \rangle = & \{ \ \langle a,b,c,d \rangle \\ & \quad \langle a,c,b,d \rangle \\ & \quad \langle a,c,d,b \rangle \\ & \quad \langle c,a,b,d \rangle \\ & \quad \langle c,a,d,b \rangle \\ & \quad \langle c,d,a,b \rangle \ \} \end{array} \tag{3.5}$$

Formally, sequence union can be captured as follows. S^* is the set of all sequences $\sigma = (s, \prec_s)$ consisting of a set $s \in S$ and total order $\prec_s$ on s. We define sequence union $\cup^*$ as follows for all $(s_1, \prec_{s_1}), (s_2, \prec_{s_2}) \in S^*$:

$$(s_1, \prec_{s_1}) \cup^* (s_2, \prec_{s_2}) \quad = \quad \{(s, \prec_s) \in S^* \mid s = s_1 \cup s_2 \ \wedge \ (\prec_{s_1} \cup \prec_{s_2}) \subseteq \prec_s\} \tag{3.6}$$

Contributions

The contribution $\mathsf{contrib}(v)$ of an ID node v is determined by the boolean function $\mathsf{unioned} : V_{\text{ID}} \to \{+, -\}$. For $\mathsf{unioned}$, we introduce the following syntactic sugar:

$$\begin{array}{lll} v[\cup+] & \text{for} & \mathsf{unioned}(v) = + \\ v[\cup-] & \text{for} & \mathsf{unioned}(v) = - \\ v[\cup\pm] & \text{for} & \mathsf{unioned}(v) = + \ \vee \ \mathsf{unioned}(v) = - \end{array} \tag{3.7}$$

We define the function $\mathsf{contrib} : V_{\text{ID}} \to V_{\text{LP}}^*$ as follows:

$$\mathsf{contrib}(v) = \left\{ \begin{array}{rll} \langle v \rangle & \text{if} & v[\cup-] \\ \mathsf{dom}(v) & \text{if} & v[\cup+] \end{array} \right. \tag{3.8}$$

That is, v's contribution is either a sequence containing only itself (if $v[\cup-]$) or its own word order domain (if $v[\cup+]$).

We are now in a position to define which are the licensed values for the function dom: the word order domain $\mathsf{dom}(v)$ at node v is in the set of word order domains licensed by the sequence union $\cup^*$ of the contributions of v's daughters:

$$\mathsf{dom}(v) \in \cup^* \{\mathsf{contrib}(v') \mid v' \in \mathsf{dtrs}_{\text{ID}}(v)\} \tag{3.9}$$

If $\mathsf{contrib}(v') = \langle v' \rangle$, we say that v' has been *inserted* into the word order domain of its mother v. If $\mathsf{contrib}(v') = \mathsf{dom}(v')$, we say that it has been *merged.*

ID and LP tree share all nodes which are inserted (3.10), but not those which are merged (3.11):

$$\mathsf{contrib}(v) = \langle v \rangle \quad \Rightarrow \quad v \in V_{\text{LP}} \tag{3.10}$$

$$\mathsf{contrib}(v) = \mathsf{dom}(v) \quad \Rightarrow \quad v \notin V_{\text{LP}} \tag{3.11}$$

$\mathsf{dtrs}_{\text{LP}}(v)$ is equal to $\mathsf{dom}(v)$ for all $v \in V_{\text{LP}}$:

$$\forall v \in V_{\text{LP}} : \mathsf{dtrs}_{\text{LP}}(v) = \mathsf{dom}(v) \tag{3.12}$$

Language-specific principles

Reape controls the value of $\mathsf{unioned}$ by so-called *language-specific principles*. For German, Reape suggests the two principles shown below, which must hold for all $v \in V_{\text{ID}}$:

$$v[\cup{+}] \quad \Rightarrow \quad \mathsf{cat}(v) \in \{\text{VP}, \text{S}\} \tag{3.13}$$

$$\mathsf{extra}(v) = + \quad \Rightarrow \quad \mathsf{cat}(v) = \text{VP} \ \wedge \ v[\cup{-}] \tag{3.14}$$

Principle (3.13) expresses that only verbal projections (i.e. nodes with category VP or S) can be $[\cup+]$ and Principle (3.14) that extraposed constituents are always $[\cup-]$, where $\mathsf{extra} : V \to \{+, -\}$ is a boolean function denoting $+$ for extraposed nodes and $-$ for non-extraposed nodes.

Linear precedence rules

Reape employs linear precedence or LP rules to restrict the number of licensed word order domains. LP rules must hold for all $v \in V_{\text{ID}}$ and for all v_1, v_2 in $\mathsf{dom}(v)$, i.e. they apply locally within a word order domain. Reape (1994) assumes the following rules for the fragment of German that we consider here:

$$\mathsf{cat}(v_1) = \text{DET} \ \wedge \ \mathsf{cat}(v_2) = \text{N} \quad \Rightarrow \quad v_1 \prec v_2 \tag{3.15}$$

$$\mathsf{cat}(v_1) = \text{NP} \ \wedge \ \mathsf{cat}(v_2) = \text{V} \quad \Rightarrow \quad v_1 \prec v_2 \tag{3.16}$$

$$\mathsf{cat}(v_1) = \text{V} \ \wedge \ \mathsf{cat}(v_2) = \text{V} \ \wedge \ v_1 \in \mathsf{dtrs}_{\text{ID}}(v_2) \quad \Rightarrow \quad v_1 \prec v_2 \tag{3.17}$$

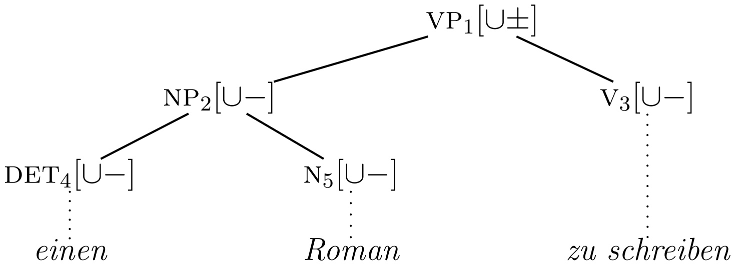
$$\mathsf{extra}(v_1) = - \ \wedge \ \mathsf{extra}(v_2) = + \quad \Rightarrow \quad v_1 \prec v_2 \tag{3.18}$$

(3.15) states that determiners precede nouns. (3.16) stipulates that NPs precede Vs (considering only verb-final clauses) and (3.17) that verbs follow their verbal complements. (3.18) states that extraposed nodes follow non-extraposed ones. For the LP rules shown, we assume the function $\mathsf{cat} : V_{\text{ID}} \to \mathit{Cats}$ mapping ID nodes to syntactic categories.

3.1.2 Examples

Noun phrases

We proceed with two easy examples of how to obtain LP trees (i.e. word order domains) from ID trees. First, consider the ID tree in (3.19) below. Nodes are represented by their phrasal category, subscripted by a unique number. The values of the nodes' unioned-attributes are annotated to their right, and we use dotted vertical edges to connect leaf nodes with their corresponding strings.



(3.19)

In order to arrive at the LP trees corresponding to (3.19), we need to know which word order domains are licensed by Reape's theory at root node $\textsc{vp}_1$. Proceeding from bottom to top, we start with calculating the dom-values at $\textsc{np}_2$:

$$\begin{aligned}\mathsf{dom}(\textsc{np}_2) \;\in\; & \mathsf{contrib}(\textsc{det}_4) \cup^* \mathsf{contrib}(\textsc{n}_5)\\ & = \langle \textsc{det}_4 \rangle \cup^* \langle \textsc{n}_5 \rangle\\ & = \{\ \langle \textsc{det}_4, \textsc{n}_5 \rangle, \langle \textsc{n}_5, \textsc{det}_4 \rangle\ \}\end{aligned} \tag{3.20}$$

Both $\textsc{det}_4$ and $\textsc{n}_5$ have been inserted into the word order domain of $\textsc{np}_2$ because they are both [∪−]. As $\langle \textsc{n}_5, \textsc{det}_4 \rangle$ is excluded by LP rule (3.15), the only licensed word order domain at $\textsc{np}_2$ is $\langle \textsc{det}_4, \textsc{n}_5 \rangle$:

$$\mathsf{dom}(\textsc{np}_2) \;=\; \langle \textsc{det}_4, \textsc{n}_5 \rangle \tag{3.21}$$

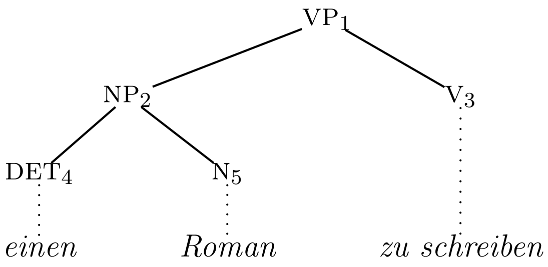
Here, we calculate the dom-values at root node $\textsc{vp}_1$:

$$\begin{aligned}\mathsf{dom}(\textsc{vp}_1) \;\in\; & \mathsf{contrib}(\textsc{np}_2) \cup^* \mathsf{contrib}(\textsc{v}_3)\\ & = \langle \textsc{np}_2 \rangle \cup^* \langle \textsc{v}_3 \rangle\\ & = \{\ \langle \textsc{np}_2, \textsc{v}_3 \rangle, \langle \textsc{v}_3, \textsc{np}_2 \rangle\ \}\end{aligned} \tag{3.22}$$

Again, both nodes $\textsc{np}_2$ and $\textsc{v}_3$ have been inserted in the domain of $\textsc{vp}_1$. The domain $\langle \textsc{v}_3, \textsc{np}_2 \rangle$ is excluded by LP rule (3.16), which leaves us with $\langle \textsc{np}_2, \textsc{v}_3 \rangle$ as the only licensed word order domain at $\textsc{vp}_1$:

$$\mathsf{dom}(\textsc{vp}_1) \;=\; \langle \textsc{np}_2, \textsc{v}_3 \rangle \tag{3.23}$$

We show the corresponding LP tree below:



(3.24)

This LP tree licenses only the linearization *einen Roman zu schreiben.*

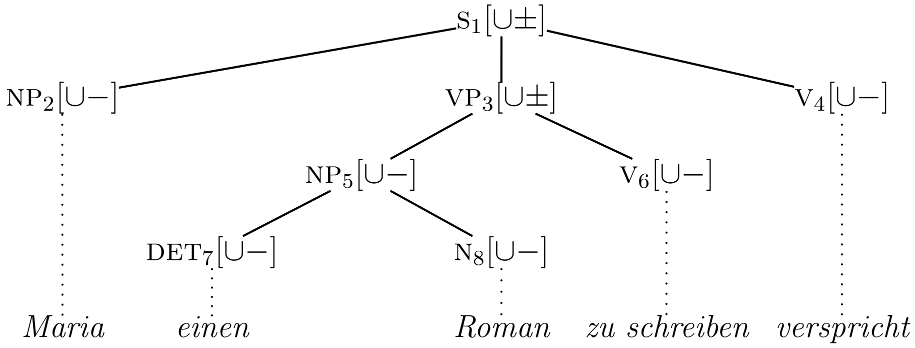
Scrambling

In this section, we show how Reape's theory tackles the scrambling example (3.1), repeated here as (3.25):

(dass)	einen	Roman	Maria	zu	schreiben	verspricht	
(that)	a	novel(acc)	Maria	to	write	promises	(3.25)

"(that) Maria promises to write a novel"

This is the ID tree analysis for (3.25):



(3.26)

LP rule (3.15) excludes sequence $\langle \textsc{n}_8, \textsc{det}_7 \rangle$. Hence, the only licensed value for $\mathsf{dom}(\textsc{np}_5)$ is $\langle \textsc{det}_7, \textsc{n}_8 \rangle$:

$$\mathsf{dom}(\textsc{np}_5) \quad = \quad \langle \textsc{det}_7, \textsc{n}_8 \rangle \tag{3.27}$$

There is also only one possible word order domain at $\textsc{vp}_3$:

$$\mathsf{dom}(\textsc{vp}_3) \quad = \quad \langle \textsc{np}_5, \textsc{v}_6 \rangle \tag{3.28}$$

since $\langle \mathrm{V}_6, \mathrm{NP}_5 \rangle$ is excluded by LP rule (3.16).

The daughters of S$_1$ are NP$_2$, VP$_3$ and V$_4$. NP$_2$ and V$_4$ must be inserted, but VPs such as VP$_3$ can be either inserted (3.29) or merged (3.30) into their mother's domain:

$$\begin{aligned} \mathsf{dom}(\mathrm{S}_1) \in\ & \langle \mathrm{NP}_2 \rangle \cup^* \langle \mathrm{VP}_3 \rangle \cup^* \langle \mathrm{V}_4 \rangle \\ & = \{\ \langle \mathrm{NP}_2, \mathrm{VP}_3, \mathrm{V}_4 \rangle, \langle \mathrm{VP}_3, \mathrm{NP}_2, \mathrm{V}_4 \rangle, \langle \mathrm{NP}_2, \mathrm{V}_4, \mathrm{VP}_3 \rangle\ \} \end{aligned} \tag{3.29}$$

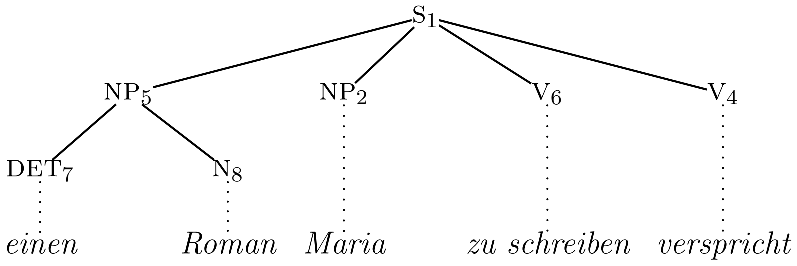
$$\begin{aligned} \mathsf{dom}(\mathrm{S}_1) \in\ & \langle \mathrm{NP}_2 \rangle \cup^* \mathsf{dom}(\mathrm{VP}_3) \cup^* \langle \mathrm{V}_4 \rangle \\ & = \langle \mathrm{NP}_2 \rangle \cup^* \langle \mathrm{NP}_5, \mathrm{V}_6 \rangle \cup^* \langle \mathrm{V}_4 \rangle \\ & = \{\ \langle \mathrm{NP}_2, \mathrm{NP}_5, \mathrm{V}_6, \mathrm{V}_4 \rangle, \langle \mathrm{NP}_5, \mathrm{NP}_2, \mathrm{V}_6, \mathrm{V}_4 \rangle\ \} \end{aligned} \tag{3.30}$$

Only (3.30) generates a linearization where *Maria* (NP$_2$) occurs between *einen Roman* (NP$_5$) and *zu schreiben* (V$_6$), i.e.:

$$\langle \mathrm{NP}_5, \mathrm{NP}_2, \mathrm{V}_6, \mathrm{V}_4 \rangle \tag{3.31}$$

That is, the respective order of the NPs *Maria* and *einen Roman* is unrestricted in (3.30), and all other permutations are excluded by LP rules (3.16) and (3.17): (3.16) ensures that NPs precede Vs, and (3.17) that *zu schreiben* (V$_6$) precedes *verspricht* (V$_4$), since $\mathrm{V}_6 \in \mathsf{dtrs}(\mathrm{V}_4)$.

We show the LP tree corresponding to word order domain (3.31) below:



(3.32)

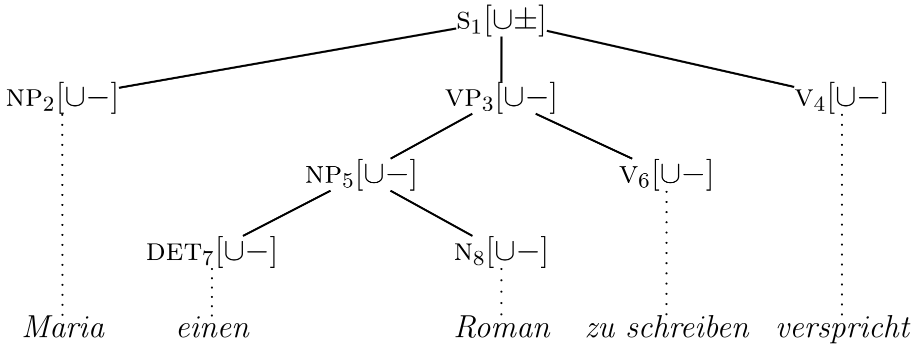
VP extraposition

Reape's approach can also treat the VP-extraposition example below:

(dass)	Maria	verspricht,	einen	Roman	zu	schreiben
(that)	Maria	promises,	a	novel(acc)	to	write

"(that) Maria promises to write a novel" (3.33)

For (3.33), we assume the ID tree analysis shown in (3.34) below. Notice that since VP$_3$ is extraposed in the linearization, it must be $[\cup-]$ by language-specific principle (3.14):



(3.34)

The word order domains at $\textsc{np}_5$ and $\textsc{vp}_3$ are the same as in the scrambling example:

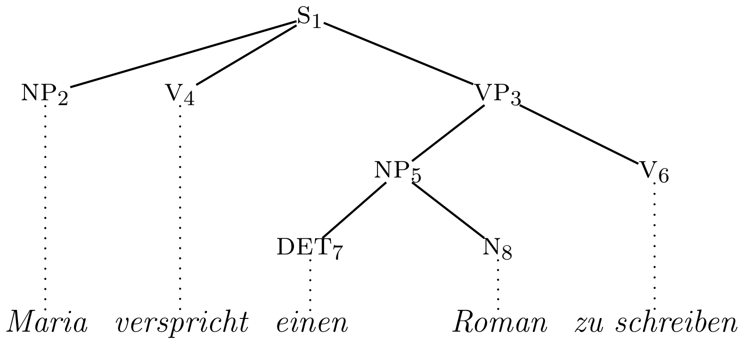
$$\begin{aligned}\mathsf{dom}(\textsc{np}_5) &= \langle \textsc{det}_7, \textsc{n}_8 \rangle & (3.35)\\ \mathsf{dom}(\textsc{vp}_3) &= \langle \textsc{np}_5, \textsc{v}_6 \rangle & (3.36)\end{aligned}$$

And $\textsc{vp}_3$ must be inserted into the domain of its mother $\textsc{s}_1$ because it is $[\cup-]$:

$$\begin{aligned}\mathsf{dom}(\textsc{s}_1) \in\ & \langle \textsc{np}_2 \rangle \cup^* \langle \textsc{vp}_3 \rangle \cup^* \langle \textsc{v}_4 \rangle \\ & = \{\ \langle \textsc{np}_2, \textsc{vp}_3, \textsc{v}_4 \rangle, \langle \textsc{vp}_3, \textsc{np}_2, \textsc{v}_4 \rangle, \langle \textsc{np}_2, \textsc{v}_4, \textsc{vp}_3 \rangle\ \}\end{aligned} \qquad (3.37)$$

The other permutations $\langle \textsc{v}_4, \textsc{np}_2, \textsc{vp}_3 \rangle$, $\langle \textsc{v}_4, \textsc{vp}_3, \textsc{np}_2 \rangle$ and $\langle \textsc{vp}_3, \textsc{v}_4, \textsc{np}_2 \rangle$ are excluded by LP rule (3.16).

Sentence (3.33) corresponds to linearization $\langle \textsc{np}_2, \textsc{v}_4, \textsc{vp}_3 \rangle$, and its LP tree is shown in (3.38):



(3.38)

Partial VP extraposition

One weakness of Reape's theory is that it cannot treat partial VP extraposition, an example of which is shown below:

(dass) Maria einen Roman verspricht, zu schreiben
(that) Maria a novel(acc) promises, to write (3.39)
"(that) Maria promises to write a novel"

The ID tree for this sentence is the same as for the full VP extraposition example (3.34). Again, by language-specific principle (3.14), VP$_3$ must be $[\cup-]$ because it is extraposed. As we saw above, this ID tree licenses the following three word order domains:

$$\begin{aligned} \mathsf{dom}(\mathrm{S}_1) \;\in\; & \langle \mathrm{NP}_2 \rangle \cup^* \langle \mathrm{VP}_3 \rangle \cup^* \langle \mathrm{V}_4 \rangle \\ & = \{\; \langle \mathrm{NP}_2, \mathrm{VP}_3, \mathrm{V}_4 \rangle, \langle \mathrm{VP}_3, \mathrm{NP}_2, \mathrm{V}_4 \rangle, \langle \mathrm{NP}_2, \mathrm{V}_4, \mathrm{VP}_3 \rangle \;\} \end{aligned} \tag{3.40}$$

but none of the sequences corresponds to linearization (3.39). That is, while Reape's theory *can* handle the extraposition of full VPs, it cannot account for the fact that particular dependents of extraposed constituents (e.g. objects) can be moved into the Mittelfeld. This defect is due to the fact that Reape's distinction between $[\cup+]$ and $[\cup-]$ is not fine-grained enough. We improve on Reape's theory in our approach, where we can specify on a per-dependent basis which dependent of an extraposed verb has to stay with its verb or may be dislocated into the Mittelfeld.

3.1.3 Criticism

Reape's word order domains were groundbreaking and paved the way for many modern HPSG grammars concerned with freer word order languages. But unfortunately, Reape's original formulation has defects.

For one, Reape's dissociation of vertical syntactic structure and horizontal linear structure is not perfect. Linear precedence constraints are still defined on categorial grounds, since Reape does not have a primitive notion of topological fields. His word order domains are only a technical notion and they cannot be straightforwardly mapped to topological fields.

The rather crude notion of 'unioning up' as expressed by the $[\cup+]/[\cup-]$-distinction does not allow Reape's theory to handle e.g. partial VP extraposition. In order to tackle extraposition phenomena in general, one needs to be able to distinguish between different kinds of arguments and allow only a subset of them to be extracted.

For instance, only PPs and relative clauses can be extracted out of NPs in German, but not determiners or adjectives.

Reape's theory also exhibits other defects with respect to extraposition: in language specific principle (3.14), he only allows VPs to be extraposed, whereas clearly also PPs, Ss and heavy NPs should be amenable to extraposition. This is one of the problems overcome by the adaptation of word order domains in (Kathol 1995), which we discuss in the next section.

3.2 Kathol

In (Kathol 1995) and also (Kathol 2000), Andreas Kathol presents a HPSG-based theory of syntax called *Linearization-Based Syntax* which is based on Reape's notion of word order domains but goes far beyond Reape's approach.

Kathol (1995) notes that Reape's approach is flawed in at least two areas. The first is extraposition: Reape (1994) allows only VPs to be extraposed, but other categories like PPs and Ss can clearly also be extraposed. Secondly, Kathol shows that Reape's analysis of raising verbs is wrong.

3.2.1 Kathol's adaptation of word order domains

Reape employs binary-valued features for both the statement of linear precedence rules, e.g. extra in (3.14), and also in stating language-specific principles (unioned). Kathol, on the contrary, dispenses with these binary-valued features altogether. Instead, he directly associates domain objects with topological properties, i.e. each domain object is assigned a topological field using the attribute topo. Kathol (1995) introduces the following set of topological fields:

field name	explanation
vf	Vorfeld
cf	complementizer field
mf	Mittelfeld
vc	verb cluster
nf	Nachfeld

(3.41)

Linear precedence rules are then replaced in (Kathol 1995) by the following *Topological Linear Precedence Statement* (TLPS) which totally orders the set of fields:

$$\mathsf{vf} \prec \mathsf{cf} \prec \mathsf{mf} \prec \mathsf{vc} \prec \mathsf{nf} \tag{3.42}$$

In addition, Kathol utilizes *Topological Cardinality Conditions* to restrict the cardinality of the Vorfeld `vf` and complementizer field `cf` to at most one.

3.2.2 Criticism

The new formulation of HPSG with word order domains as suggested by Kathol (1995) overcomes some of the defects of Reape's original proposal. The main asset of Kathol's approach is the primitive notion of fields, as opposed to Reape's binary-valued features. This primitive notion of topological fields allows Kathol to achieve a dissociation between immediate dominance and linear precedence that is more convincing than in Reape's proposal. In Kathol's approach, linear precedence constraints are stated not on categorial grounds but order topological fields (TLPS). He also dispenses with Reape's language-specific principles in favor of increased lexicalization.

3.3 Bröker

The DACHS-framework (Dependency Approach to Coupling Heterogenuous knowledge Sources) is a variant of dependency grammar proposed by Norbert Bröker (Bröker 1999). DACHS aims at decoupling the dependency tree from surface word order, to the effect that the theory does not need to compromise the semantic motivation of dependencies and allows for a declarative and formally precise characterization of word order.

DACHS makes use of two tree structures: the non-projective dependency tree structure and the word order domain (or just 'order domain') structure. Both trees are linked but structurally dissimilar. Surface word order is derived by traversing the tree induced by the word order domain structure.

DACHS is formalized using a description logic based on modal logic that owes much to the work of Blackburn (1994). Dependencies and the mapping from the dependency tree to order domain structures are described by modal operators. Simple properties such as word class and features used in the expression of precedence predicates are described by modal properties.

3.3.1 Word order domain structure

In Bröker's theory, a word order domain m is a set of words. The cardinality of an order domain can be restricted to at most one, at least one or exactly one element. No restriction on the cardinality means that zero or more domain elements are licensed. Each word is associated with a sequence of order domains where one domain in this sequence must contain the word itself. A word order domain structure is a partial order on the set $\mathcal{M}$ of word order domains, based on set inclusion.

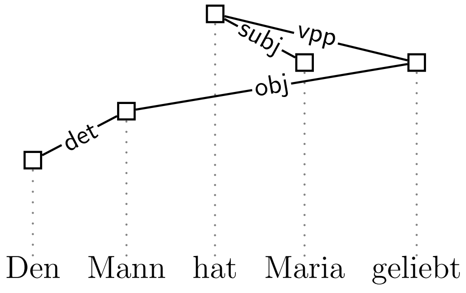
Order domain elements are assigned *features* which are later used in specifying precedence predicates that constrain surface word order. These features are used to model notions from topological fields theory: `initial` corresponds to the Vorfeld, `middle` to the Mittelfeld and `final` to the Nachfeld.

DACHS employs the notion of precedence predicates from Hudson's Word Grammar (Hudson 1990) to state constraints on surface word order within domains. Bröker postulates two kinds of precedence predicates: the first orders a word with respect to the other elements in the domain it is associated with. The second states constraints on the order among the other elements in the domain.

3.3.2 Example

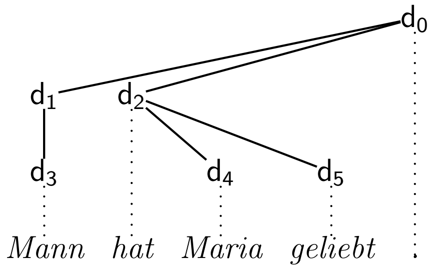
Here is an example sentence and its corresponding dependency tree analysis (3.44):

Den Mann hat Maria geliebt.
The man(*acc*) has Maria loved. (3.43)
"Maria has loved the man."



(3.44)

The following order domain structure corresponds to the dependency tree (3.44):



(3.45)

Here, the sentence period introduces the single top domain d_0 for the complete sentence. The finite verb *hat* is associated with domain d_2 and introduces by lexical assignment a sequence of domains $\langle \mathsf{d}_1, \mathsf{d}_2 \rangle$, requiring that d_1 must precede d_2. d_1 roughly corresponds to the Vorfeld in topological fields theory and d_2 to the Mittelfeld. *Mann*, *Maria* and *geliebt* introduce the order domains d_3, d_4 and d_5 on their own. Word order domains are sets of words. The word order domains in the example contain the following sets of words:

$$\begin{array}{lcl} \mathsf{d}_0 & = & \{\textit{Mann, hat, Maria, geliebt, .}\} \\ \mathsf{d}_1 & = & \{\textit{Mann}\} \\ \mathsf{d}_2 & = & \{\textit{hat, Maria, geliebt}\} \\ \mathsf{d}_3 & = & \{\textit{Mann}\} \\ \mathsf{d}_4 & = & \{\textit{Maria}\} \\ \mathsf{d}_5 & = & \{\textit{geliebt}\} \end{array} \tag{3.46}$$

Order within a domain is controlled by precedence predicates. Bröker posits the following conjunction of precedence predicates for finite verbs like *hat*:

$$\textit{hat} \;\Rightarrow\; <_* \;\wedge \tag{3.47}$$

$$\{\texttt{subj}, \texttt{obj}\} < \texttt{vpp} \tag{3.48}$$

(3.47) and (3.48) apply only to domains which have been introduced by *hat* and which also contain *hat*, i.e. d_2 in the example. The first conjunct (3.47) orders *hat* with respect to the other elements in such a domain: it is only satisfied if *hat* precedes all other words in the domain. The second conjunct (3.48) states a constraint on the order among the other elements in such a domain: it is satisfied if no subject or object follows the past participle ($\texttt{vpp}$). (3.47) is satisfied in the example since no word precedes *hat* in d_2. (3.48) is also satisfied because neither the object *Mann* (which is included in d_1 and d_3) nor the subject *Maria* precede the past participle *geliebt* in d_2.

3.3.3 Linking dependency and order domain structure

DACHS establishes the linking between dependency tree and word order domain structure as follows: a node in the dependency tree may either be inserted into a domain introduced by its direct (syntactical) head, or it may be extracted into a domain introduced by a transitive head, which is then called the *positional head* of the dependent.

In the example given in (3.45) above, the subject *Maria* is inserted into the domain $\mathtt{d}_2$ of its direct head *hat*. *Mann* is not inserted into the domain $\mathtt{d}_5$ of its direct head *geliebt*, but extracted into domain $\mathtt{d}_1$ introduced by its positional head *hat*.

The distance between positional heads and their dependents is restricted: for each node, Bröker defines a set of dependency relations which may link its direct and positional heads. Extraction is impossible if that set is empty: in this case, the positional head of the node equals its direct head.

To illustrate this idea, we refer again to the the example pictured in (3.45) above. Here, the extracted object *Mann* lexically defines the set $L = \{\mathtt{vpp}\}$ of dependency relations which may link its direct head with its positional head. The direct head of *Mann* is *geliebt*, and *geliebt* is a `vpp`-dependent of *hat*. Since $\mathtt{vpp} \in L$, the direct head *geliebt* may be linked to *hat*. *hat* is therefore the positional head of *Mann*.

3.3.4 Criticism

DACHS is highly similar to our theory of TDG (to be introduced in the chapters to follow). Firstly, DACHS also decouples dependency structure from linear structure. Dependency trees as used in DACHS represent the same structures as ID trees in TDG.

Still, DACHS and TDG differ in some respects. For one, DACHS does not utilize a primitive notion of topological fields but uses so-called *features* to model fields and also other concepts. The nature of these features is not entirely clear. Some features like `initial`, `middle` and `final` obviously model the fields Vorfeld, Mittelfeld and Nachfeld from topological fields theory. On the other hand, there are features like `norel`, whose interpretation is not as clear-cut.

Another point of criticism applies to the precedence predicates. Whereas TDG states ordering constraints on the level of topological fields (i.e. Vorfeld precedes Mittelfeld etc.), DACHS reverts back to dependency relations in the statement of its precedence predicates (see (3.48)). We regard this as a withdrawal from the original scheme of strictly decoupling dependency structure and surface word order.

In our opinion, constraints on surface word order should only be expressed on the word order domain level, and be free from any considerations regarding dependency relations.

Lastly, another divergence of DACHS and TDG is how the theories are formalized. DACHS makes use of modal logic in the spirit of (Blackburn 1994), whereas TDG opts for model-theoretic syntax casted in graph-theoretical terms.

3.4 Gerdes and Kahane

The formulation of a 'formal dependency grammar using a topological hierarchy' outlined in (Gerdes & Kahane 2001) is also called Topological Dependency Grammar. To distinguish it from our approach we call their framework TDG′ in the following. TDG′ places itself in the context of the general framework of Meaning-Text-Theory (MTT) (Mel'čuk 1988), which considers the modeling of a language as a bidirectional correspondence between meanings and texts. TDG′ corresponds to the syntactic module of MTT, providing the correspondence between syntactical dependency trees and morphological strings.

TDG′ posits two structurally dissimilar but linked analysis structures: a syntactic dependency tree and a topological structure. The syntactic dependency tree is an non-ordered dependency tree whose edges are labeled with syntactic relations. The syntactic dependency structure does not incorporate considerations regarding surface word order. The topological structure is a 'phrase structure' tree derived from the dependency tree using a set of correspondence rules.

3.4.1 Topological structure

A domain in TDG′ is a sequence of topological fields, and these fields are in turn assigned to words. For example, the *main domain* MD is a sequence of the following fields:

$$\mathsf{vf}, [, \mathsf{mf},], \mathsf{nf} \tag{3.49}$$

where vf corresponds to the Vorfeld, [to the left sentence bracket, mf to the Mittelfeld,] to the right sentence bracket and nf to the Nachfeld. If a word of a certain category lands in a certain field, it may create an embedded domain, which is again associated with a sequence of fields.

Gerdes and Kahane liken domains to 'boxes' and fields to 'compartments' inside a box. Compartments are linearly ordered within a box, and can again accommodate new boxes. This gives rise to a hierarchy of boxes which is the topological structure, and is also called ‘phrase structure’ in (Gerdes & Kahane 2001).

3.4.2 Example

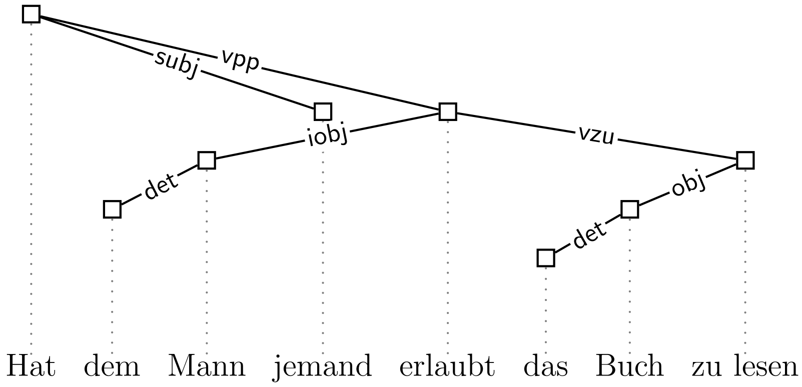
As an example, we go through a TDG′ analysis of the sentence glossed here:

Hat	dem	Mann	jemand	erlaubt,	das	Buch	zu lesen?
Has	the	man(dat)	anybody(nom)	allowed,	the	book	to read?

“Has anybody allowed the man to read the book?”

(3.50)

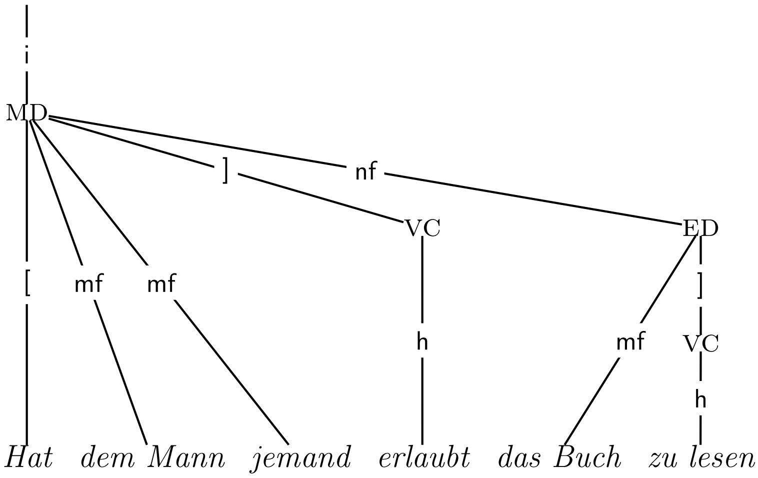
The corresponding syntactic dependency tree is depicted below:



(3.51)

Here, *jemand* is the subject of *hat* (grammatical role: subj) and *erlaubt* its past participle complement (vpp). *Mann* is the indirect object (iobj) of *erlaubt* and *zu lesen* its *zu*-infinitival complement (vzu), *Buch* is the object (obj) of *zu lesen.*

The syntactic dependency tree pictured in (3.51) above gives rise to the topological structure shown in (3.52):



(3.52)

Notice that we depict the topological structure as a phrase structure tree, contrary to the less conventional tree depictions in (Gerdes & Kahane 2001). The phrasal nodes are labeled by domain names like MD for *main domain*, the terminal nodes by words and the edges by field names like mf for Mittelfeld and nf for Nachfeld. i denotes the initial field.

Another notation for topological structures is labeled bracketed string notation. Below, we show the topological structure displayed in (3.52) above in labeled bracketed string notation:

$$[\textsc{md}\,\textit{Hat}_{[}\textit{dem Mann}_{\mathsf{mf}}\textit{jemand}_{\mathsf{mf}}[\textsc{vc}\,\textit{erlaubt}_{\mathsf{h}}]_{]}[\textsc{ed}\,\textit{das Buch}_{\mathsf{mf}}[\textsc{vc}\,\textit{zu lesen}_{\mathsf{h}}]_{]}]_{\mathsf{nf}}]_{\mathsf{i}} \quad (3.53)$$

Here, we write field names as subscripts and domain names follow opening brackets.

3.4.3 Grammar definition

A TDG′ grammar is defined in a rule-based fashion using four sets of rules:

- box creation rules
- box description rules
- field description rules
- correspondence rules

In addition, a *permeability order* is employed to restrict extraction.

Box creation rules

'Phrasal nodes' in the topological structure, i.e. domains or boxes, are introduced by *box creation rules*. A box creation rule $(\mathsf{c}, \mathsf{f}, \textsc{b}, \mathsf{f}')$ indicates that a word of category c placed in field f may create a domain $\textsc{b}$ and lands in field f' of that domain. Hence each word creating a domain is associated with two fields: the field f it is placed in and the field f' in the domain $\textsc{b}$ which it has created. Graphically, f is the label of its incoming edge, and f' the node label of the created domain.

The main domain for instance is introduced by the following box creation rule:

$$(\texttt{vfin}, \mathsf{i}, \textsc{md}, [) \tag{3.54}$$

indicating that a finite verb (category: `vfin`) such as *hat* in (3.52) placed in the initial field i may create a domain $\textsc{md}$ and lands in the left sentence bracket [of that domain.

If a non-finite verb (category: `vnonfin`) is placed in one of the so-called *major fields majf* such as Vorfeld (vf) or Nachfeld (nf), then it may create an embedded domain $\textsc{ed}$. The verb itself lands in the right sentence bracket] of that embedded domain:

$$(\texttt{vnonfin}, \mathit{majf}, \textsc{ed},]) \tag{3.55}$$

Note that (3.55) is not a box creation rule but a box creation rule schema: it expands to one rule for each major field. In the example topological structure (3.52) above, (3.55) induces an embedded domain $\textsc{ed}$ in the Nachfeld, covering the string *das Buch zu lesen.*

Box description rules

The order among the fields inside a domain or box is characterized by *box description rules*. A box description rule $\textsc{b} \rightarrow \mathsf{f}_1, \ldots, \mathsf{f}_n$ indicates that domain $\textsc{b}$ consists of the sequence of fields $\mathsf{f}_1, \ldots, \mathsf{f}_n$.

For instance, the sequence of fields in the main domain is described by the following box description rule:

$$\textsc{md} \quad \rightarrow \quad \mathsf{vf}, [, \mathsf{mf},], \mathsf{nf} \tag{3.56}$$

The order of fields in an embedded domain is reflected in this box description rule:

$$\textsc{ed} \quad \rightarrow \quad \mathsf{mf},], \mathsf{nf} \tag{3.57}$$

Field description rules

Field description rules state constraints on the cardinality of fields. A field description rule is a pair $(\mathsf{f}, w) \in F \times W$, where F is the set of all fields and $W = \{!, ?, +, *\}$ a set of wildcards. A field f must contain either exactly one element (!) , at a most one element (?), at least one element (+) or any number of elements ($*$).

As an example, Gerdes & Kahane (2001) constrain the number of elements in the left sentence bracket to exactly one by stating the field description rule $([, !)$ and in the Mittelfeld to an arbitrary number of elements by stating rule $(\mathsf{mf}, *)$.

Correspondence rules

Correspondence rules establish the linkage between syntactic dependency tree and topological structure. A correspondence rule looks like this:

$$(\mathsf{r}, \mathsf{c}_1, \mathsf{f}_1, \mathsf{c}_2, \mathsf{f}_2, \textsc{b}) \tag{3.58}$$

Such a rule indicates that if there is a dependency edge $w_1 {-}\mathsf{r}{\rightarrow} w_2$ from word w_1 with category c_1 (and placed in field f_1) to word w_2 with category c_2, labeled with the grammatical relation r, then w_2 may be placed in field f_2.

The last argument $\textsc{b}$ is used to control extraction or, as Gerdes and Kahane put it, *emancipation* out of domains. Therefore, they define for each TDG′ grammar a *permeability order*, which is a partial order on the set B of domains. By a correspondence rule such as (3.58), w_2 may emancipate out of domains with permeability $\leq \textsc{b}$.

The part of the permeability order given in (Gerdes & Kahane 2001) which is relevant for our purposes (including the domains $\textsc{vc}$, $\textsc{ed}$ and $\textsc{md}$) is shown below:

$$\textsc{vc} < \textsc{ed} < \textsc{md} \tag{3.59}$$

(3.59) indicates that extraction out of verb clusters ($\textsc{vc}$) is easier than extraction out of embedded domains ($\textsc{ed}$), and that in turn is easier than extraction out of main domains ($\textsc{md}$).

Here is an example correspondence rule schema: the positioning of a non-verbal element in a major field:

$$(\mathsf{r}, \mathtt{v}, \mathit{anyf}, \mathtt{nv}, \mathit{majf}, \textsc{ed}) \tag{3.60}$$

stating that if there is a dependency edge $w_1 {-}\mathsf{r}{\rightarrow} w_2$ from a verbal element (category: $\mathtt{v}$) w_1 placed in any field (anyf) to a non-verbal element (category: $\mathtt{nv}$) w_2

labeled with grammatical relation r, then the w_2 may land in a major field (*majf*). Thereby, w_2 may be emancipated out of domains with permeability $\leq$ ED.

An example of the operation of the permeability order (3.59) and correspondence rule (3.60) can be read off our example in (3.51) and (3.52) above: here, the indirect object *dem Mann* of *erlaubt* is emancipated into the Mittelfeld of the main domain. On its way, *diesen Mann* is extracted out of the verb cluster (VC) domain of *erlaubt*. The extraction is licensed since VC $\leq$ ED holds in (3.59).

3.4.4 Criticism

TDG′ (Gerdes & Kahane 2001) closely resembles our approach. Again, TDG′ posits two levels of representation, the non-ordered syntactic dependency tree (ID tree) and a topological structure (LP tree). However, there are a number of differences between the two approaches.

For one, whereas TDG (this study) makes use of dependency trees (where each node corresponds 1:1 to a word) on both levels, Gerdes and Kahane construct a phrase structure-like topological structure including phrasal (non-terminal) nodes.

Another crucial difference is that Gerdes and Kahane's approach is formalized in a rule-based fashion, as opposed to TDG and DACHS, which are highly lexicalized.

In spite of these differences, we think that TDG′ can be lexicalized and that the topological phrase structure analysis is finitely expandable. Thus, we suggest that it is possible to establish a mapping from TDG′ to TDG and vice versa. This way, TDG′ could make use of the computational techniques developed for TDG in order to attain a parser implementation. So far, Gerdes and Kahane (p.c.) have implemented a generator which obtains from a given dependency tree the set of all licensed topological structures, but they do not yet have a practical parser for TDG′.

3.5 Summary

In this chapter, we have discussed four rather similar approaches, two of them based on HPSG (Reape 1994), (Kathol 1995) and two based on dependency grammar (Bröker 1999), (Gerdes & Kahane 2001). All of them have in common the fundamental idea to dissociate vertical syntactic structure or immediate dominance from horizontal linear order or linear precedence. We showed that although Reape's theory of word order domains has proven very influential within and outside the

HPSG community, it is not free from flaws. Kathol's Linearization-Based Syntax overcomes some of these flaws. Bröker's adaptation of word order domains for dependency grammar is promising but falls foul of similar problems as Reape's theory: it does not incorporate a primitive notion of topological fields and does not satisfactorily dissociate immediate dominance and linear precedence. Gerdes and Kahane's TDG$'$ in turn can be criticized mostly for its rule-based rather than lexicalized character and its rather sketchy formalization.

In the following chapters, we will introduce a new way of stating constraints on linear precedence in the context of dependency grammar, which remedies the problems plaguing the approaches outlined in this chapter.

Chapter 4

LP analyses

In chapter 2, we introduced the notion of an ID analysis. In this chapter, we present the notion of a linear precedence (LP) analysis. An LP analysis consists of an LP tree, a lexical assignment, a total order on the set of nodes and a node label assignment. LP trees are ordered dependency trees representing topological structures in the spirit of topological fields theory. The well-formedness conditions for LP analyses comprise general principles and lexicalized principles. Because we do not yet formalize the well-formedness conditions in great detail, we refer to chapter 9 for a more in-depth account.

4.1 LP trees

An LP analysis consists mainly of a *topological dependency tree* or LP tree. In defining LP trees, we make use of notions borrowed from topological fields theory[1] as explained in chapter 1. The idea behind topological fields theory is to divide sentences into contiguous substrings and to assign to these substrings so-called topological fields. Here is an example analysis:

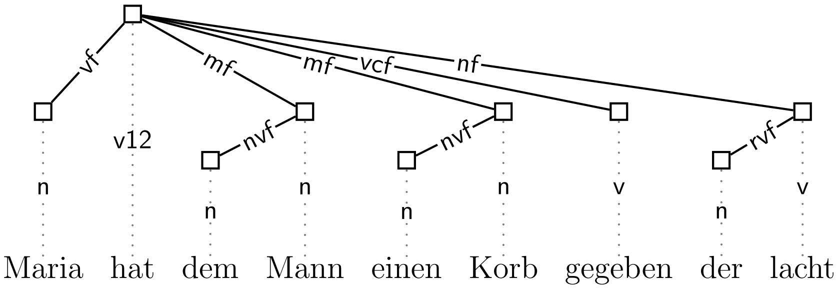
Vorfeld	(	Mittelfeld	)	Nachfeld
Maria *Maria*	*hat* *has*	*dem Mann einen Korb* *the man a basket*	*gegeben* *given*	*der lacht.* *who laughs.*

(4.1)

[1] Notice that although we apply TDG (and in particular the LP part) for the description of German surface word order here, TDG is intended to be a grammar formalism for any language, in the same way as topological fields theory can be applied in modified forms to other languages (Kathol 1995), (Penn 1999), (Kruijff 2001).

We partition the sentence into six substrings and assign them to five topological fields: *Maria* in the Vorfeld, *hat* in the left sentence bracket, *dem Mann* and *einen Korb* in the Mittelfeld, *gegeben* in the right sentence bracket and *der lacht* in the Nachfeld.

In our dependency-based setting, topological dependency trees or LP trees are headed structures: we distinguish *topological heads* and their *topological dependents.* In the example, *hat* is the topological head, *offering* the topological fields Vorfeld, Mittelfeld, right sentence bracket and Nachfeld for its topological dependents to land in. Here is an LP tree analysis of the example sentence:



(4.2)

As in ID trees, nodes in LP trees correspond 1:1 to words in the analyzed string. But while ID tree edges are labeled with grammatical roles, LP tree edges are labeled by topological fields. Also, each node in the LP tree is assigned a *node label* which we annotate on the dotted line. We give an overview of the topological fields used in the example LP tree (4.2) below:

field	meaning
vf	Vorfeld
mf	Mittelfeld
vcf	right sentence bracket
nf	Nachfeld
nvf	nominal Vorfeld

(4.3)

In (4.2), *Maria*, *Mann*, *Korb*, *gegeben* and *lacht* are topological dependents of the topological head *hat*. *Maria* is a dependent in the Vorfeld (vf), *Mann* and *Korb* are dependents in the Mittelfeld (mf), *gegeben* in the right sentence bracket (vcf) and *lacht* in the Nachfeld (nf). The determiners of the common nouns *Mann* and *Korb*, *dem* and *einen*, are topological dependents in the 'nominal Vorfeld'[2] (nvf) of the noun. Nouns and determiners are assigned the node label n, and verbs the node label v, except for the root *hat* which is assigned node label v12 where 'v' stands for verb and '1' and '2' for either verb-first or verb-second position. We use node

[2] We justify this terminology in chapter 7.

labels to position topological heads with respect to their topological dependents: e.g. in (4.2), v12 orders the finite verb *hat* (in the left sentence bracket) to the right of the Vorfeld (vf) and to the left of the Mittelfeld (mf).

4.2 LP principles

We turn now to the well-formedness conditions for LP analyses, which are a refinement of those for ID analyses. An LP analysis $(V, E_{\text{LP}}, \varepsilon, <, \mathsf{label}_{\text{N}})$ comprises a set of nodes V, a finite set of edges $E_{\text{LP}} \subseteq V \times V \times \mathcal{F}_{\mathsf{E}}$ (where $\mathcal{F}_{\mathsf{E}}$ is the set of edge labels[3]), a lexical assignment ε, a total order $<$ on the set of nodes and a node label assignment $\mathsf{label}_{\text{N}} : V \to \mathcal{F}_{\mathsf{N}}$ (where $\mathcal{F}_{\mathsf{N}}$ is the set of node labels). A valid LP analysis must satisfy the set of LP principles, comprising *general principles* and *lexicalized principles.*

We write f for an edge label in the set of edge labels $\mathcal{F}_{\mathsf{E}}$. An edge from node w to node w' labeled with f is written $w{-}f{\rightarrow}w'$, and we say that w' lands in the field f of w or *lands on w*. The set of node labels is $\mathcal{F}_{\mathsf{N}}$, and we write n for a node label in this set.[4]

4.2.1 General principles

Treeness principle

The treeness principle remains as in chapter 2.

Order principle

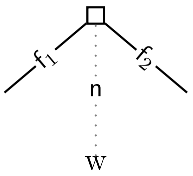
Principle 4.1 *Each analysis must be well-ordered.*

Consider the picture in (4.4) where the topological head w has two topological dependents labeled with f_1 and f_2 (where $f_1, f_2 \in \mathcal{F}_{\mathsf{E}}$) and w has node label n

[3] The $\mathcal{F}_{\mathsf{E}}$ set of edge labels corresponds to the set topological fields.

[4] The "N" subscripting $\mathcal{F}_{\mathsf{N}}$ stands for 'node' and the "E" subscripting $\mathcal{F}_{\mathsf{E}}$ for 'edge'.

(where $\mathsf{n} \in \mathcal{F}_{\mathsf{N}}$):



(4.4)

Contrary to ID analyses, which are non-ordered, LP analyses are ordered trees. To establish this order, we posit a *total order* $\prec$ on the set $\mathcal{F}_{\mathsf{E}}$ of edge labels:

$$f_1 \prec f_2 \tag{4.5}$$

(4.5) induces a total order on the *yields*, i.e. the set of nodes reachable by traversing downwards zero or more LP edges of the topological dependents of a node. Thus by (4.5), the yields of f_1-labeled topological dependents must precede the yields of f_2-labeled dependents. The induced order is partial: if two topological dependents land in the same field, they are not ordered with respect to each other.

As a concrete example, consider the LP tree given in (4.2) above. We order the set $\mathcal{F}_{\mathsf{E}}$ of topological fields occuring in this tree as follows:

$$\mathsf{nvf} \prec \mathsf{vf} \prec \mathsf{mf} \prec \mathsf{vcf} \prec \mathsf{nf} \tag{4.6}$$

(4.6) induces the following partial order among the yields of the topological dependents of *hat* in LP tree (4.2):

$$\begin{array}{ccccccc} \mathsf{vf} & \prec & \mathsf{mf} & \prec & \mathsf{vcf} & \prec & \mathsf{nf} \\ [\textit{Maria}] & \prec & [\textit{dem Mann}] & \prec & [\textit{gegeben}] & \prec & [\textit{der lacht}] \\ & & [\textit{einen Korb}] & & & & \end{array} \tag{4.7}$$

The induced order is partial because the two NPs *dem Mann* and *einen Korb* both land in the same field (the mf of *hat*) and are thus not ordered with respect to each other.

The order on the set $\mathcal{F}_{\mathsf{E}}$ of edge labels does not tell us the position of the topological head with respect to its topological dependents. We position topological heads with respect to their topological dependents using *node labels*. Therefore, we extend the total order on the set $\mathcal{F}_{\mathsf{E}}$ of edge labels to a total order on the set $\mathcal{F} = \mathcal{F}_{\mathsf{E}} \uplus \mathcal{F}_{\mathsf{N}}$ of edge and node labels. For instance the total order shown in (4.8) results from extending the order in (4.5) with node label n between f_1 and f_2:

$$f_1 \prec \mathsf{n} \prec f_2 \tag{4.8}$$

In (4.4), w is assigned node label n. Therefore, (4.8) induces an order where the topological head w is positioned between the yields of its topological dependents labeled with f_1 and f_2.

In the concrete example, (4.7) does not tell us where the topological head *hat* should occur in this sequence. Thus, we extend the total order on the set of edge labels given in (4.6) to a total order on the set $\mathcal{F} = \mathcal{F}_{\mathsf{E}} \uplus \mathcal{F}_{\mathsf{N}}$ of edge and node labels:

$$\mathsf{nvf} \prec \mathsf{n} \prec \mathsf{vf} \prec \mathsf{v12} \prec \mathsf{mf} \prec \mathsf{vcf} \prec \mathsf{v} \prec \mathsf{nf} \tag{4.9}$$

Because *hat* has node label v12 (verb-first or verb-second position) in LP tree (4.2), it must occur between the vf and the mf by the total order (4.9):

$$\begin{array}{ccccccccc} \mathsf{vf} & \prec & \mathsf{v12} & \prec & \mathsf{mf} & \prec & \mathsf{vcf} & \prec & \mathsf{nf} \\ [\textit{Maria}] & \prec & \textit{hat} & \prec & [\textit{dem Mann}] & \prec & [\textit{gegeben}] & \prec & [\textit{der lacht}] \\ & & & & [\textit{einen Korb}] & & & & \end{array} \tag{4.10}$$

To sum up, the order principle consists of the following conditions, given a total order $\prec$ on the set $\mathcal{F} = \mathcal{F}_{\mathsf{E}} \uplus \mathcal{F}_{\mathsf{N}}$ of topological fields (edge labels) and node labels:

$$w\text{–}f_1\text{→}w_1 \ \wedge\ w\text{–}f_2\text{→}w_2 \ \wedge\ f_1 \prec f_2 \ \Rightarrow\ w_1 < w_2 \tag{4.11}$$

$$w_1\text{→}^* w_1' \ \wedge\ w_2\text{→}^* w_2' \ \wedge\ w_1 < w_2 \ \Rightarrow\ w_1' < w_2' \tag{4.12}$$

$$w\text{–}f\text{→}w_1 \ \wedge\ f \prec \mathsf{label}_{\mathrm{N}}(w) \ \Rightarrow\ w_1 < w \tag{4.13}$$

$$w\text{–}f\text{→}w_1 \ \wedge\ \mathsf{label}_{\mathrm{N}}(w) \prec f \ \Rightarrow\ w < w_1 \tag{4.14}$$

where $<$ is a total order on the set V of nodes. (4.11) orders the daughters of a node w with respect to each other. (4.12) states that if w_1 precedes w_2, then all nodes in the yield of w_1 must also precede all nodes in the yield of w_2. (4.13) and (4.14) orders mothers with respect to their daughters using node labels.

An LP analysis $(V, E_{\mathrm{LP}}, \varepsilon, <, \mathsf{label}_{\mathrm{N}})$ is well-formed under the order principle if the conditions (4.11–4.14) are satisfied. We say that the LP analysis is then *well-ordered.*

Projectivity principle

Principle 4.2 *Each analysis must be projective.*

LP analyses are *projective*: the yield[5] of each node must cover a contiguous substring. The projectivity principle is a prerequisite for using topological fields theory: topological fields theory orders *contiguous substrings* with respect to each other, and only projective analyses give rise to contiguous substrings.

[5] Roughly, the yield of a node in a dependency analysis can be regarded as a phrase structure constituent: for example the yield of a non-finite verb node corresponds to a VP-constituent, and the yield of a noun to an NP.

An LP analysis $(V, E_{\text{LP}}, \varepsilon, <, \mathsf{label}_{\text{N}})$ is well-formed under the projectivity principle only if the yield of each node covers a contiguous substring.

4.2.2 Lexicalized principles

We further restrict the number of admissible LP analyses by lexicalized principles. The set of lexicalized principles for LP analyses contains the *accepted edge labels principle*, the *valency principle* and the *edge constraints* (which remain as before for ID analyses) and the new *accepted node labels principle.*

For stating lexicalized principles, we pose the set $\mathcal{A}_{\text{LP}} = \{\mathsf{labels}_{\text{LP}}, \mathsf{valency}_{\text{LP}}, \mathsf{labels}_{\text{N}}\}$ of *LP attributes.* We employ $\mathsf{labels}_{\text{LP}}$ in the formulation of the accepted edge labels principle, $\mathsf{valency}_{\text{LP}}$ for the valency principle and $\mathsf{labels}_{\text{N}}$ for the accepted node labels principle.

Accepted edge labels principle

The accepted edge labels principle remains the same as in chapter 2. Here, the principle makes use of the lexical attribute $\mathsf{labels}_{\text{LP}} : \mathcal{E}_t \to 2^{\mathcal{F}_{\mathsf{E}}}$ which maps lexical types to sets of accepted edge labels. We call the value of the $\mathsf{labels}_{\text{LP}}$-attribute *accepted fields.*

Valency principle

We have already presented the valency principle in chapter 2. Here, the principle employs the lexical attribute $\mathsf{valency}_{\text{LP}} : \mathcal{E}_t \to 2^{v(\mathcal{F}_{\mathsf{E}})}$ mapping lexical types to sets of field valency specifications. We call the value of the $\mathsf{valency}_{\text{LP}}$-attribute *field valency.* If either of the valency specifications *f*, *f?* or *f** is in $\mathsf{valency}_{\text{LP}}(w)$, we say that w *offers f.* Using terminology from GB theory, we say that *f* is then a *landing site* for other nodes.

Accepted node labels principle

Principle 4.3 *Each node must accept its node label.*

Each node is lexically assigned a set of possible node labels using the lexical attribute $\mathsf{labels}_{\text{N}} : \mathcal{E}_t \to 2^{\mathcal{F}_{\mathsf{N}}}$ mapping lexical types to sets of node labels. The node label assignment $\mathsf{label}_{\text{N}} : V \to \mathcal{F}_{\mathsf{N}}$ maps each node to its node label. The accepted node labels principle is defined as follows for all nodes $w \in V$:

$$\mathsf{label}_{\text{N}}(w) \in \mathsf{labels}_{\text{N}}(w) \tag{4.15}$$

An LP analysis $(V, E_{\text{LP}}, \varepsilon, <, \mathsf{label}_{\text{N}})$ is well-formed under the accepted node labels principle only if (4.15) holds for all nodes $w \in V$.

Edge constraints

The edge constraints principle remains as introduced in chapter 2. We call edge constraints for LP analyses *LP edge constraints*.

4.3 Examples

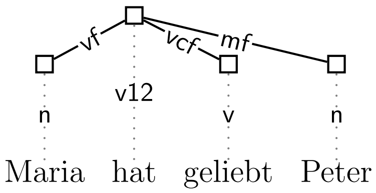
In this section, we illustrate the well-formedness conditions for LP analyses by providing examples. We begin with specifying a total order on the set $\mathcal{F} = \mathcal{F}_{\mathsf{E}} \uplus \mathcal{F}_{\mathsf{N}}$ of fields $\mathcal{F}_{\mathsf{E}} = \{\mathsf{vf}, \mathsf{mf}, \mathsf{vcf}, \mathsf{nf}\}$ and node labels $\mathcal{F}_{\mathsf{N}} = \{\mathsf{n}, \mathsf{v12}, \mathsf{v}\}$:

$$\mathsf{n} \prec \mathsf{vf} \prec \mathsf{v12} \prec \mathsf{mf} \prec \mathsf{vcf} \prec \mathsf{v} \prec \mathsf{nf} \tag{4.16}$$

We present in Figure 4.1 an example lexicon including the proper names *Maria* and *Peter*, the transitive past participle *geliebt* (*loved*) and the finite perfect auxiliary *hat* (*has*). Like before, the lexical assignment ε straightforwardly maps words to their corresponding lexical entries. Both the node label assignment $\mathsf{label}_{\text{N}}$ and the total order $<$ assumed in each example are those implicitly used in the graphical rendition its LP tree.

4.3.1 Order principle violation

The LP tree below violates the order principle:



(4.17)

$$(\textit{Maria} \quad , \begin{bmatrix} \mathsf{labels}_{\mathrm{LP}} & : & \{\mathsf{vf},\mathsf{mf}\} \\ \mathsf{labels}_{\mathrm{N}} & : & \{\mathsf{n}\} \\ \mathsf{valency}_{\mathrm{LP}} & : & \emptyset \end{bmatrix})$$

$$(\textit{Peter} \quad , \begin{bmatrix} \mathsf{labels}_{\mathrm{LP}} & : & \{\mathsf{vf},\mathsf{mf}\} \\ \mathsf{labels}_{\mathrm{N}} & : & \{\mathsf{n}\} \\ \mathsf{valency}_{\mathrm{LP}} & : & \emptyset \end{bmatrix})$$

$$(\textit{geliebt} \quad , \begin{bmatrix} \mathsf{labels}_{\mathrm{LP}} & : & \{\mathsf{vf},\mathsf{vcf}\} \\ \mathsf{labels}_{\mathrm{N}} & : & \{\mathsf{v}\} \\ \mathsf{valency}_{\mathrm{LP}} & : & \{\mathsf{mf}*\} \end{bmatrix})$$

$$(\textit{hat} \quad , \begin{bmatrix} \mathsf{labels}_{\mathrm{LP}} & : & \emptyset \\ \mathsf{labels}_{\mathrm{N}} & : & \{\mathsf{v12}\} \\ \mathsf{valency}_{\mathrm{LP}} & : & \{\mathsf{vf}?,\mathsf{mf}*,\mathsf{vcf}?,\mathsf{nf}?\} \end{bmatrix})$$

Figure 4.1: Example lexicon

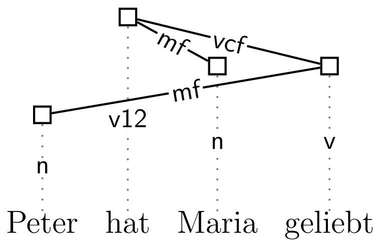
Here, *geliebt* in the vcf precedes *Peter* in the mf, but the total order given in (4.16) states that $\mathsf{mf} \prec \mathsf{vcf}$. Thus, condition (4.11) of the order principle is violated:

$$\textit{hat}\text{–}\mathsf{mf}\!\rightarrow\!\textit{Peter} \;\wedge\; \textit{hat}\text{–}\mathsf{vcf}\!\rightarrow\!\textit{geliebt} \;\wedge\; \mathsf{mf} \prec \mathsf{vcf} \quad\Rightarrow\quad \textit{Peter} < \textit{geliebt} \tag{4.18}$$

because *geliebt* < *Peter* in LP tree (4.17).

4.3.2 Projectivity principle violation

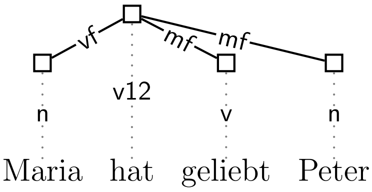
The following LP tree violates the projectivity principle since the yield of *geliebt* (*Peter geliebt*) does not cover a contiguous string but is interrupted by *hat* and *Maria*, resulting in a non-projective LP tree:



(4.19)

4.3.3 Accepted edge labels principle violation

The LP tree below exhibits a violation of the accepted edge labels principle:



(4.20)

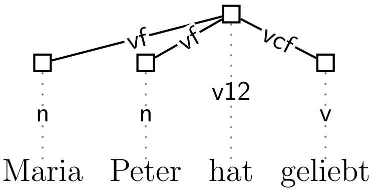
The incoming edge of *geliebt* is labeled with field mf, but mf is not in the set of accepted fields of *geliebt*. Thus, the accepted edge labels constraint instantiated below is violated:

$$\textit{hat}\text{—}\mathsf{mf}\text{→}\textit{geliebt} \quad\Rightarrow\quad \mathsf{mf} \in \mathsf{labels}_{\text{LP}}(\textit{geliebt}) \tag{4.21}$$

because $\mathsf{mf} \notin \mathsf{labels}_{\text{LP}}(\textit{geliebt}) = \{\mathsf{vf}, \mathsf{vcf}\}$.

4.3.4 Valency principle violation

The following LP tree violates the valency principle:



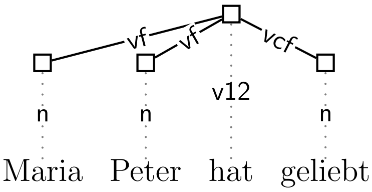
(4.22)

By the lexical assignment, *hat* only offers at most one position for topological dependents in the Vorfeld by field valency specification vf?, but in the tree above there are two vf-dependents. Thus, the LP tree violates the instantiation of the valency principle displayed below:

$$\mathsf{vf?} \in \mathsf{valency}_{\text{LP}}(\textit{hat}) \quad\Rightarrow\quad |\mathsf{vf}(\textit{hat})| \leq 1 \tag{4.23}$$

4.3.5 Accepted node labels principle violation

The LP tree below violates the accepted node labels principle:



(4.24)

Here, *geliebt* has node label n, but n is not in the set of accepted node labels of *geliebt*. Hence, the accepted node labels constraint instantiated below is violated:

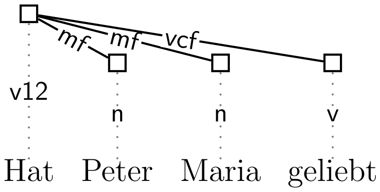
$$\mathsf{label}_{\text{N}}(\textit{geliebt}) \quad \in \quad \mathsf{labels}_{\text{N}}(\textit{geliebt}) \tag{4.25}$$

because

$$\mathsf{label}_{\text{N}}(\textit{geliebt}) = \mathsf{n} \quad \notin \quad \mathsf{labels}_{\text{N}}(\textit{geliebt}) = \{\mathsf{v}\} \tag{4.26}$$

4.3.6 Well-formed LP tree

The following LP tree is well-formed under the total order (4.16) and the lexical assignment:



(4.27)

Here, the total order in (4.16) induces the following ordering, where the two nouns *Peter* and *Maria* in the Mittelfeld (mf) are not ordered with respect to each other:

$$\begin{array}{ccccc} \mathsf{v12} & \prec & \mathsf{mf} & \prec & \mathsf{vcf} \\ [\textit{Hat}] & \prec & [\textit{Peter}] & \prec & [\textit{geliebt}] \\ & & [\textit{Maria}] & & \end{array} \tag{4.28}$$

4.4 Summary

We introduced the notion of a linear precedence (LP) analysis. An LP analysis consists of an LP tree, a lexical assignment, a total order on the set of nodes and a node label assignment. LP trees represent topological structures in the spirit of topological fields theory but from a dependency-based perspective: we distinguish topological heads and topological dependents, and order the yields of topological dependents according to a total order on the set of edge labels.

The next chapter brings together the notions of ID and LP analyses: we present the concept of an ID/LP analysis consisting of an ID and an LP analysis sharing the same lexical assignment. We relate ID and LP analyses to each other by positing ID/LP principles associated with the notion of climbing.

Chapter 5

ID/LP analyses

In this chapter, we bring together the notions of ID and LP analyses characterized in the previous chapters. We define an ID/LP analysis as a pair of an ID analysis and an LP analysis sharing the same lexical assignment, and relate ID and LP analyses to each other by the climbing principle. As a result, the shape of the LP tree is a flattening of the ID tree's. We refine the climbing principle by the subtrees principle and the barriers principle.

5.1 ID/LP principles

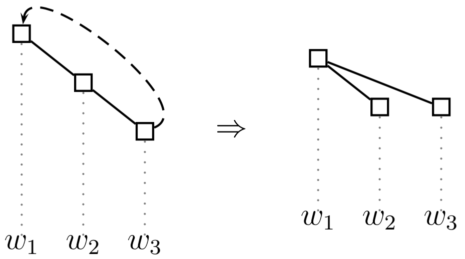
An ID/LP analysis is a pair of an ID analysis $(V, E_{\text{ID}}, \varepsilon)$ and an LP analysis $(V, E_{\text{LP}}, \varepsilon, <, \mathsf{label}_{\text{N}})$. Both analyses share the same lexical assignment ε. $<$ is a total order on the set V of nodes and $\mathsf{label}_{\text{N}}$ a node label assignment. We specify a set of well-formedness conditions called *ID/LP principles* to relate the ID and the LP analyses to each other. As a result, both analyses are mutually constraining. In the version of TDG presented in this thesis, we posit three ID/LP principles: the *climbing principle*, the *subtrees principle* and the *barriers principle*.

5.1.1 Climbing principle

The elementary idea behind climbing is that nodes in the ID tree may 'climb up' from a lower position in the ID tree to a position higher in the LP tree. As a result, the shape of the LP tree is a flattening of the ID tree's, in the same sense as in similar approaches (such as those presented in chapter 3). In the paradigm of dependency grammar, climbing can be likened to what is called *lifting* in (Kahane et al. 1998) and *emancipation* in (Gerdes & Kahane 2001). Climbing is also similar to *movement* in GB (Debusmann 2001). Here is the climbing principle:

Principle 5.1 *A node must land on a transitive head*[1]*.*

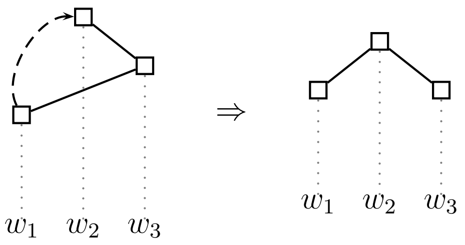
How does the climbing principle bring about a flattening of LP trees with respect to ID trees? Here is an abstract example:



(5.1)

On the left hand side of (5.1) is an ID tree and on the right hand side an LP tree. We omit edge and node labels for simplicity. As indicated by the dashed arrow in the ID tree, node w_3 'climbs up' to node w_1, resulting in an LP tree on the right hand side where w_3 is a daughter of w_1. Observe that w_1 is a *transitive head* of w_3 (as required by principle 5.1) by virtue of being in the set of nodes above w_3 in the ID tree. Obviously, the LP tree on the right hand side is flatter than the ID tree on the left hand side: it has only depth one instead of depth two.

The following example shows that by flattening, we can turn a non-projective ID tree into a projective LP tree:



(5.2)

The ID tree on the left hand side of (5.2) is non-projective: the set of nodes reachable by traversing downwards zero or more nodes from w_3, i.e. $\{w_3, w_1\}$, does not represent a contiguous sequence because w_2 interrupts the two. Contrarily, the LP tree on the right hand side of (5.2) is projective: w_1 has climbed up to w_2. Thus, climbing has resulted in an LP tree whose shape is a flattening of the ID tree and has also transformed the non-projective ID tree into a projective LP tree.

[1] The set of transitive heads of a node w' contains all nodes above w' in the ID tree. The term is due to Norbert Bröker (Bröker 1999).

Forced climbing

Climbing can be 'forced' by means of the lexicon. This is achieved by simultaneously specifying role and field valency in a way that a syntactic dependent of a node cannot be a topological dependent of the node in the LP tree. Following the climbing principle, the only 'way out' for the syntactic dependent is to climb up to a node higher in the tree which offers an appropriate field for it.

As an example, here is an abbreviated lexical entry[2] for the ditransitive past participle verb *gegeben*:

$$(\textit{gegeben}\ ,\ \begin{bmatrix} \mathsf{valency}_{\mathrm{ID}} & : & \{\mathsf{adv}*, \mathsf{pp}?, \mathsf{obj}, \mathsf{iobj}\} \\ \mathsf{valency}_{\mathrm{LP}} & : & \emptyset \\ & \ldots & \end{bmatrix}) \tag{5.3}$$

gegeben subcategorizes for an object and an indirect object in its role valency. It can additionally be modified by adverbs and a PP. In its field valency, *gegeben* offers no field.

Nouns like for example *Mann* and *Korb* can only land in the Vorfeld (vf) or the Mittelfeld (mf), as indicated by the following abbreviated lexical entries:

$$(\textit{Mann}\ ,\ \begin{bmatrix} \mathsf{labels}_{\mathrm{ID}} & : & \{\mathsf{subj}, \mathsf{obj}, \mathsf{iobj}\} \\ \mathsf{labels}_{\mathrm{LP}} & : & \{\mathsf{vf}, \mathsf{mf}\} \\ & \ldots & \end{bmatrix}) \tag{5.4}$$

$$(\textit{Korb}\ ,\ \begin{bmatrix} \mathsf{labels}_{\mathrm{ID}} & : & \{\mathsf{subj}, \mathsf{obj}, \mathsf{iobj}\} \\ \mathsf{labels}_{\mathrm{LP}} & : & \{\mathsf{vf}, \mathsf{mf}\} \\ & \ldots & \end{bmatrix}) \tag{5.5}$$

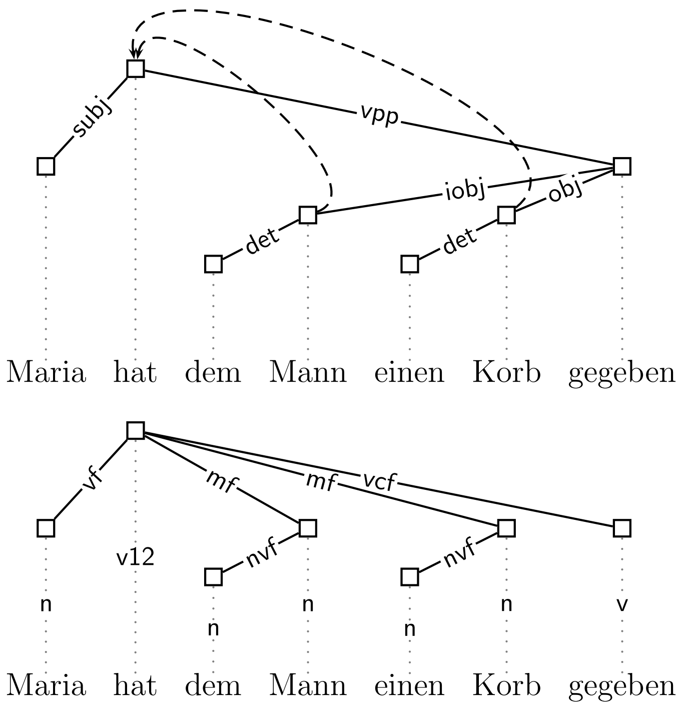
And finite verbs in verb-second position such as *hat* offer the fields vf, mf, vcf and nf in their field valency:

$$(\textit{hat}\ ,\ \begin{bmatrix} \mathsf{valency}_{\mathrm{LP}} & : & \{\mathsf{vf}?, \mathsf{mf}*, \mathsf{vcf}, \mathsf{nf}?\} \\ & \ldots & \end{bmatrix}) \tag{5.6}$$

Now consider sentence (5.7) below. We show an ID/LP analysis of it in (5.8) where the ID tree is depicted in the upper half and the LP tree in the lower half:

Maria hat dem Mann einen Korb gegeben.
Maria has the man a basket given. (5.7)
"Maria has given the man a basket."

[2] We indicate by ellipsis that the lexical entry is incomplete.


subj
vpp
iobj
obj
det
det
Maria hat dem Mann einen Korb gegeben
vf
mf
mf
vcf
nvf
nvf
v12
n
n
n
n
n
v
Maria hat dem Mann einen Korb gegeben

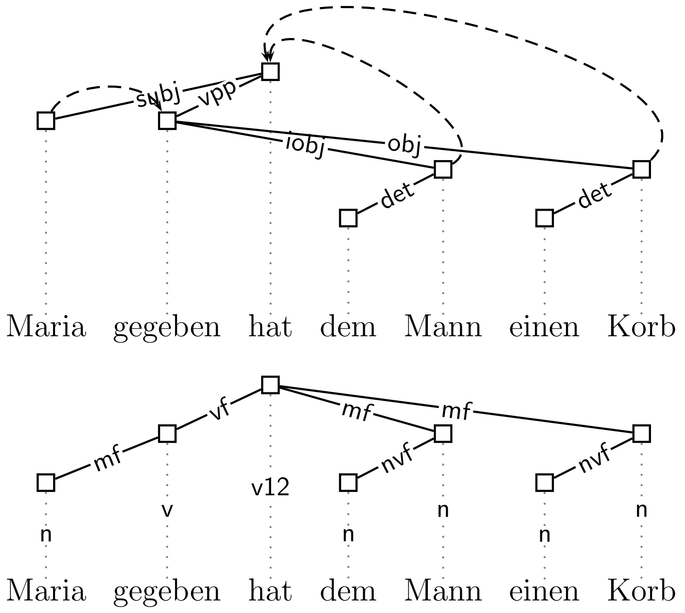
(5.8)

In (5.8), climbing was ‘forced’ by the participating lexical entries: although *Mann* and *Korb* are syntactic dependents of *gegeben* in the ID tree, they cannot be topological dependents of *gegeben* in the LP tree since *gegeben*, by virtue of its field valency, does not offer a field. Due to the climbing principle, the only ‘way out’ for the two nominal complements is to climb up to a node higher in the tree which offers an appropriate field (vf or mf) for them. The only node higher in the tree is the root *hat*. In its field valency, *hat* offers both vf and mf. Thus, as in the example, *Mann* and *Korb* can both land in the mf of *hat*.

Climbing principle violation

To further illustrate the climbing principle, we exhibit in (5.10) an ID/LP analysis that violates the climbing principle, and thereby gives rise to the following ungrammatical linearization:

∗Maria	gegeben	hat	dem	Mann	einen	Korb.	(5.9)
Maria	given	has	the	man(dat)	a	basket(acc).	


subj
vpp
iobj
obj
det
det
Maria gegeben hat dem Mann einen Korb
vf
mf
mf
mf
nvf
nvf
v12
n
v
n
n
n
n
Maria gegeben hat dem Mann einen Korb

(5.10)

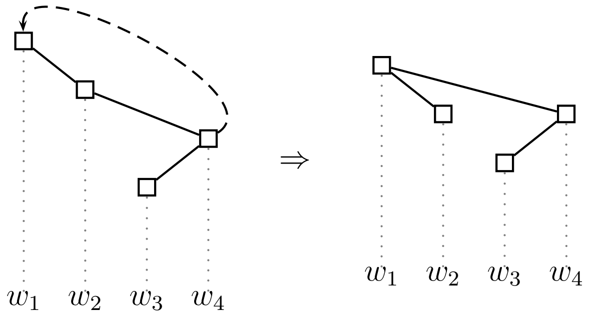
ID/LP analysis (5.10) is invalid because the node corresponding to *Maria* does not climb up but moves to its sister *gegeben*.

5.1.2 Subtrees principle

We refine the notion of climbing embodied by the climbing principle by the subtrees principle. The idea behind this well-formedness condition is that if a node climbs up, it must take its entire subtree along:

Principle 5.2 *A node must land on, or climb higher than its syntactic head.*

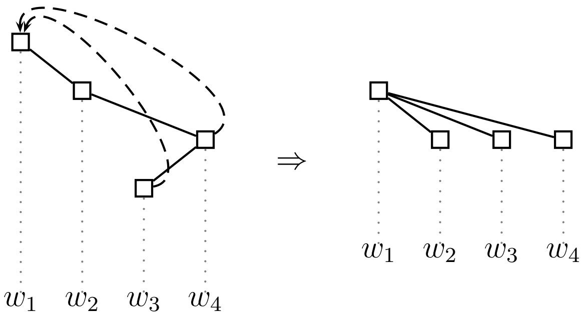
We illustrate the subtrees principle by an example. Consider the ID/LP analysis shown below, where w_4 climbs up to w_1:



(5.11)

By the subtrees principle, w_3 must either land on or climb higher than its (syntactic) head w_4. In (5.11), it lands on its head w_4. In (5.12) below, it climbs higher than

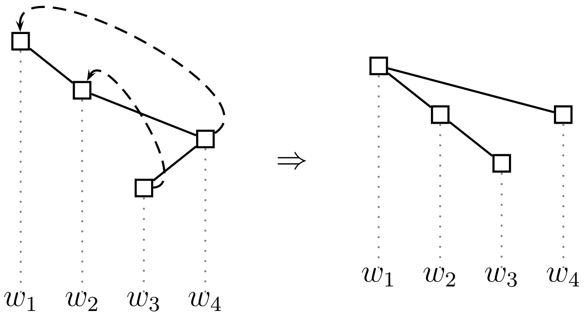
w_4 and lands on w_1:



(5.12)

Subtrees principle violations

In (5.13), we display an example where the subtrees principle is violated because w_3 neither lands on its head w_4 nor climbs higher than w_4, but it lands on w_2:

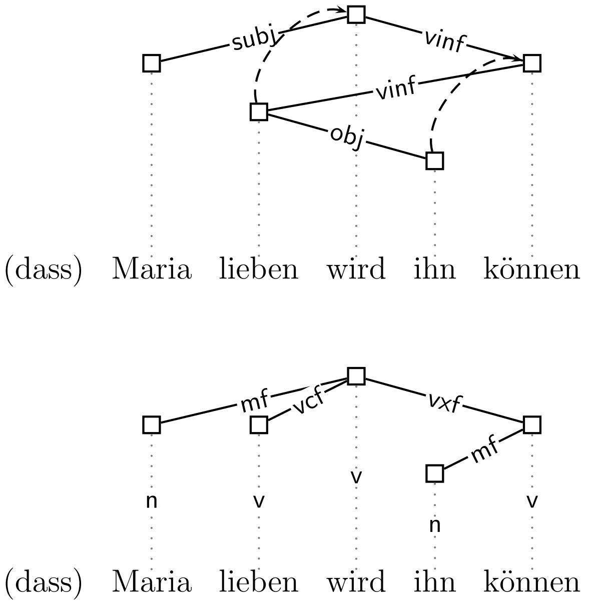


(5.13)

ID/LP analysis (5.15) also displays a violation of the subtrees principle and thereby gives rise to the ungrammatical linearization below:

*(dass)	Maria	lieben	wird	ihn	können
(that)	Maria	love	will	him	can

(5.14)


subj
vinf
vinf
obj
(dass) Maria lieben wird ihn können
mf
vcf
vxf
mf
n
v
v
n
v
(dass) Maria lieben wird ihn können

(5.15)

In (5.15), *lieben* climbs up into the vcf of *wird* but does not take its entire subtree along. In particular, *lieben* 'leaves behind' its object *ihn* in the mf of *können*.

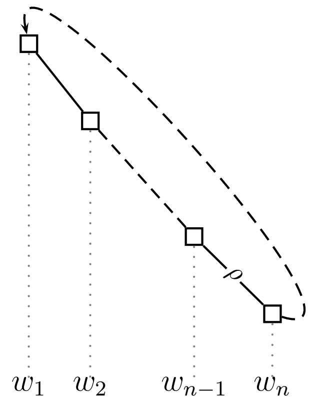
5.1.3 Barriers principle

We further restrict climbing by the lexicalized barriers principle:

Principle 5.3 *A node may not climb through a barrier.*

We only consider a simple notion of barrier. The essential idea is to prevent a node w' from 'climbing through' another node w. Which nodes are allowed to climb through w and which are not is determined by the grammatical role of w'. The barriers principle applies the ID/LP attribute blocks : $\mathcal{E}_t \to 2^{\mathcal{R}}$ which maps each lexical type to a set of grammatical roles which it 'blocks'.

As an example, consider the following abstract example ID tree:



(5.16)

In (5.16), w_n climbs up to its transitive head w_1, and by doing that, it climbs through nodes $w_{n-1}, \ldots, w_2$. w_n has role ρ. Any of the nodes w_i (for $2 \leq i \leq n-1$) blocks w_n if ρ is in the set of roles blocked by w_i. If a node w_i blocks w_n, w_n cannot climb up through w_i (to a node above w_i).

Barriers principle violation

Here is an example illustrating the barriers principle. We define a lexicon including the common nouns *Frau* (*woman*) and *Mannes* (*man*) and the definite determiner *des* (*the(gen)*). The assumed lexical assignment ε is again a straightforward mapping of nodes to their corresponding lexical entries. We begin with the lexical entry for *Frau*:

$$(\textit{Frau} \quad , \quad \left[\begin{array}{lcl} \textsf{valency}_{\text{ID}} & : & \{\textsf{det?}, \textsf{genmod?}\} \\ \textsf{valency}_{\text{LP}} & : & \{\textsf{nvf?}, \textsf{nxf?}\} \\ & \ldots & \end{array}\right]) \tag{5.17}$$

(5.17) subcategorizes for an optional determiner (grammatical role: det) and an optional genitival modifier (genmod). In its field valency, it offers the fields nvf and nxf. nvf stands for 'nominal Vorfeld' and is the position of a noun's determiner or genitival modifier to the left of the noun. nxf stands for 'nominal extraposition field' and is the position of a genitival modifier following the noun.[3]

The lexical entry for the genitive determiner *des* is displayed in the following:

$$(\textit{des} \quad , \quad \left[\begin{array}{lcl} \textsf{labels}_{\text{ID}} & : & \{\textsf{det}\} \\ \textsf{labels}_{\text{LP}} & : & \{\textsf{nvf}\} \\ & \ldots & \end{array}\right]) \tag{5.18}$$

[3] We justify this terminology in chapter 7.

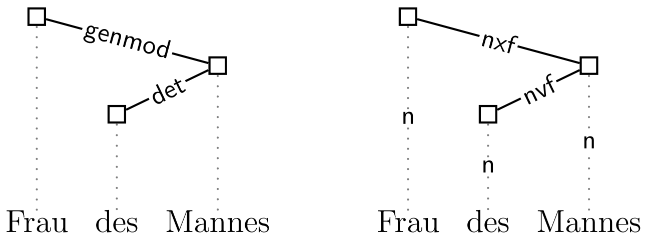
The only accepted role of *des* is det. It can only land in the nominal Vorfeld nvf.

And here is the lexical entry for the genitive noun *Mannes*:

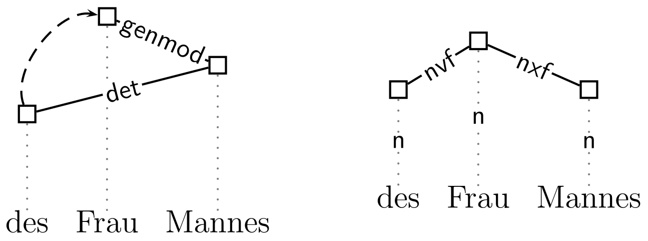
$$(\textit{Mannes}\ ,\ \begin{bmatrix} \mathsf{labels}_{\mathrm{ID}} & : & \{\mathsf{genmod}\} \\ \mathsf{valency}_{\mathrm{ID}} & : & \{\mathsf{det?}, \mathsf{genmod?}\} \\ \mathsf{labels}_{\mathrm{LP}} & : & \{\mathsf{nvf}, \mathsf{nxf}\} \\ \mathsf{valency}_{\mathrm{LP}} & : & \{\mathsf{nvf?}, \mathsf{nxf?}\} \\ & \dots & \end{bmatrix}) \tag{5.19}$$

Mannes accepts only role genmod. As a genitival modifier, it can land either in the nvf to the left of the noun it modifies or in the nxf to the right.

Without the barriers principle, the lexical entries above license (besides others) the following two ID/LP tree analyses:



(5.20)



(5.21)

In the LP tree on the right hand side of (5.20), *Mannes* lands in the nominal extraposition field (nxf) to the right of *Frau*. The determiner *des* lands in the nominal Vorfeld (nvf) of *Mannes*. In the LP tree in (5.21), *Mannes* still resides in the nxf of *Mannes*, but *des* has climbed up into the nvf of *Frau*.

Only the first analysis (5.20) represents a grammatical linearization, while (5.21) is ungrammatical: determiners like *des* cannot be extracted out of their NPs in German. In other words, we must prevent that determiners climb through nouns. To this end, we refine the lexical entry for the noun *Mannes*:

$$(\textit{Mannes}\ ,\ \begin{bmatrix} \mathsf{blocks} & : & \{\mathsf{det}\} \\ & \dots & \end{bmatrix}) \tag{5.22}$$

By (5.22), *Mannes* now blocks determiners (role det) and the barriers principle excludes ID/LP analysis (5.21) because *des* cannot climb through *Mannes* anymore (and hence cannot climb up to *Frau*).

5.2 Summary

We defined an ID/LP analyses to be a pair of an ID and an LP analysis sharing the same lexical assignment. A set of well-formedness conditions called ID/LP principles relate ID and LP analyses to each other. As a result, the two analyses are mutually constraining. The elementary idea behind the ID/LP principles is that of climbing: nodes can climb up from a position lower in the ID tree to a position higher up in the LP tree. Climbing causes the shape of the LP tree to be a flattening of the ID tree's. The central climbing principle is refined by two additional ID/LP principles: the subtrees principle and the barriers principle. The subtrees principle requires that only whole subtrees may climb up. The barriers principle is a lexicalized principle and restricts extraction possibilities.

In the following chapter, we introduce the TDG lexicon.

Chapter 6

ID/LP Lexicon

This chapter presents the TDG ID/LP lexicon which is a finite binary relation between strings and lexical types. The lexicon consists of lexical types arranged in a lexical type hierarchy and lexical entries which can be obtained from the lexical types by lexical inheritance. By inheritance, we improve lexical economy, facilitate lexicon development and ease the statement of linguistic generalizations.

6.1 Lexical type hierarchy

In this section, we introduce the notion of a *lexical type hierarchy* defined on the basis of complete lattices. The lexical type hierarchy allows us to specify lexical types in a more succinct way and facilitates the statement of linguistic generalizations.

A TDG ID/LP lexicon is a pair $(\mathcal{E}, \mathcal{A})$ consisting of a finite set $\mathcal{E}$ of lexical entries e and a finite set $\mathcal{A}$ of lexical attributes α_i. A lexical entry is a pair of (s, e_t) of a string s from the set *Strs* of strings and a lexical type e_t from the set of lexical types $\mathcal{E}_t$.

Each lexical attribute $\alpha_i \in \mathcal{A} : \mathcal{E}_t \to D_i$ maps lexical types to values in some domain D_i. We arrange the range D_i of lexical attribute α_i in a complete lattice $L_i = \langle D_i, \top_i, \bot_i, \sqcap_i, \sqcup_i \rangle$ where $\top_i$ is the *top element* and $\bot_i$ the *bottom element* of lattice L_i. $\sqcap_i$ denotes the *greatest lower bound*-operation and $\sqcup_i$ the *least upper bound*-operation.

6.1.1 Lexicon lattice

We do not only want to arrange the ranges of lexical attributes in lattices but also the set $\mathcal{E}_t$ of lexical types. We call the complete lattice in which we arrange the lexical types *lexicon lattice* and obtain this lattice by composition of the lattices L_i corresponding to the individual lexical attributes $\alpha_i \in \mathcal{A}$. We define the composition $L_1 \otimes \ldots \otimes L_n$ of lattices $L_i = \langle D_i, \top_i, \bot_i, \sqcap_i, \sqcup_i \rangle$ as follows:

$$\begin{array}{l} L_1 \otimes \ldots \otimes L_n = \\ \langle D_1 \times \ldots \times D_n, \langle \top_1, \ldots, \top_n \rangle, \langle \bot_1, \ldots, \bot_n \rangle, \sqcap_1 \otimes \ldots \otimes \sqcap_n, \sqcup_1 \otimes \ldots \otimes \sqcup_n \rangle \end{array} \tag{6.1}$$

The greatest lower bound- and least upper bound-operations of lattice $L_1 \otimes \ldots \otimes L_n$ are defined below:

$$\langle x_1, \ldots, x_n \rangle \ \sqcap_1 \otimes \ldots \otimes \sqcap_n \ \langle y_1, \ldots, y_n \rangle = \langle x_1 \sqcap_1 y_1, \ldots, x_n \sqcap_n y_n \rangle \tag{6.2}$$

$$\langle x_1, \ldots, x_n \rangle \ \sqcup_1 \otimes \ldots \otimes \sqcup_n \ \langle y_1, \ldots, y_n \rangle = \langle x_1 \sqcup_1 y_1, \ldots, x_n \sqcup_n y_n \rangle \tag{6.3}$$

We define a lexical type $e_t \in \mathcal{E}_t$ as a tuple

$$\langle \alpha_1(e_t), \ldots, \alpha_n(e_t) \rangle \quad \in \quad D_1 \times \ldots \times D_n \tag{6.4}$$

Using AVM-notation, we write

$$[\alpha_1 : \alpha_1(e_t) \ldots \alpha_n : \alpha_n(e_t)] \tag{6.5}$$

instead of $\langle \alpha_1(e_t), \ldots, \alpha_n(e_t) \rangle$.

6.1.2 Lexical inheritance

We can now define our notion of lexical inheritance: we say that a lexical type e directly inherits (from now on, inherits) from lexical types e_i iff $e = e_1 \sqcap \ldots \sqcap e_n$.[1] Lexical inheritance can be seen as a *specialization* of lexical types in order to obtain *subtypes*.

We illustrate lexical inheritance by an example below where lexical type e inherits from the lexical types e_1 and e_2:

$$\begin{array}{l} e = \\ e_1 \sqcap e_2 = \\ [\alpha_1 : \alpha_1(e_1) \ldots \alpha_n : \alpha_n(e_1)] \ \sqcap_1 \otimes \ldots \otimes \sqcap_n \ [\alpha_1 : \alpha_1(e_2) \ldots \alpha_n : \alpha_n(e_2)] = \\ [\alpha_1 : \alpha_1(e_1) \sqcap_1 \alpha_1(e_2) \ldots \alpha_n : \alpha_n(e_1) \sqcap_n \alpha_n(e_2)] \end{array} \tag{6.6}$$

[1] Since the greatest lower bound-lattice operation is associative and commutative, the order of the elements $e_1, \ldots, e_n$ is irrelevant.

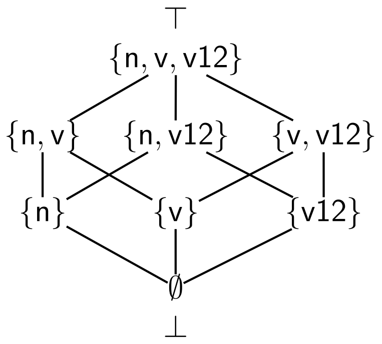
6.2 Lattice types

For the version of TDG proposed in this thesis, we only consider set-valued lexical attributes: $D_i = 2^{S_i}$ where S_i is an arbitrary finite set. We arrange S_i in either an intersective lattice or an accumulative lattice, both of which we introduce below.

6.2.1 Intersective set lattices

An intersective set lattice forms a partial order over the power set 2^S of a set S. The top element is the set S and the bottom element the empty set. The greatest lower bound of two sets is defined by set intersection, and the least upper bound by set union. We write $\mathcal{L}_\cap(S) = \langle 2^S, S, \emptyset, \cap, \cup \rangle$ for the intersective set lattice for S.

As an example, consider the intersective set lattice for $S = \{\mathsf{n}, \mathsf{v}, \mathsf{v12}\}$:



(6.7)

With respect to (6.7), the following equations hold:

$$\{\mathsf{v}\} \sqcup \{\mathsf{v12}\} = \{\mathsf{v}\} \cup \{\mathsf{v12}\} = \{\mathsf{v}, \mathsf{v12}\} \tag{6.8}$$

$$\{\mathsf{n}, \mathsf{v12}\} \sqcup \{\mathsf{v12}\} = \{\mathsf{n}, \mathsf{v12}\} \cup \{\mathsf{v12}\} = \{\mathsf{n}, \mathsf{v12}\} \tag{6.9}$$

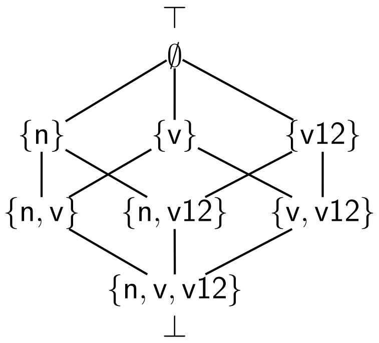
$$\{\mathsf{v}\} \sqcap \{\mathsf{v12}\} = \{\mathsf{v}\} \cap \{\mathsf{v12}\} = \emptyset \tag{6.10}$$

$$\{\mathsf{n}, \mathsf{v12}\} \sqcap \{\mathsf{v12}\} = \{\mathsf{n}, \mathsf{v12}\} \cap \{\mathsf{v12}\} = \{\mathsf{v12}\} \tag{6.11}$$

6.2.2 Accumulative set lattices

An accumulative set lattice also leads to a partial order over the power set 2^S of the set S it is defined on. The top element is the empty set $\emptyset$ and the bottom element is the set S. The greatest lower bound of two sets is defined by set union, and the least upper bound by set intersection. We write $\mathcal{L}_{\cup}(S) = \langle 2^S, \emptyset, S, \cup, \cap \rangle$ for an accumulative set lattice for S.

As an example, here is the accumulative set lattice for $S = \{\mathsf{n}, \mathsf{v}, \mathsf{v12}\}$:



(6.12)

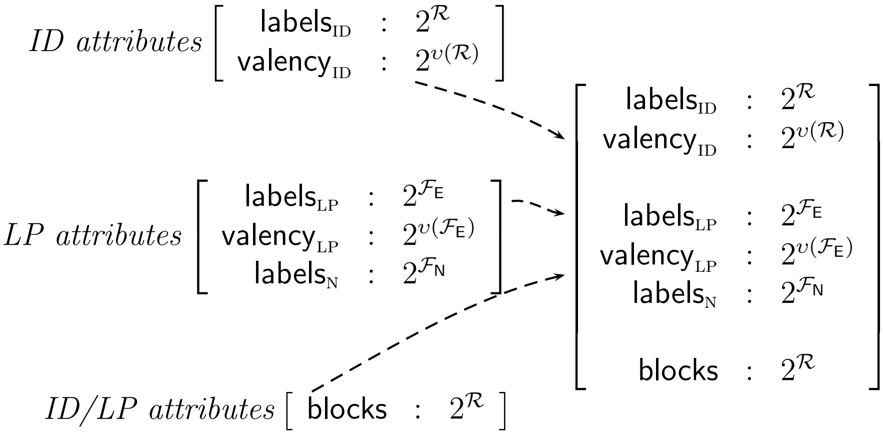
As can easily be seen, (6.12) equals (6.7) but upside down.

6.3 Lexical attributes

In this section, we define the set $\mathcal{A}$ of all lexical attributes α for the version of TDG introduced in this thesis. $\mathcal{A}$ consists of three sets:

- the set $\mathcal{A}_{\text{ID}} = \{\mathsf{labels}_{\text{ID}}, \mathsf{valency}_{\text{ID}}\}$ of ID attributes
- the set $\mathcal{A}_{\text{LP}} = \{\mathsf{labels}_{\text{LP}}, \mathsf{valency}_{\text{LP}}, \mathsf{labels}_{\text{N}}\}$ of LP attributes
- the set $\mathcal{A}_{\text{IDLP}} = \{\mathsf{blocks}\}$ of ID/LP attributes

Thus, $\mathcal{A} = \mathcal{A}_{\text{ID}} \uplus \mathcal{A}_{\text{LP}} \uplus \mathcal{A}_{\text{IDLP}}$, and the signature of a TDG lexical type is as follows:



(6.13)

We arrange the values of the lexical attributes in lattices: we arrange the attributes $\mathsf{labels}_{\mathrm{ID}}$, $\mathsf{labels}_{\mathrm{LP}}$ and $\mathsf{labels}_{\mathrm{N}}$ in intersective set lattices and the attributes $\mathsf{valency}_{\mathrm{ID}}$, $\mathsf{valency}_{\mathrm{LP}}$ and blocks in accumulative set lattices:

$$\left[\begin{array}{rcll} \mathsf{labels}_{\mathrm{ID}} & : & \mathcal{L}_{\cap}(\mathcal{R}) & = \langle 2^{\mathcal{R}}, \mathcal{R}, \emptyset, \cap, \cup \rangle \\ \mathsf{valency}_{\mathrm{ID}} & : & \mathcal{L}_{\cup}(v(\mathcal{R})) & = \langle 2^{v(\mathcal{R})}, \emptyset, v(\mathcal{R}), \cup, \cap \rangle \\ \mathsf{labels}_{\mathrm{LP}} & : & \mathcal{L}_{\cap}(\mathcal{F}_{\mathsf{E}}) & = \langle 2^{\mathcal{F}_{\mathsf{E}}}, \mathcal{F}_{\mathsf{E}}, \emptyset, \cap, \cup \rangle \\ \mathsf{labels}_{\mathrm{N}} & : & \mathcal{L}_{\cap}(\mathcal{F}_{\mathsf{N}}) & = \langle 2^{\mathcal{F}_{\mathsf{N}}}, \mathcal{F}_{\mathsf{N}}, \emptyset, \cap, \cup \rangle \\ \mathsf{valency}_{\mathrm{LP}} & : & \mathcal{L}_{\cup}(v(\mathcal{F}_{\mathsf{E}})) & = \langle 2^{v(\mathcal{F}_{\mathsf{E}})}, \emptyset, v(\mathcal{F}_{\mathsf{E}}), \cup, \cap \rangle \\ \mathsf{blocks} & : & \mathcal{L}_{\cup}(\mathcal{R}) & = \langle 2^{\mathcal{R}}, \emptyset, \mathcal{R}, \cup, \cap \rangle \end{array}\right] \tag{6.14}$$

These choices allow to conveniently build up a lexical type hierarchy. We illustrate this in an example in section 6.4.

For notational convenience, we allow attributes with value $\top$ to be omitted and call these attributes *maximal attributes*. Here are the $\top$-values for the lexical attributes defined above:

attribute	$\top$
$\mathsf{labels}_{\mathrm{ID}}$	$\mathcal{R}$
$\mathsf{valency}_{\mathrm{ID}}$	$\emptyset$
$\mathsf{labels}_{\mathrm{LP}}$	$\mathcal{F}_{\mathsf{E}}$
$\mathsf{labels}_{\mathrm{N}}$	$\mathcal{F}_{\mathsf{N}}$
$\mathsf{valency}_{\mathrm{ID}}$	$\emptyset$
blocks	$\emptyset$

(6.15)

For instance the following lexical type for finite verbs only specifies the ID attributes $\mathsf{labels}_{\mathrm{ID}}$ and $\mathsf{valency}_{\mathrm{ID}}$:

$$t\_fin \; = \; \left[\begin{array}{rcl} \mathsf{labels}_{\mathrm{ID}} & : & \emptyset \\ \mathsf{valency}_{\mathrm{ID}} & : & \{\mathsf{subj}, \mathsf{adv}{*}, \mathsf{pp}?\} \end{array}\right] \tag{6.16}$$

The other attributes, viz. the LP and the ID/LP attributes, are maximal attributes: their value is $\top$ in their respective lattices. Thus, (6.16) amounts to the fully expanded lexical type below:

$$t\_fin \; = \; \left[\begin{array}{rcl} \mathsf{labels}_{\mathrm{ID}} & : & \emptyset \\ \mathsf{valency}_{\mathrm{ID}} & : & \{\mathsf{subj}, \mathsf{adv}{*}, \mathsf{pp}?\} \\ \mathsf{labels}_{\mathrm{LP}} & : & \mathcal{F}_{\mathsf{E}} \\ \mathsf{labels}_{\mathrm{N}} & : & \mathcal{F}_{\mathsf{N}} \\ \mathsf{valency}_{\mathrm{LP}} & : & \emptyset \\ \mathsf{blocks} & : & \emptyset \end{array}\right] \tag{6.17}$$

6.4 Example

Here is an example illustrating how to obtain lexical entries from the lexical type hierarchy. We begin with defining abbreviated lexical types (ID attributes only) for non-finite verbs (*t_nonfin*), bare infinitives (*t_inf*) and ditransitive verbs (*t_ditr*). Non-finite verbs inherit from the following lexical type:

$$t\_nonfin \;=\; \begin{bmatrix} \mathsf{labels}_{\mathrm{ID}} & : & \{\mathsf{vinf}, \mathsf{vzu}, \mathsf{vpp}\} \\ \mathsf{valency}_{\mathrm{ID}} & : & \{\mathsf{adv}{*}, \mathsf{pp}?\} \\ & \ldots & \end{bmatrix} \tag{6.18}$$

The set of accepted roles of (6.18) includes vinf (bare infinitive), vzu (*zu*-infinitive) and vpp (past participle). The role valency includes an arbitrary number of adjectives and zero or one PP-modifiers.

Bare infinitives inherit from the lexical type given below:

$$t\_inf \;=\; \begin{bmatrix} \mathsf{labels}_{\mathrm{ID}} & : & \{\mathsf{vinf}\} \\ & \ldots & \end{bmatrix} \tag{6.19}$$

The only accepted role is vinf. We omit the maximal attribute $\mathsf{valency}_{\mathrm{ID}}$ which is therefore assigned the $\top$-value (here: $\top = \emptyset$).

Ditransitive verbs inherit from the following lexical type:

$$t\_ditr \;=\; \begin{bmatrix} \mathsf{valency}_{\mathrm{ID}} & : & \{\mathsf{obj}, \mathsf{iobj}\} \\ & \ldots & \end{bmatrix} \tag{6.20}$$

The role valency includes an object (obj) and an indirect object (iobj). We omit the attribute $\mathsf{labels}_{\mathrm{ID}}$ which is therefore assigned the $\top$-value (here: $\top = \mathcal{R}$).

By lexical inheritance, we can now obtain the lexical entry for the non-finite bare infinitival ditransitive verb *geben* as follows:

$$(geben \;\;,\;\; t\_nonfin \sqcap t\_inf \sqcap t\_ditr) \tag{6.21}$$

(6.21) results in the following lexical entry:

$$(geben \;\;,\;\; \begin{bmatrix} \mathsf{labels}_{\mathrm{ID}} & : & \{\mathsf{vinf}\} \\ \mathsf{valency}_{\mathrm{ID}} & : & \{\mathsf{adv}{*}, \mathsf{pp}?, \mathsf{obj}, \mathsf{iobj}\} \\ & \ldots & \end{bmatrix}) \tag{6.22}$$

We arrange the lexical attributes denoting accepted roles ($\mathsf{labels}_{\mathrm{ID}}$), accepted fields ($\mathsf{labels}_{\mathrm{LP}}$) and accepted node labels ($\mathsf{labels}_{\mathrm{N}}$) in intersective set lattices to allow specialization by narrowing down sets. In the example, we specialized the set of accepted roles from $\{\mathsf{vinf}, \mathsf{vzu}, \mathsf{vpp}\}$ in lexical type *t_nonfin* (6.18) to $\{\mathsf{vinf}\}$ in (6.22):

$$\begin{array}{lcl} \mathsf{labels}_{\mathrm{ID}}(\mathit{geben}) & = & \mathsf{labels}_{\mathrm{ID}}(\mathit{t\_nonfin}) \sqcap \mathsf{labels}_{\mathrm{ID}}(\mathit{t\_inf}) \sqcap \mathsf{labels}_{\mathrm{ID}}(\mathit{t\_ditr}) = \\ & & \{\mathsf{vinf}, \mathsf{vzu}, \mathsf{vpp}\} \sqcap \{\mathsf{vinf}\} \sqcap \mathcal{R} = \\ & & \{\mathsf{vinf}, \mathsf{vzu}, \mathsf{vpp}\} \cap \{\mathsf{vinf}\} \cap \mathcal{R} = \\ & & \{\mathsf{vinf}\} \end{array} \tag{6.23}$$

The lexical attributes denoting role valency ($\mathsf{valency}_{\mathrm{ID}}$), field valency ($\mathsf{valency}_{\mathrm{LP}}$) and blocked roles ($\mathsf{blocks}$) are arranged in accumulative set lattices to allow specialization by adding elements: in the example, we specialized the role valency from $\{\mathsf{adv}{*}, \mathsf{pp}?\}$ in lexical type *t_nonfin* (6.18) to the set $\{\mathsf{adv}{*}, \mathsf{pp}?, \mathsf{obj}, \mathsf{iobj}\}$ in (6.22):

$$\begin{array}{ll} \mathsf{valency}_{\mathrm{ID}}(\mathit{geben}) = & \mathsf{valency}_{\mathrm{ID}}(\mathit{t\_nonfin}) \sqcap \mathsf{valency}_{\mathrm{ID}}(\mathit{t\_inf}) \sqcap \mathsf{valency}_{\mathrm{ID}}(\mathit{t\_ditr}) = \\ & \{\mathsf{adv}{*}, \mathsf{pp}?\} \sqcap \emptyset \sqcap \{\mathsf{obj}, \mathsf{iobj}\} = \\ & \{\mathsf{adv}{*}, \mathsf{pp}?\} \cup \emptyset \cup \{\mathsf{obj}, \mathsf{iobj}\} = \\ & \{\mathsf{adv}{*}, \mathsf{pp}?, \mathsf{obj}, \mathsf{iobj}\} \end{array} \tag{6.24}$$

6.5 Summary

This chapter introduced the TDG lexicon. The set of lexical types is arranged in a lexical type hierarchy called lexicon lattice which is modeled by a complete lattice. The lexicon lattice is the composition of the lattices corresponding to the individual lexical attributes. We define lexical inheritance with respect to the lexicon lattice using the greatest lower bound-lattice operation.

The next chapter presents a TDG grammar fragment of German. We will use this fragment in chapter 8 to tackle a number of notorious phenomena in German syntax.

Chapter 7

German grammar fragment

We present a TDG grammar fragment for German which will enable us to elegantly tackle a variety of notorious syntactic phenomena. Still, the fragment is only intended to be a demonstration of TDG, and is not meant to be a definitive grammar for German. For clarity, we introduce grammar specifications concerned with the ID part separately from those concerned with the LP part.

7.1 ID part

In this section, we introduce grammar specifications concerned with the ID part of TDG. These specifications include the set $\mathcal{R}$ of grammatical roles and the definition of a set of lexical types defined on the basis of ID attributes.

7.1.1 Definitions

Grammatical roles

The set $\mathcal{R}$ of grammatical roles for the grammar fragment is given in Table 7.1.

7.1.2 Subcategorization

For verbs in general, we define a set of lexical types dealing with subcategorization. For instance a transitive verb is characterized by requiring an object, and a ditransitive by requiring both an object and and indirect object. As an example,

role	full name
adj	adjective
adv	adverb
comp	complementizer
det	determiner
genmod	genitival modifier
genobj	genitival object
iobj	dative indirect object
obj	accusative object
pp	prepositional phrase
rel	relative clause
sub	subordinate clause
subj	nominative subject
vinf	bare infinitive
vpp	past participle
vzu	zu-infinitive
zu	zu-particle

Table 7.1: The set of grammatical roles for the grammar fragment

we show the lexical type *t_tr_id* for transitive verbs below:

$$t\_tr\_id \quad = \quad \left[\begin{array}{lcl} \mathsf{valency}_{\mathrm{ID}} & : & \{\mathsf{obj}\} \end{array} \right] \tag{7.1}$$

Another example is the lexical type *t_ditr_id* for ditransitive verbs:

$$t\_ditr\_id \quad = \quad \left[\begin{array}{lcl} \mathsf{valency}_{\mathrm{ID}} & : & \{\mathsf{obj}, \mathsf{iobj}\} \end{array} \right] \tag{7.2}$$

And verbs subcategorizing for a bare infinitive inherit from the following lexical type:

$$t\_infc\_id \quad = \quad \left[\begin{array}{lcl} \mathsf{valency}_{\mathrm{ID}} & : & \{\mathsf{vinf}\} \end{array} \right] \tag{7.3}$$

7.1.3 Finite verbs

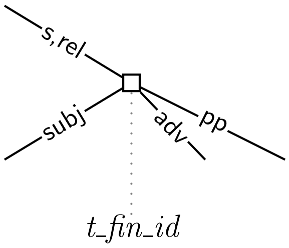
The ID part of the lexical type *t_fin_id* for finite verbs looks like this:

$$t\_fin\_id \quad = \quad \left[\begin{array}{rcl} \mathsf{labels}_{\mathrm{ID}} & : & \{\mathsf{sub}, \mathsf{rel}\} \\ \mathsf{valency}_{\mathrm{ID}} & : & \{\mathsf{subj}, \mathsf{adv}*, \mathsf{pp}?\} \end{array} \right] \tag{7.4}$$

The set of accepted roles in (7.4) includes sub and rel, i.e. a finite verb can either be root, the head of a relative clause (rel) or the head of a subordinate clause (sub).

The role valency consists of a required subject, an arbitrary number of adverbial modifiers (adv) and at most one prepositional (pp) modifier.

Here is a graphical illustration of (7.4), illustrating the possible incoming and outgoing edges[1] of a node with type *t_fin_id*:



(7.5)

Note that since ID trees are non-ordered, the order of the edges in these figures is arbitrary.

By inheritance, the role valency of finite verbs can be augmented with other complements such as obj and iobj for a ditransitive verb. E.g. the lexical type *t_fin_ditr_id* for finite ditransitive verbs can be obtained as displayed here:

$$t\_fin\_ditr\_id \quad = \quad t\_fin\_id \sqcap t\_ditr\_id \tag{7.6}$$

resulting in the following expanded lexical type:

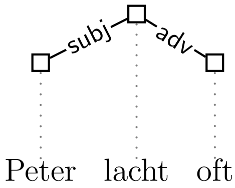
$$t\_fin\_ditr\_id \quad = \quad \begin{bmatrix} \mathsf{labels}_{\mathrm{ID}} & : & \{\mathsf{sub}, \mathsf{rel}\} \\ \mathsf{valency}_{\mathrm{ID}} & : & \{\mathsf{subj}, \mathsf{adv}{*}, \mathsf{pp}?, \mathsf{obj}, \mathsf{iobj}\} \end{bmatrix} \tag{7.7}$$

Examples

Consider the following sentence:

Peter lacht oft.
Peter laughs often. (7.8)
"Peter often laughs."

We show the corresponding ID tree below:



(7.9)

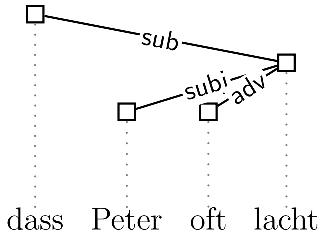
[1] In the illustrations, the number of the outgoing edges does not accurately reflect the lexical type since we do not graphically distinguish optional and obligatory dependents.

Here, the finite verb *lacht* is the root. *Peter* is the subject of *lacht* and *oft* an adverbial modifier.

The ID tree shown in (7.11) corresponds to sentence (7.10):

dass	Peter	oft	lacht	
that	Peter	often	laughs	(7.10)

"that Peter often laughs"



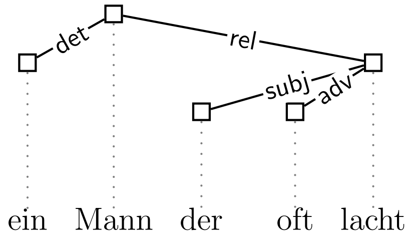
(7.11)

Here, *lacht* is the head of a subordinate clause, as indicated by its role sub. Again, *Peter* is the subject of *lacht* and *oft* an adverbial modifier.

In the ID tree for (7.12), *lacht* is the head of a relative clause having grammatical role rel:

ein	Mann,	der	oft	lacht	
a	man,	who	often	laughs	(7.12)

"a man who often laughs"



(7.13)

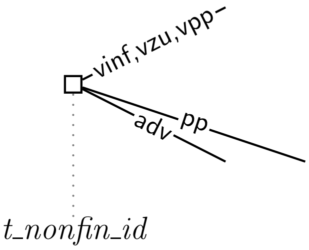
Here, *lacht* heads a relative clause which modified the noun *Mann*.

7.1.4 Non-finite verbs

The grammatical role of a non-finite verb (lexical type: *t_nonfin_id*) is either vinf (bare infinitive), vzu (*zu*-infinitive) or vpp (past participle). As opposed to finite verbs, non-finite verbs never have a subject:

$$t\_nonfin\_id = \begin{bmatrix} \mathsf{labels}_{\mathrm{ID}} & : & \{\mathsf{vinf}, \mathsf{vzu}, \mathsf{vpp}\} \\ \mathsf{valency}_{\mathrm{ID}} & : & \{\mathsf{adv}*, \mathsf{pp}?\} \end{bmatrix} \tag{7.14}$$

Here is a graphical illustration of (7.14):



(7.15)

We use lexical inheritance to obtain lexical types for bare infinitives (vinf), *zu*-infinitives (vzu) and past participles (vpp):

$$t\_vinf\_id \quad = \quad t\_nonfin\_id \sqcap \left[\ \mathsf{labels}_{\mathrm{ID}} \ : \ \{\mathsf{vinf}\} \ \right] \tag{7.16}$$

$$t\_vzu\_id \quad = \quad t\_nonfin\_id \sqcap \left[\begin{array}{rcl} \mathsf{labels}_{\mathrm{ID}} & : & \{\mathsf{vzu}\} \\ \mathsf{valency}_{\mathrm{ID}} & : & \{\mathsf{zu}\} \end{array} \right] \tag{7.17}$$

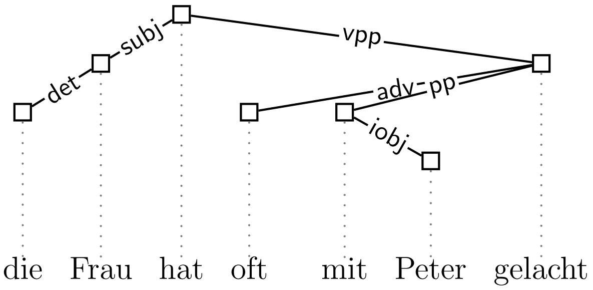
$$t\_vpp\_id \quad = \quad t\_nonfin\_id \sqcap \left[\ \mathsf{labels}_{\mathrm{ID}} \ : \ \{\mathsf{vpp}\} \ \right] \tag{7.18}$$

Example

Consider the following example sentence:

Die Frau hat oft mit Peter gelacht
The woman has often with peter laughed (7.19)
"The woman has often laughed with Peter."

The ID tree analysis of (7.19) is shown below:



(7.20)

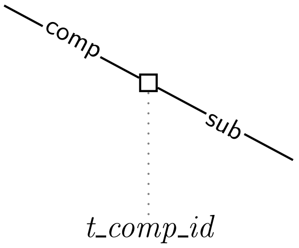
Here, the past participle *gelacht* is a vpp-dependent of the root *hat*. *gelacht* is modified by the adverb *oft* and by the prepositional phrase *mit Peter*.

7.1.5 Complementizers

Complementizers like *dass* inherit from lexical type *t_comp_id*:

$$t\_comp\_id \;=\; \begin{bmatrix} \mathsf{labels}_{\text{ID}} & : & \{\mathsf{comp}\} \\ \mathsf{valency}_{\text{ID}} & : & \{\mathsf{sub}\} \end{bmatrix} \tag{7.21}$$

Here is the graphical version of that lexical type:

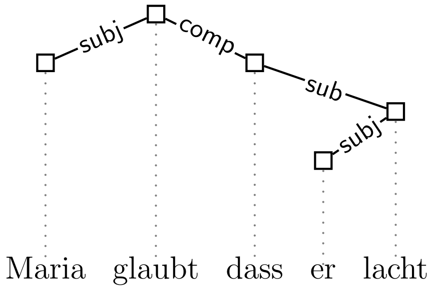


(7.22)

Example

An example sentence including the complementizer *dass* and its corresponding ID tree analysis are given below:

Maria glaubt, dass er lacht.
Maria thinks, that he laughs. (7.23)
"Maria thinks that he laughs."



(7.24)

Here, *dass* is a comp-dependent of the finite verb *glaubt.* The subordinate clause headed by *lacht* is a sub-dependent of *dass.*

7.1.6 Adverbs

Adverbs inherit from *t_adv_id*:

$$t\_adv\_id \;=\; \begin{bmatrix} \mathsf{labels}_{\text{ID}} & : & \{\mathsf{adv}\} \end{bmatrix} \tag{7.25}$$

7.1.7 Zu-particle

The *zu*-particle inherits from lexical type *t_zu_id*:

$$t\_zu\_id \;=\; \left[\;\textsf{labels}_{\text{ID}} \;:\; \{\textsf{zu}\}\;\right] \tag{7.26}$$

7.1.8 Nouns

All nouns inherit from the following lexical type:

$$t\_noun\_id \;=\; \left[\;\textsf{labels}_{\text{ID}} \;:\; \{\textsf{subj}, \textsf{obj}, \textsf{iobj}, \textsf{genobj}, \textsf{genmod}\}\;\right] \tag{7.27}$$

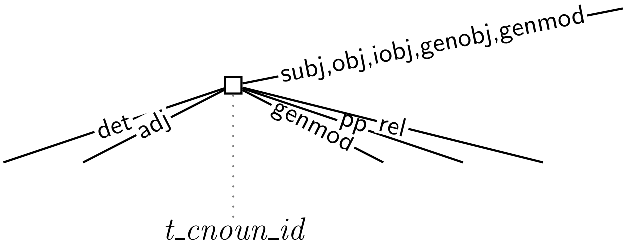
The role of the noun depends on its case: e.g. if the noun is in accusative case, $\textsf{labels}_{\text{ID}}$ contains only role $\textsf{obj}$.[2]

Common nouns

Common nouns inherit from the following lexical type:

$$t\_cnoun\_id \;=\; t\_noun\_id \sqcap \left[\;\textsf{valency}_{\text{ID}} \;:\; \{\textsf{det}?, \textsf{adj}{*}, \textsf{genmod}?, \textsf{pp}?, \textsf{rel}?\}\;\right] \tag{7.28}$$

Below is a graphical illustration of (7.28):



(7.29)

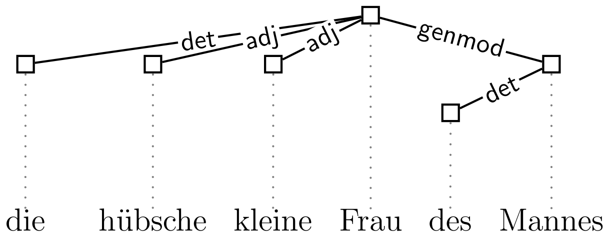
Examples

Here are two examples for common nouns. An ID analysis of the NP glossed below is depicted in (7.31):

die	hübsche	kleine	Frau	des	Mannes
the	pretty	small	woman	the	man(gen)

"the man's pretty small woman" (7.30)

[2] For simplicity, we have not included a treatment of agreement and case in the grammar fragment. We illustrate how to handle this in appendix A.

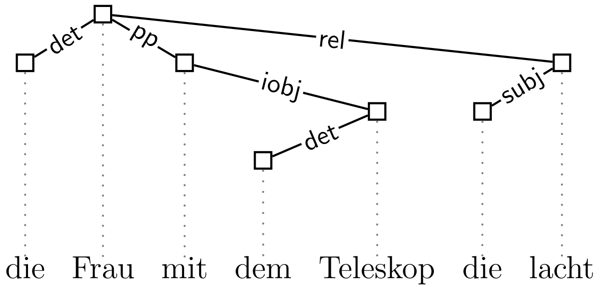


(7.31)

In (7.31), *die* is the determiner of the noun *Frau*, and *hübsche* and *kleine* are adjectival modifiers. The noun phrase *des Mannes* is a genitival modifier of *Frau*.

The ID tree analysis of the sentence glossed in (7.32) is shown in (7.33):

die Frau mit dem Teleskop, die lacht
the woman with the telescope, who laughs (7.32)
"the woman with the telescope who laughs"



(7.33)

In (7.33), the noun *Frau* is modified by the PP *mit dem Teleskop* and the relative clause *die lacht*.

Proper names

Proper names inherit from the following lexical type:

$$t\_pname\_id \quad = \quad t\_noun\_id \sqcap \left[\; \mathsf{valency}_{\text{ID}} \quad : \quad \{\mathsf{det}?, \mathsf{adj}{*}, \mathsf{genmod}?, \mathsf{pp}?, \mathsf{rel}?\} \; \right] \tag{7.34}$$

Personal pronouns

Personal pronouns inherit from the following lexical type:

$$t\_perpro\_id \quad = \quad t\_noun\_id \tag{7.35}$$

Relative pronouns

Relative pronouns inherit from the following lexical type:

$$t\_relpro\_id \quad = \quad t\_noun\_id \tag{7.36}$$

7.1.9 Determiners

Determiners inherit from lexical type *t_det_id*:

$$t\_det\_id \quad = \quad \left[\begin{array}{lll} \mathsf{labels}_{\mathrm{ID}} & : & \{\mathsf{det}\} \end{array} \right] \tag{7.37}$$

7.1.10 Adjectives

Adjectives inherit from *t_adj_id*:

$$t\_adj\_id \quad = \quad \left[\begin{array}{lll} \mathsf{labels}_{\mathrm{ID}} & : & \{\mathsf{adj}\} \end{array} \right] \tag{7.38}$$

7.1.11 Prepositions

Prepositions inherit from lexical type *t_prep_id*:

$$t\_prep\_id \quad = \quad \left[\begin{array}{lll} \mathsf{labels}_{\mathrm{ID}} & : & \{\mathsf{pp}\} \end{array} \right] \tag{7.39}$$

The role valency of a preposition contains, depending on the type of the preposition, either obj (if the preposition combines with a accusative complement) or iobj (dative complement). This is reflected in the role valency of the two lexical subtypes *t_prep_obj_id* and *t_prep_iobj_id*:[3]

$$t\_prep\_obj\_id \quad = \quad t\_prep\_id \sqcap \left[\begin{array}{lll} \mathsf{valency}_{\mathrm{ID}} & : & \{\mathsf{obj}\} \end{array} \right] \tag{7.40}$$

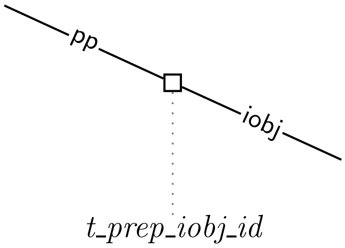
$$t\_prep\_iobj\_id \quad = \quad t\_prep\_id \sqcap \left[\begin{array}{lll} \mathsf{valency}_{\mathrm{ID}} & : & \{\mathsf{iobj}\} \end{array} \right] \tag{7.41}$$

For example, the resulting lexical type *t_prep_iobj_id* looks like this:

$$t\_prep\_iobj\_id \quad = \quad \left[\begin{array}{lll} \mathsf{labels}_{\mathrm{ID}} & : & \{\mathsf{pp}\} \\ \mathsf{valency}_{\mathrm{ID}} & : & \{\mathsf{iobj}\} \end{array} \right] \tag{7.42}$$

[3] It is also possible to distinguish between the two types of prepositions using case. We opt for the simpler way of distinguishing them here.

As an illustration, we show a graphical version of (7.42) below:



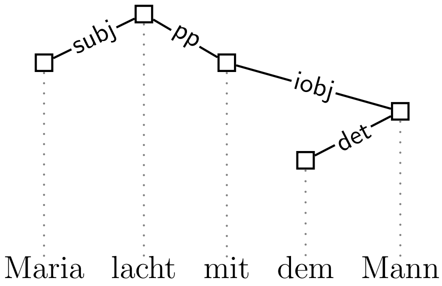
(7.43)

Example

Here is an example sentence including the prepositional phrase *mit dem Mann* and its corresponding ID analysis:

Maria	lacht	mit	dem	Mann.
Maria	laughs	with	the	man.

"Maria laughs with the man." (7.44)



(7.45)

Here, *mit* is a pp-dependent of *lacht*. In turn, *Mann* is a iobj-dependent of *mit*.

7.2 LP part

We proceed with the LP part of the grammar fragment. Thereby, we introduce the set $\mathcal{F}_{\mathsf{E}}$ of topological fields, the set $\mathcal{F}_{\mathsf{N}}$ of node labels and a total order on the set $\mathcal{F} = \mathcal{F}_{\mathsf{E}} \uplus \mathcal{F}_{\mathsf{N}}$ of topological fields and node labels. Then, we introduce a set of lexical types defined on the basis of LP attributes and also the ID/LP attribute blocks.

7.2.1 Definitions

Topological fields

The set $\mathcal{F}_{\mathsf{E}}$ of topological fields defined for the grammar fragment is depicted in Table 7.2.

field	full name
if	intraposition field
mf	Mittelfeld
nf	Nachfeld
nmf	nominal Mittelfeld
nvf	nominal Vorfeld
nxf	nominal extraposition field
pf	particle field
rvf	relative clause Vorfeld
vcf	verb canonical field
vf	Vorfeld
vxf	verb extraposition field

Table 7.2: The set of topological fields for the grammar fragment

The fields vf, mf, vcf and nf correspond to Vorfeld, Mittelfeld, right sentence bracket and Nachfeld in topological fields theory. vcf abbreviates 'verb canonical field': it is not only the counterpart of the right sentence bracket-position in topological fields theory but also, more generally, the landing site for verbs in *canonical position*, i.e. in the typical, non-marked position. The 'particle field' pf is the landing site for the *zu*-particle.

The 'intraposition field' if is the landing site for VPs intraposed to the left of the Mittelfeld. The 'relative clause Vorfeld' rvf is the position of the relative pronoun in relative clauses to the left of the Mittelfeld (also to the left of the intraposition field).[4] rvf is also the landing site for NPs, PPs or VPs participating in a *pied piping construction* (see also the discussion about pied piping in chapter 8 and appendix A).

The 'verb extraposition field' vxf is a position for VPs extraposed to the right of their governors, but to the left of the Nachfeld. Postulating the vxf goes against many theories (e.g. Kathol 1995) adopting topological fields theory where extraposed VPs land in the Nachfeld. But if extraposed VPs landed in the same field as other

[4] Appendix A contains a more in-depth discussion about relative clauses and especially the relative pronoun and the rvf-field.

extraposed material such as extraposed relative clauses, then extraposed VPs and extraposed relative clauses in the Nachfeld should not be ordered with respect to each other. However, the following contrast demonstrates that they *are* ordered: extraposed VPs must precede extraposed relative clauses:

	(dass)	eine	Frau	versucht,	zu	schlafen,	die	lacht
	(that)	a	woman	tries,	to	sleep,	who	laughs

"(that) a woman who laughs tries to sleep" (7.46)

*	(dass)	eine	Frau	versucht,	die	lacht,	zu	schlafen
	(that)	a	woman	tries,	who	laughs,	to	sleep

(7.47)

In (7.46), *zu schlafen* is the extraposed VP and *die lacht* the extraposed relative clause. As can be seen from the ungrammaticality of (7.47), *zu schlafen* must precede *die lacht*. We capture this ordering by assigning different fields to extraposed VPs and extraposed relative clauses: extraposed VPs land in the vxf and extraposed relative clauses in the nf, and vxf precedes nf.

We coin the fields 'nominal Vorfeld' (nvf) and 'nominal Mittelfeld' (nmf) to hint at the similarity between the topological structures induced by nouns and verbs. nvf is the position of the determiner to the very left of the governing noun. We liken this position to the Vorfeld. nmf is the position for adjectives between the determiner and the noun. Since a noun can be modified by any number of adjectives in any order[5], we liken the adjective position to the Mittelfeld of a verb: here, non-verbal complements can also be rather freely permuted. Finally, the 'nominal extraposition field' (nxf) is the field for a genitival modifier or a PP following the noun. It is the counterpart of the verb extraposition field vxf.

Node labels

The set $\mathcal{F}_{\mathsf{N}}$ of node labels is shown here:

node label	full name
n	words related to nouns
v12	finite verbs (left sentence bracket)
v	words related to verbs

(7.48)

We assign node label n to all nouns and to all words related to nouns such as determiners and adjectives. Node label v12 is assigned to finite verbs in the left sentence bracket in verb-first and verb-second sentences. The '1' in v12 stands for verb-first and the '2' for verb-second. Node label v applies to all other verbs and also to words related to verbs such as the *zu*-particle.

[5] Notice that the order of the adjectives modifying a noun tends to have semantic implications.

Global order

The set $\mathcal{F} = \mathcal{F}_{\mathsf{E}} \uplus \mathcal{F}_{\mathsf{N}}$ of topological fields and node labels is totally ordered. The total order utilized for the grammar fragment is displayed below:

$$\mathsf{nvf} \prec \mathsf{nmf} \prec \mathsf{n} \prec \mathsf{nxf} \prec \mathsf{vf} \prec \mathsf{v12} \prec \mathsf{rvf} \prec \mathsf{if} \prec \mathsf{mf} \prec \mathsf{vcf} \prec \mathsf{pf} \prec \mathsf{v} \prec \mathsf{vxf} \prec \mathsf{nf} \tag{7.49}$$

In the course of this section, we will subdivide the total order shown in (7.49) into so-called *topological domains* induced by lexical types. As an example, we show the lexical type *t_cnoun_lp* for common nouns, which we anticipate in (7.50) below (omitting the $\mathsf{labels}_{\mathrm{LP}}$-attribute):

$$t\_cnoun\_lp \ = \ \left[\begin{array}{ccl} \mathsf{labels}_{\mathrm{N}} & : & \{\mathsf{n}\} \\ \mathsf{valency}_{\mathrm{LP}} & : & \{\mathsf{nvf}, \mathsf{nmf}{*}, \mathsf{nxf}?, \mathsf{nf}?\} \\ & \ldots & \end{array}\right] \tag{7.50}$$

Nouns are assigned the node label n and they offer the fields nvf, nmf, nxf and nf. Considering the total order in (7.49), *t_cnoun_lp* induces the following topological domain:

$$\mathsf{nvf} \prec \mathsf{nmf} \prec \mathsf{n} \prec \mathsf{nxf} \prec \mathsf{nf} \tag{7.51}$$

where the noun (node label n) follows the fields nvf (for determiners and genitival modifiers) and nmf (adjectives), and precedes nxf (PPs and genitival modifiers) and nf (relative clauses).

7.2.2 Finite verbs

In the LP part of the grammar fragment, finite verbs inherit from lexical type *t_fin_lp*:

$$t\_fin\_lp \ = \ \left[\begin{array}{ccl} \mathsf{labels}_{\mathrm{N}} & : & \{\mathsf{v}, \mathsf{v12}\} \\ \mathsf{valency}_{\mathrm{LP}} & : & \{\mathsf{if}?, \mathsf{mf}{*}, \mathsf{vcf}?, \mathsf{vxf}?, \mathsf{nf}?\} \end{array}\right] \tag{7.52}$$

The node label of a finite verb is either v (in verb-final position) or v12 (in left sentence bracket position). Each finite verb offers the positions intraposition field (if), Mittelfeld (mf), right sentence bracket (vcf), verb extraposition field (vxf) and Nachfeld (nf). We do not allow climbing through finite verb nodes: all roles are blocked.

We distinguish three types of finite verbs, depending on their position: (a) verbs in left sentence bracket (verb-first or verb-second) position, (b) verbs in verb-final subordinate clause position and (c) verbs in verb-final relative clause position. We obtain the corresponding lexical types *t_v12_lp*, *t_sub_lp* and *t_rel_lp* by lexical inheritance.

Verb-first and verb-second

Finite verbs in the left sentence bracket inherit from lexical type *t_v12_lp*:

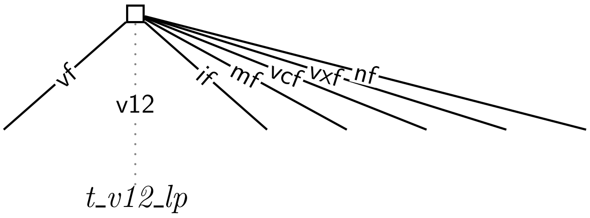
$$t\_v12\_lp \quad = \quad t\_fin\_lp \sqcap \begin{bmatrix} \mathsf{labels}_{\mathrm{ID}} & : & \emptyset \\ \mathsf{labels}_{\mathrm{LP}} & : & \emptyset \\ \mathsf{labels}_{\mathrm{N}} & : & \{\mathsf{v12}\} \\ \mathsf{valency}_{\mathrm{LP}} & : & \{\mathsf{vf?}\} \end{bmatrix} \tag{7.53}$$

t_v12_lp specializes lexical type *t_fin_lp*. It also fixes the value of the ID attribute $\mathsf{labels}_{\mathrm{ID}}$ to the empty set. The set of accepted fields is also empty[6] and its node label is v12. The field valency of a finite verb in verb-first or verb-second position includes besides the fields inherited from *t_fin_lp* also the optional Vorfeld-position (vf?). The Vorfeld in a verb-first sentence is empty and the Vorfeld is non-empty in a verb-second sentence.

The topological domain induced by the lexical type *t_v12_lp* and the total order displayed in (7.49) is shown here:

$$\mathsf{vf} \prec \mathsf{v12} \prec \mathsf{if} \prec \mathsf{mf} \prec \mathsf{vcf} \prec \mathsf{vxf} \prec \mathsf{nf} \tag{7.54}$$

Below, we show a graphical illustration of *t_v12_lp*:



(7.55)

Notice that contrary to the graphical illustrations we display in the ID part, the order of the edges is crucial in illustrations of lexical types concerned with the LP part.

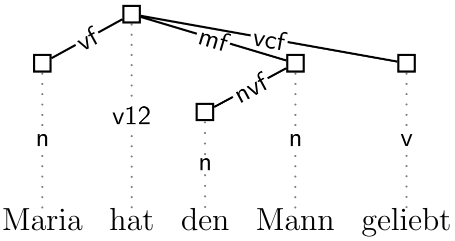
Examples

We continue with some examples. The first is a verb-second sentence:

Maria	hat	den	Mann	geliebt.	
Maria	has	the	man(acc)	loved.	(7.56)

"Maria has loved the man."

[6] If the sets of accepted roles and accepted fields are empty, then verbs in this position must always be the root. But sentences like *Er sagt, er sei der beste.* show that this is not the case: here, *sei* is in verb-second position but not the root of the sentence. For simplicity, we do not treat this phenomenon here.



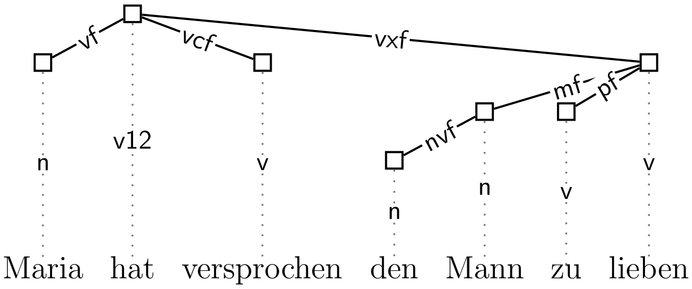
(7.57)

where *Maria* in the Vorfeld (vf) correctly precedes the finite verb *hat* with node label v12. *hat* in turn precedes the NP *den Mann* in the Mittelfeld (mf) and the past participle *geliebt* in the right sentence bracket (vcf).

In the example below, the past participle *versprochen* lands in the right sentence bracket position and the VP *den Mann zu lieben* is extraposed into the vxf:

Maria	hat	versprochen,	den	Mann	zu	lieben.
Maria	has	promised,	the	man(acc)	to	love.

"Maria has promised to love the man." (7.58)



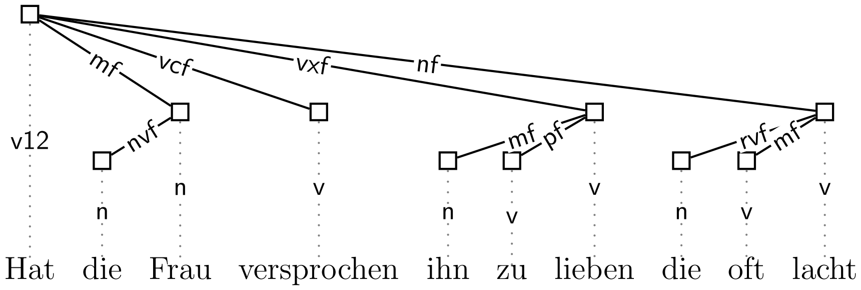
(7.59)

The next example is a verb-first sentence (the vf is empty). Here, the fields vcf, vxf and nf of the finite verb *hat* are all used up: the right sentence bracket (vcf) is filled by the past participle *versprochen*, the verb extraposition field (vxf) by the extraposed VP *ihn zu lieben* and the Nachfeld (nf) by the relative clause *die oft lacht*:

Hat	die	Frau	versprochen,	ihn	zu	lieben,	die	oft	lacht?
Has	the	woman	promised,	him	to	love,	who	often	laughs?

"Has the woman who often laughs promised to love him?"

(7.60)



(7.61)

Verb-final subordinate clause

Finite verbs in verb-final position inherit from the following lexical type:

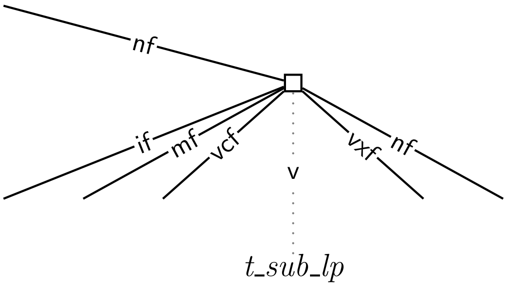
$$t\_sub\_lp \quad = \quad t\_fin\_lp \sqcap \begin{bmatrix} \mathsf{labels}_{\mathrm{ID}} & : & \{\mathsf{sub}\} \\ \mathsf{labels}_{\mathrm{LP}} & : & \{\mathsf{nf}\} \\ \mathsf{labels}_{\mathrm{N}} & : & \{\mathsf{v}\} \end{bmatrix} \tag{7.62}$$

Verbs of this type have role sub, land only in the Nachfeld nf and have node label v. Their field valency equals the field valency inherited from *t_fin_lp*.

Here is the topological domain induced by the lexical type in (7.62) and the total order in (7.49):

$$\mathsf{if} \prec \mathsf{mf} \prec \mathsf{vcf} \prec \mathsf{v} \prec \mathsf{vxf} \prec \mathsf{nf} \tag{7.63}$$

And below, we depict the corresponding graphical illustration:



(7.64)

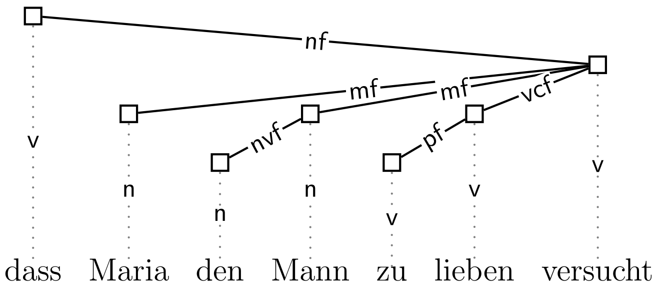
Example

Here is an example LP analysis of a subordinate clause:

dass	Maria	den	Mann	zu	lieben	versucht
that	Maria	the	man(acc)	to	love	tries

"that Maria tries to love the man" (7.65)

Inducing the following ID tree analysis:



(7.66)

Here, the finite verb *versucht* lands in the nf of the complementizer *dass*. The two NPs *Maria* and *den Mann* land in the mf of *versucht*, and the verbal complement *zu lieben* in the vcf.

Verb-final relative clause

Finite verbs heading a relative clause inherit from lexical type *t_rel_lp*:

$$t\_rel\_lp \quad = \quad t\_fin\_lp \sqcap \begin{bmatrix} \mathsf{labels}_{\mathrm{ID}} & : & \{\mathsf{rel}\} \\ \mathsf{labels}_{\mathrm{LP}} & : & \{\mathsf{nf}\} \\ \mathsf{labels}_{\mathrm{N}} & : & \{\mathsf{v}\} \\ \mathsf{valency}_{\mathrm{LP}} & : & \{\mathsf{rvf}\} \end{bmatrix} \tag{7.67}$$

Verbs of this type have role rel. They can only land in the Nachfeld (nf), accept node label v and offer the relative clause Vorfeld (rvf). rvf is the landing site of the relative pronoun and also pied piped material such as PPs and VPs.

The attentive reader might ask the question why we use the new field 'relative clause Vorfeld' (rvf) and why we do not let relative pronouns (and pied piped NPs, PPs and VPs) land in the Vorfeld (vf)? The answer is that if we indeed applied vf for the analysis of relative clauses, we would overgenerate with respect to VP pied piping. The set of constituents which may participate in a VP pied piping construction (landing in the field rvf) is much smaller than the set of constituents which may be fronted (landing in the field vf). E.g. VPs of any type may be fronted, including VPs headed by bare infinitives, past participles or *zu*-infinitives. But only the latter can also occur in a pied piping construction.

Here are some examples. In (7.68) and (7.69) below, the VP *den Mann zu lieben* is fronted into the Vorfeld (7.68) and takes part in a pied piping construction (7.69). Both sentences are grammatical because the VP is headed by a *zu*-infinitive:

Den	Mann	zu	lieben	verspricht	Maria.	
The	man(acc)	to	love	promises	Maria.	(7.68)

"Maria promises to love the man."

ein	Mann,	den	zu	lieben	Maria	verspricht	
a	man,	whom	to	love	Maria	promises	(7.69)

"a man whom Maria promises to love"

(7.70) and (7.71) contain the VP *den Mann geliebt* which is headed by the past participle *geliebt*. VPs of this kind may also fronted (7.70), but they cannot participate in a pied piping construction, as demonstrated by the ungrammaticality of (7.71):

Den	Mann	geliebt	hat	Maria.	
The	man(acc)	loved	has	Maria.	(7.70)

"Maria has loved the man."

*	ein	Mann,	den	geliebt	Maria	hat	(7.71)
	a	man,	whom	loved	Maria	has	

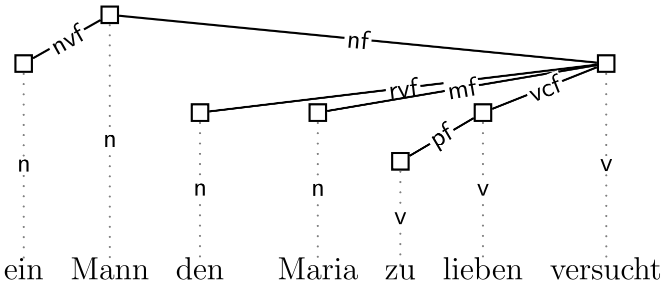
Since past participles such as *geliebt can* be fronted, they must include vf in their set of accepted fields. If we did not distinguish between vf and rvf, i.e. if we postulated that relative pronouns and pied piped material both landed in the vf, then *geliebt* could also participate in pied piping constructions. But this is obviously not possible, as (7.71) demonstrates. We account for this distinction by including rvf in the set of accepted fields of *zu*-infinitives and by *not* including it in the set of accepted fields of past participles and bare infinitives.

Example

Here is an example of a finite verb heading a relative clause:

ein	Mann,	den	Maria	zu	lieben	versucht	
a	man,	whom	Maria	to	love	tries	(7.72)

"a man whom Maria tries to love"



(7.73)

The relative sentence is headed by the finite verb *versucht* which lands in the nf of the modified noun *Mann*. In turn, the relative pronoun *den* lands in the rvf of *versucht*. The other dependents of *versucht*, *Maria* and *zu lieben*, land in its mf and vcf respectively.

7.2.3 Non-finite verbs

We distinguish between non-finite verbs in *canonical position* and non-finite verbs in *non-canonical position*. Typically, verbs are in canonical position if the corresponding linearization is unmarked. The canonical position of a non-finite verb is to the left of its verbal governor (in a verb-final sentence) or in the right sentence bracket (in a verb-first or verb-second sentence). If a non-finite verb is either fronted into the Vorfeld, participates in a pied piping construction or is intraposed or extraposed, we say that this verb is in non-canonical position.

Canonical position

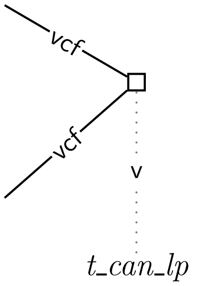
If in canonical position (lexical type: *t_can_lp*), a non-finite verb lands in the vcf of its governing verb, has node label v, and offers vcf. It does not block any role:

$$t\_can\_lp \quad = \quad \begin{bmatrix} \mathsf{labels}_{\mathrm{LP}} & : & \{\mathsf{vcf}\} \\ \mathsf{labels}_{\mathrm{N}} & : & \{\mathsf{v}\} \\ \mathsf{valency}_{\mathrm{LP}} & : & \{\mathsf{vcf}?\} \\ \mathsf{blocks} & : & \emptyset \end{bmatrix} \tag{7.74}$$

(7.74) induces the following topological domain:

$$\mathsf{vcf} \prec \mathsf{v} \tag{7.75}$$

The corresponding graphical illustration is shown here:



(7.76)

In canonical position, non-finite verbs do not offer the Mittelfeld-position mf: non-verbal dependents are thus forced to climb up into the mf or vf offered by nodes higher up in the tree. This idea is also reflected in Gerdes & Kahane's (2001) approach where non-finite verbs in the right sentence bracket open a 'degenerate' box with only one position offered to a verbal dependent to the left (corresponding to the offered vcf-position in our proposal).

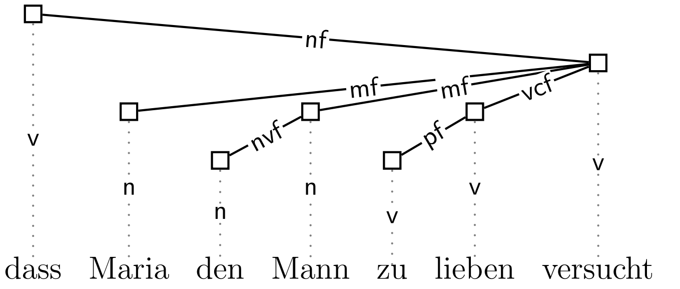
It should be noted that verbs in canonical position do not offer fields for verbal dependents in any of the non-canonical positions vf, rvf, if and vxf, and they do not offer nf.

Example

An example analysis involving a non-finite verb in canonical position is given below:

dass	Maria	den	Mann	zu	lieben	versucht
that	Maria	the	man(acc)	to	love	tries

“that Maria tries to love the man” (7.77)



(7.78)

Here, the *zu*-particle of the non-finite verb *lieben* lands in its pf. Since *lieben* in canonical position (vcf), it does not offer mf, and its object *Mann* is forced climb up into the mf of the finite verb *versucht.*

Non-canonical position

In non-canonical position, i.e. in the vf (fronted), rvf (pied piped), if (intraposed) or vxf (extraposed), a non-finite verb is assigned the following lexical type:

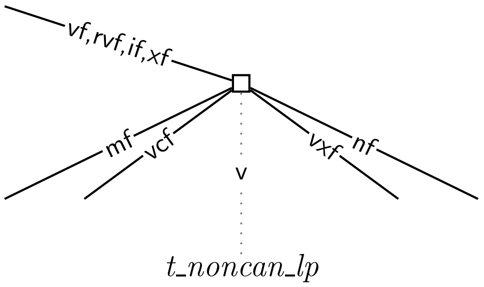
$$\textit{t\_noncan\_lp} \;=\; \begin{bmatrix} \mathsf{labels}_{\text{LP}} & : & \{\mathsf{vf},\mathsf{rvf},\mathsf{if},\mathsf{vxf}\} \\ \mathsf{labels}_{\text{N}} & : & \{\mathsf{v}\} \\ \mathsf{valency}_{\text{LP}} & : & \{\mathsf{mf}{*},\mathsf{vcf}?,\mathsf{vxf}?,\mathsf{nf}?\} \\ \mathsf{blocks} & : & \{\mathsf{adv},\mathsf{pp}\} \end{bmatrix} \tag{7.79}$$

Verbs in non-canonical position block their adjuncts, i.e. adverbs (adv) and PPs (pp).

The topological domain corresponding to (7.79) is shown below:

$$\mathsf{mf} \prec \mathsf{vcf} \prec \mathsf{v} \prec \mathsf{vxf} \prec \mathsf{nf} \tag{7.80}$$

And here is an illustration of lexical type *t_noncan_lp*:



(7.81)

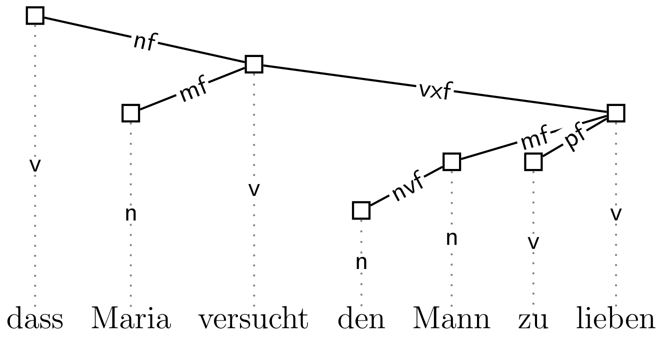
Similar to finite verbs in verb-final position, non-finite verbs in non-canonical position offer the fields mf and vcf to their left, and vxf and nf to their right.

Example

Here is an example sentence including an infinitive in non-canonical position (extraposed):

dass	Maria	versucht,	den	Mann	zu	lieben
that	Maria	tries,	the	man(acc)	to	love

"that Maria tries to love the man" (7.82)



(7.83)

Here, the VP *den Mann zu lieben* is extraposed into the vxf of the finite verb *versucht*. The object *Mann* of *lieben* is not forced to climb up as in the canonical case (7.78) because *lieben* is in non-canonical position, and hence *does* offer the field mf for *Mann* to land in.

Distinguishing non-canonical positions

Only certain kinds of non-finite verbs can land in certain non-canonical positions. Roughly, only *zu*-infinitives can land in any non-canonical position. Modals like *können* can be fronted or extraposed, but not intraposed or pied piped. All other verbs can only be fronted, but neither extraposed nor intraposed or pied piped. To capture these distinctions, we introduce the additional lexical types *t_front_lp* for fronted verbs, *t_pied_lp* for pied piped verbs, *t_intra_lp* for intraposed verbs and *t_extra_lp* for extraposed verbs. Here is the lexical type for fronted verbs:

$$t\_front\_lp \quad = \quad t\_noncan\_lp \sqcap \left[\ \mathsf{labels}_{\mathrm{LP}} \ : \ \{\mathsf{vf}\} \ \right] \tag{7.84}$$

Pied piped verbs inherit from *t_pied_lp*:

$$t\_pied\_lp \quad = \quad t\_noncan\_lp \sqcap \left[\ \mathsf{labels}_{\mathrm{LP}} \ : \ \{\mathsf{rvf}\} \ \right] \tag{7.85}$$

Intraposed verbs inherit from lexical type *t_intra_lp*:

$$t\_intra\_lp \quad = \quad t\_noncan\_lp \sqcap \left[\ \mathsf{labels}_{\mathrm{LP}} \ : \ \{\mathsf{if}\} \ \right] \tag{7.86}$$

And extraposed verbs from lexical type *t_extra_lp*:

$$t\_extra\_lp \quad = \quad t\_noncan\_lp \sqcap \left[\ \mathsf{labels}_{\mathrm{LP}} \ : \ \{\mathsf{vxf}\} \ \right] \tag{7.87}$$

Zu-infinitives

Zu-infinitives inherit from lexical type *t_vzu_lp*:

$$t\_vzu\_lp \quad = \quad \left[\begin{array}{rcl} \mathsf{valency}_{\mathrm{LP}} & : & \{\mathsf{pf}\} \\ \mathsf{blocks} & : & \{\mathsf{zu}\} \end{array} \right] \tag{7.88}$$

In their field valency, they require their *zu*-particle to land in the particle field (pf). They also block their *zu*-particle (role zu).

7.2.4 Complementizers

Here is the LP part of the lexical type *t_comp_lp* for complementizers such as *dass*:

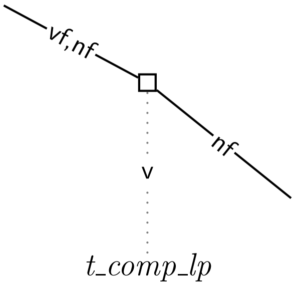
$$t\_comp\_lp \quad = \quad \left[\begin{array}{rcl} \mathsf{labels}_{\mathrm{LP}} & : & \{\mathsf{vf}, \mathsf{nf}\} \\ \mathsf{labels}_{\mathrm{N}} & : & \{\mathsf{v}\} \\ \mathsf{valency}_{\mathrm{LP}} & : & \{\mathsf{nf}\} \\ \mathsf{blocks} & : & \emptyset \end{array} \right] \tag{7.89}$$

Complementizers land either in the vf or in the nf of a verb. They offer the field nf for the subordinate clause, and block no role.

The corresponding topological domain requires that the complementizer (node label: v) precede the subordinate clause in the nf:

$$\mathsf{v} \prec \mathsf{nf} \tag{7.90}$$

Here is a graphical illustration of (7.89):

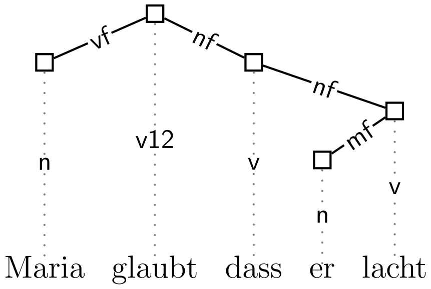


(7.91)

Example

We give an example LP tree analysis below:

Maria glaubt, dass er lacht.
Maria thinks, that he laughs. (7.92)
"Maria thinks that he laughs."



(7.93)

In (7.93), the complementizer *dass* lands in the nf of the finite verb *glaubt* in verb-second position. In turn, the subordinate clause headed by *lacht* lands in the nf of *dass*.

7.2.5 Adverbs

Adverbs inherit from *t_adv_lp*:

$$t\_adv\_lp = \begin{bmatrix} \mathsf{labels}_{\text{LP}} & : & \{\mathsf{vf}, \mathsf{mf}\} \\ \mathsf{labels}_{\text{N}} & : & \{\mathsf{v}\} \end{bmatrix} \tag{7.94}$$

Adverbs can land in the vf or mf and are assigned node label v.

7.2.6 Zu-particle

The *zu*-particle inherits from lexical type *t_zu_lp*:

$$t\_zu\_lp \quad = \quad \left[\begin{array}{lcl} \mathsf{labels}_{\mathrm{LP}} & : & \{\mathsf{pf}\} \\ \mathsf{labels}_{\mathrm{N}} & : & \{\mathsf{v}\} \end{array}\right] \tag{7.95}$$

The *zu* particle lands in the pf and is assigned node label v.

7.2.7 Nouns

All nouns inherit from the following lexical type:

$$t\_noun\_lp \quad = \quad \left[\begin{array}{lcl} \mathsf{labels}_{\mathrm{LP}} & : & \{\mathsf{vf}, \mathsf{rvf}, \mathsf{mf}, \mathsf{nvf}, \mathsf{nxf}\} \\ \mathsf{labels}_{\mathrm{N}} & : & \{\mathsf{n}\} \end{array}\right] \tag{7.96}$$

(7.96) states that the only accepted node label of a noun is n. Nouns can land either in the Vorfeld (vf), relative clause Vorfeld (rvf) or the Mittelfeld (mf) of a verb. As a genitival modifier, they can land either to the left of a governing noun in the nominal Vorfeld (nvf) or to the right in the nominal extraposition field (nxf).

Common nouns

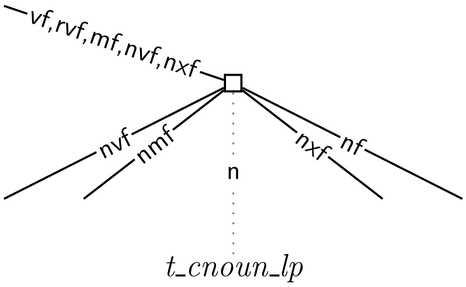
Common nouns inherit from lexical type *t_cnoun_lp*:

$$t\_cnoun\_lp \quad = \quad t\_noun\_lp \sqcap \left[\begin{array}{rcl} \mathsf{valency}_{\mathrm{LP}} & : & \{\mathsf{nvf}, \mathsf{nmf}{*}, \mathsf{nxf}?, \mathsf{nf}?\} \\ \mathsf{blocks} & : & \{\mathsf{det}, \mathsf{adj}, \mathsf{genmod}\} \end{array}\right] \tag{7.97}$$

The lexical type (7.97) induces the following topological domain:

$$\mathsf{nvf} \prec \mathsf{nmf} \prec \mathsf{n} \prec \mathsf{nxf} \prec \mathsf{nf} \tag{7.98}$$

And we display the corresponding illustration below:



(7.99)

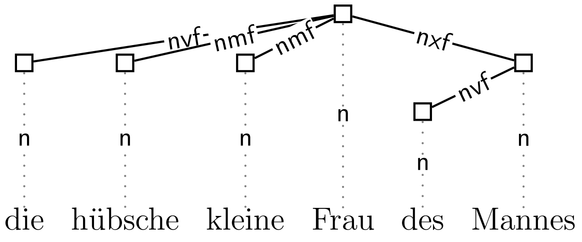
Each noun is assigned the node label n. To its left, it offers the fields nvf and nmf: nvf for its determiner or for a fronted genitival modifier and nmf for an arbitrary number of adjectives. To its right, a noun offers nxf and nf: nxf is the landing site for either a genitival or a prepositional modifier, and nf the landing site for non-extraposed relative clauses. Determiners, adjectives and genitival modifiers are blocked, whereas PPs and relative clauses may be extracted.

Examples

Genitival modifiers may either follow or precede the noun they modify. In the example below, the genitival modifier *des Mannes* lands in the nxf to the right of the noun *Frau*:

die	hübsche	kleine	Frau	des	Mannes	
the	pretty	small	woman	the	man(gen)	(7.100)

"the man's pretty small woman"

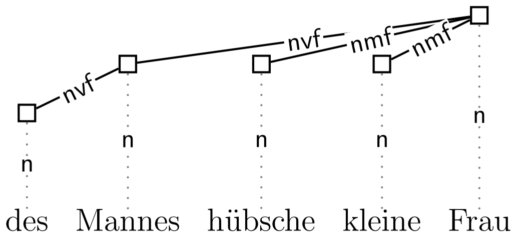


(7.101)

The genitival modifier *des Mannes precedes* the noun *Frau* in the following example:

des	Mannes	hübsche	kleine	Frau	
the	man(gen)	pretty	small	woman	(7.102)

"the man's pretty small woman"



(7.103)

Here, the modifier 'takes over' the position of noun's determiner and there may be no determiner in addition to it, which renders the following linearization ungrammatical:

∗	die	des	Mannes	hübsche	kleine	Frau	(7.104)
	die	the	man(gen)	pretty	small	woman	

In our grammar, we capture this fact as follows: nouns offer the nvf-field, and both determiners and genitival modifiers are allowed to land that field. Since the nvf may only contain up to one topological dependent, either the determiner or the genitival modifier may land in there, but not both.

Proper names

Proper names inherit from the following lexical type:

$$t\_pname\_lp \quad = \quad t\_noun\_lp \sqcap \begin{bmatrix} \textsf{valency}_{\text{LP}} & : & \{\textsf{nvf}?, \textsf{nmf}*, \textsf{nxf}?, \textsf{nf}?\} \\ \textsf{blocks} & : & \{\textsf{det}, \textsf{adj}, \textsf{genmod}\} \end{bmatrix} \tag{7.105}$$

The only difference between proper names and common nouns in the grammar fragment is that proper names have an optional nominal Vorfeld (nvf), whereas the nvf is obligatory for common nouns.

Personal pronouns

Personal pronouns inherit from the lexical type given below:

$$t\_perpro\_lp \quad = \quad t\_noun\_lp \tag{7.106}$$

Relative pronouns

Relative pronouns inherit from the lexical type given below:

$$t\_relpro\_lp \quad = \quad t\_noun\_lp \sqcap \begin{bmatrix} \textsf{labels}_{\text{LP}} & : & \{\textsf{rvf}, \textsf{mf}, \textsf{nvf}, \textsf{nxf}\} \end{bmatrix} \tag{7.107}$$

Relative pronouns typically land in the ‘relative clause Vorfeld’ of the finite verb heading the relative clause. In addition, they can land in the Mittelfeld of a pied piped *zu*-infinitive VP, or in the nvf or nxf of a pied piped NP, or in the nxf of a pied piped PP. Contrary to other nouns, relative pronouns cannot land in the vf.

7.2.8 Determiners

Determiners inherit from lexical type *t_det_lp*:

$$t\_det\_lp \quad = \quad \begin{bmatrix} \textsf{labels}_{\text{LP}} & : & \{\textsf{nvf}\} \\ \textsf{labels}_{\text{N}} & : & \{\textsf{n}\} \end{bmatrix} \tag{7.108}$$

t_det_lp requires that determiners land in the the nvf. Their node label is n.

7.2.9 Adjectives

Adjectives inherit from *t_adj_lp*:

$$t\_adj\_lp \;=\; \begin{bmatrix} \mathsf{labels}_{\mathrm{LP}} & : & \{\mathsf{nmf}\} \\ \mathsf{labels}_{\mathrm{N}} & : & \{\mathsf{n}\} \end{bmatrix} \tag{7.109}$$

Adjectives land in the nmf and are accept node label n.

7.2.10 Prepositions

Prepositions inherit from the following lexical type:

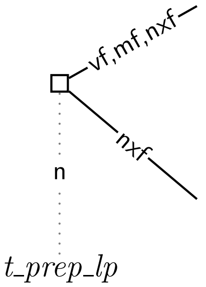
$$t\_prep\_lp \;=\; \begin{bmatrix} \mathsf{labels}_{\mathrm{LP}} & : & \{\mathsf{vf}, \mathsf{rvf}, \mathsf{mf}, \mathsf{nxf}\} \\ \mathsf{labels}_{\mathrm{N}} & : & \{\mathsf{n}\} \\ \mathsf{valency}_{\mathrm{LP}} & : & \{\mathsf{nxf}\} \\ \mathsf{blocks} & : & \{\mathsf{obj}, \mathsf{iobj}\} \end{bmatrix} \tag{7.110}$$

Prepositions may either land in the vf or the mf of a verb, or to the right of a noun in the nxf. In addition, they can also land in the relative clause Vorfeld rvf of a relative clause when they participate in a pied piping construction. Their node label is n and their nominal complement is required to land in their nxf. Prepositions block the roles obj and iobj.

The following topological domain corresponds to lexical type *t_prep_lp*:

$$\mathsf{n} \prec \mathsf{nxf} \tag{7.111}$$

A graphical illustration of *t_prep_lp* is displayed here:

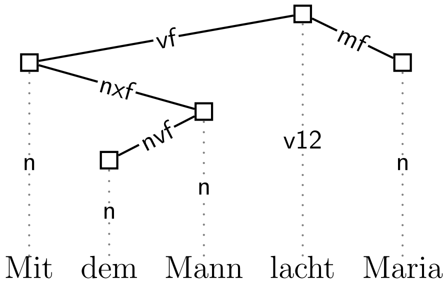


(7.112)

Example

We give an example LP tree analysis below:

$$\begin{array}{lllll} \text{Mit} & \text{dem} & \text{Mann} & \text{lacht} & \text{Maria.} \\ \text{With} & \text{the} & \text{man} & \text{laughs} & \text{Maria.} \\ \multicolumn{5}{l}{\textit{“Maria laughs with the man.”}} \end{array} \qquad (7.113)$$



(7.114)

Here, the preposition *mit* lands in the Vorfeld (vf) of the finite verb *lacht* and *Mann* lands in the nxf of *mit*.

7.3 Lexical entries

In this section, we demonstrate how to derive lexicon entries from the lexical types defined introduced above.

7.3.1 Finite verbs

Here is how we obtain the lexical entry for the finite verb *gibt* in verb-final subordinate clause position:

$$(\textit{gibt} \quad , \quad \textit{t\_fin\_id} \sqcap \textit{t\_ditr\_id} \sqcap \textit{t\_sub\_lp}) \qquad (7.115)$$

gibt is a finite verb (*t_fin_id*), it is ditransitive (*t_ditr_id*) and lands in verb-final subordinate clause position (*t_sub_lp*).

(7.115) gives rise to the following lexical entry:

$$(\textit{gibt} \quad , \quad \begin{bmatrix} \mathsf{labels}_{\mathrm{ID}} & : & \{\mathsf{sub}\} \\ \mathsf{valency}_{\mathrm{ID}} & : & \{\mathsf{subj}, \mathsf{adv}*, \mathsf{pp}?, \mathsf{obj}, \mathsf{iobj}\} \\ \mathsf{labels}_{\mathrm{LP}} & : & \{\mathsf{nf}\} \\ \mathsf{labels}_{\mathrm{N}} & : & \{\mathsf{v}\} \\ \mathsf{valency}_{\mathrm{LP}} & : & \{\mathsf{if}?, \mathsf{mf}*, \mathsf{vcf}?, \mathsf{vxf}?, \mathsf{nf}?\} \\ \mathsf{blocks} & : & \mathcal{R} \end{bmatrix}) \qquad (7.116)$$

Making use of lexical ambiguity, we propose for each finite verb two further lexical entries applying in left sentence bracket position (7.117) and in verb-final relative clause position (7.118) respectively:

$$(\textit{gibt} \quad , \quad \textit{t\_fin\_id} \sqcap \textit{t\_ditr\_id} \sqcap \textit{t\_v12\_lp}) \tag{7.117}$$

$$(\textit{gibt} \quad , \quad \textit{t\_fin\_id} \sqcap \textit{t\_ditr\_id} \sqcap \textit{t\_rel\_lp}) \tag{7.118}$$

7.3.2 Non-finite verbs

Below, we give an example of how to obtain the lexical entry for the past participle *gegeben*:

$$(\textit{gegeben} \quad , \quad \textit{t\_vpp\_id} \sqcap \textit{t\_ditr\_id} \sqcap \textit{t\_can\_lp}) \tag{7.119}$$

gegeben is a past participle (*t_vpp_id*), ditransitive (*t_ditr_id*) and lands in canonical position (*t_can_lp*).

(7.119) induces the following lexical entry:

$$(\textit{gegeben} \quad , \quad \begin{bmatrix} \mathsf{labels}_{\mathrm{ID}} & : & \{\mathsf{vpp}\} \\ \mathsf{valency}_{\mathrm{ID}} & : & \{\mathsf{adv}*, \mathsf{pp}?, \mathsf{obj}, \mathsf{iobj}\} \\ \mathsf{labels}_{\mathrm{LP}} & : & \{\mathsf{vcf}\} \\ \mathsf{labels}_{\mathrm{N}} & : & \{\mathsf{v}\} \\ \mathsf{valency}_{\mathrm{LP}} & : & \{\mathsf{vcf}?\} \\ \mathsf{blocks} & : & \emptyset \end{bmatrix}) \tag{7.120}$$

Past participles like *gegeben* can also land in the Vorfeld in a verb-second sentence. We reflect this by proposing an additional lexical entry:

$$(\textit{gegeben} \quad , \quad \textit{t\_vpp\_id} \sqcap \textit{t\_ditr\_id} \sqcap \textit{t\_front\_lp}) \tag{7.121}$$

7.3.3 Complementizers

Here is how we obtain the lexical entry for the complementizer *dass*:

$$(\textit{dass} \quad , \quad \textit{t\_comp\_id} \sqcap \textit{t\_comp\_lp}) \tag{7.122}$$

$$(\textit{dass} \quad , \quad \begin{bmatrix} \mathsf{labels}_{\mathrm{ID}} & : & \{\mathsf{comp}\} \\ \mathsf{valency}_{\mathrm{ID}} & : & \{\mathsf{sub}\} \\ \mathsf{labels}_{\mathrm{LP}} & : & \{\mathsf{vf}, \mathsf{nf}\} \\ \mathsf{labels}_{\mathrm{N}} & : & \{\mathsf{v}\} \\ \mathsf{valency}_{\mathrm{LP}} & : & \{\mathsf{nf}\} \\ \mathsf{blocks} & : & \emptyset \end{bmatrix}) \tag{7.123}$$

7.3.4 Nouns

We construct the lexical entry for the common noun *Frau* below:

$$(\mathit{Frau} \quad , \quad \mathit{t\_cnoun\_id} \sqcap \mathit{t\_cnoun\_lp}) \tag{7.124}$$

$$(\mathit{Frau} \quad , \quad \left[\begin{array}{rcl} \mathsf{labels}_{\mathrm{ID}} & : & \{\mathsf{subj}, \mathsf{obj}, \mathsf{iobj}, \mathsf{genobj}, \mathsf{genmod}\} \\ \mathsf{valency}_{\mathrm{ID}} & : & \{\mathsf{det}?, \mathsf{adj}*, \mathsf{genmod}?, \mathsf{pp}?, \mathsf{rel}?\} \\ \mathsf{labels}_{\mathrm{LP}} & : & \{\mathsf{vf}, \mathsf{rvf}, \mathsf{mf}, \mathsf{nvf}, \mathsf{nxf}\} \\ \mathsf{labels}_{\mathrm{N}} & : & \{\mathsf{n}\} \\ \mathsf{valency}_{\mathrm{LP}} & : & \{\mathsf{nvf}, \mathsf{nmf}*, \mathsf{nxf}?, \mathsf{nf}?\} \\ \mathsf{blocks} & : & \{\mathsf{det}, \mathsf{adj}, \mathsf{genmod}\} \end{array}\right]) \tag{7.125}$$

7.3.5 Prepositions

An example lexical entry for the preposition *mit* is constructed below:

$$(\mathit{mit} \quad , \quad \mathit{t\_prep\_iobj\_id} \sqcap \mathit{t\_prep\_lp}) \tag{7.126}$$

$$(\mathit{mit} \quad , \quad \left[\begin{array}{rcl} \mathsf{labels}_{\mathrm{ID}} & : & \{\mathsf{pp}\} \\ \mathsf{valency}_{\mathrm{ID}} & : & \{\mathsf{iobj}\} \\ \mathsf{labels}_{\mathrm{LP}} & : & \{\mathsf{vf}, \mathsf{rvf}, \mathsf{mf}, \mathsf{nxf}\} \\ \mathsf{labels}_{\mathrm{N}} & : & \{\mathsf{n}\} \\ \mathsf{valency}_{\mathrm{LP}} & : & \{\mathsf{nxf}\} \\ \mathsf{blocks} & : & \{\mathsf{obj}, \mathsf{iobj}\} \end{array}\right]) \tag{7.127}$$

7.4 Summary

We proposed a TDG grammar fragment for German. Since TDG is a highly lexicalized grammar formalism, the focus of this chapter was on how to specify the lexical type hierarchy. We specified the lexicon in a modular fashion by introducing the lexical types concerned with ID and LP (and ID/LP) attributes separately. In the final section, we showed how to put together lexical entries from the lexical types introduced before.

In the following chapter, we demonstrate that the grammar fragment is able to tackle a variety of notorious phenomena in German syntax.

Chapter 8

Phenomena

This chapter applies the TDG grammar fragment laid out in the preceding chapter to some of the most notorious phenomena in German syntax. We begin with analyzing scrambling (Ross 1967) in the Mittelfeld. Then, we turn to VP*-dislocation-related constructions, including extraposition, intraposition and fronting. The so-called auxiliary flip construction (Hinrichs & Nakazawa 1994) is discussed thereafter, followed by a section about relative clauses, covering relative clause extraposition and pied piping. Many of the analyses given in this chapter have already been presented in (Duchier & Debusmann 2001).*

8.1 Scrambling

Non-verbal material can be quite freely permuted in the Mittelfeld: in addition to the linearization glossed in (8.1), the linearization in (8.2) is also possible, where the VP *einen Roman zu schreiben* is discontinuous:

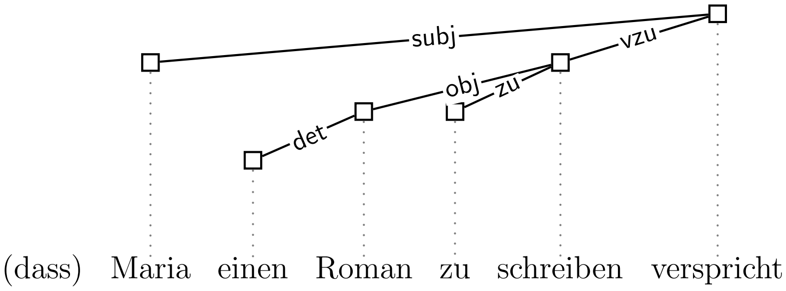
(dass)	Maria	einen	Roman	zu	schreiben	verspricht	
(that)	Maria	a	novel(acc)	to	write	promises	(8.1)

"(that) Maria promises to write a novel"

(dass)	einen	Roman	Maria	zu	schreiben	verspricht	
(that)	a	novel(acc)	Maria	to	write	promises	(8.2)

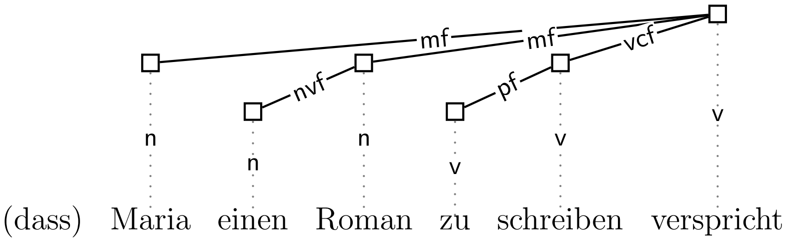
"(that) Maria promises to write a novel"

The ID tree is the same for both (8.1) and (8.2):

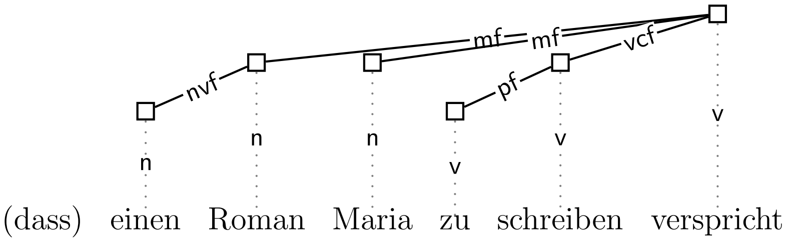


(8.3)

And here are the LP trees corresponding to (8.1) and (8.2):



(8.4)



(8.5)

Because both NPs *Maria* and *einen Roman* land in the same field, the mf of the finite verb *verspricht*, they are not ordered with respect to each other and both LP trees (8.4) and (8.5) are licensed. In (8.4), *Maria* precedes *einen Roman*, and in (8.5), *einen Roman* precedes *Maria*.

8.2 VP dislocation

There are four kinds of VP dislocation in German: fronting, extraposition, intraposition and VP pied piping. In verb-second sentences, VPs headed by any non-finite verb may be fronted into the Vorfeld. VPs headed by a *zu*-infinitive may, in addition, be dislocated to the right of their verbal governors or to the left. Dislocation to the right is called extraposition. Dislocation to the left of the Mittelfeld (but not into the Vorfeld) is called intraposition. In relative clauses (see also section 8.4 below), *zu*-infinitives can additionally take part in a pied piping construction.

Notice that for simplicity, we do not take into account the issue of coherence and incoherence. In simple terms, only a certain class of verbs which can construct 'incoherently' allows extraposition, intraposition or pied piping of its *zu*-infinitival complement. We discuss the issue of coherence and incoherence in appendix A.

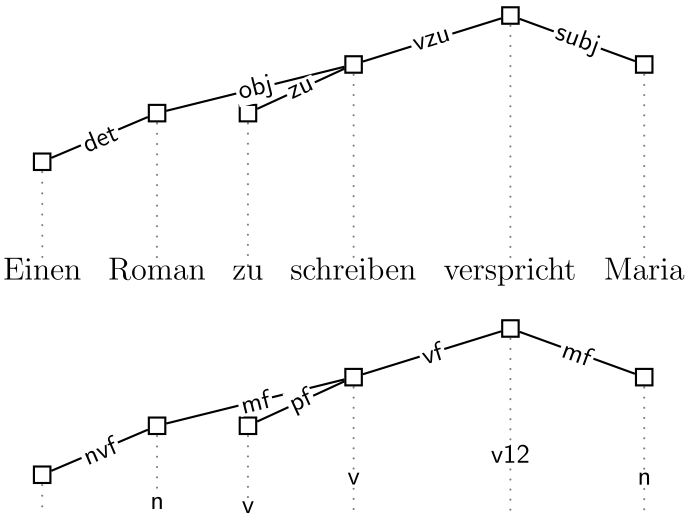
8.2.1 Fronting

Here is an example where the full VP *einen Roman zu schreiben* is fronted to the left of the finite verb *verspricht*:

Einen	Roman	zu	schreiben	verspricht	Maria.
A	novel(acc)	to	write	promises	Maria.

"Maria promises to write a novel." (8.6)

We show the corresponding ID/LP analysis below:



(8.7)

Here, the *zu*-infinitive *zu schreiben* lands in the Vorfeld (vf) of the finite verb *verspricht*. It takes along its object *Roman* which lands in its Mittelfeld (mf). The subject *Maria* lands in the mf of its governor *verspricht*.

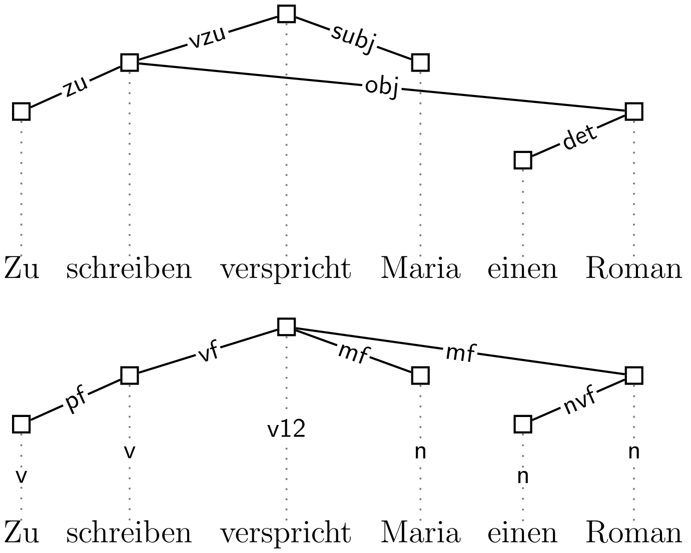
8.2.2 Partial fronting

Dependents of fronted verbs may be dislocated into the Mittelfeld, resulting in partial fronting. In the following example, *zu schreiben* is fronted but its object NP *einen Roman* is dislocated into the Mittelfeld:

Zu	schreiben	verspricht	Maria	einen	Roman.
To	write	promises	Maria	a	novel(acc).

"Maria promises to write a novel." (8.8)

In the corresponding ID/LP analysis, the partial VP *zu schreiben* is fronted into the vf while the object *Roman* climbs up into the mf of *verspricht*:


vzu
subj
zu
obj
det
Zu schreiben verspricht Maria einen Roman
vf
mf
mf
pf
nvf
v12
v
v
n
n
n
Zu schreiben verspricht Maria einen Roman

(8.9)

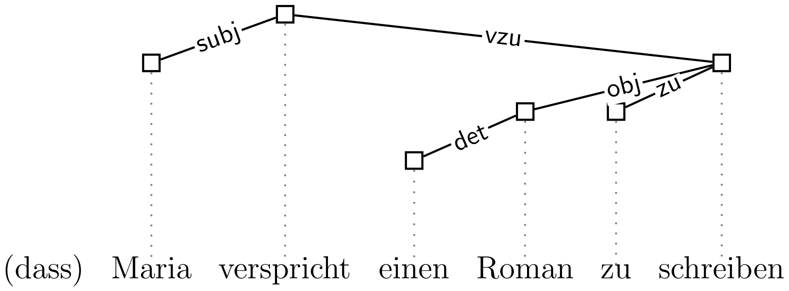
8.2.3 Extraposition

Extraposition of VPs proceeds similar to fronting: here, a VP is dislocated to the right of its verbal governor. An example is glossed below:

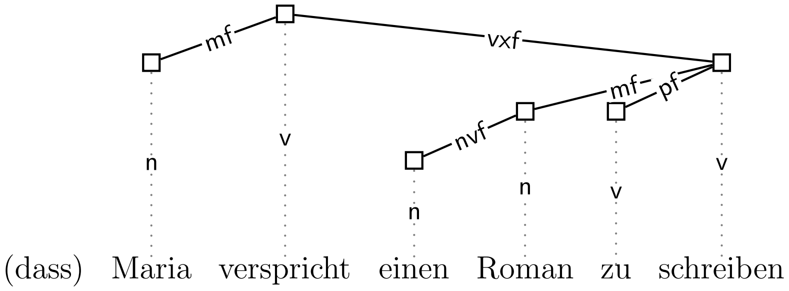
(dass)	Maria	verspricht,	einen	Roman	zu	schreiben
(that)	Maria	promises,	a	novel(acc)	to	write

"(that) Maria promises to write a novel" (8.10)

The landing site for extraposed VPs is the verb extraposition field (vxf). In the ID/LP analysis presented below in (8.11), the VP *einen Roman zu schreiben* is extraposed, taking its object-dependent *einen Roman* along in its mf:



(8.11)



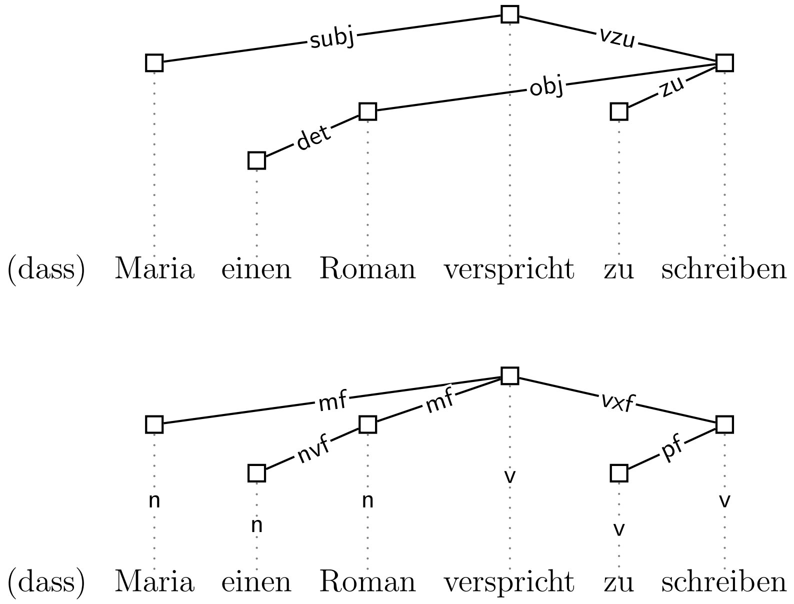
8.2.4 Partial extraposition

Partial VPs may also be extraposed. An example is depicted below:

(dass)	Maria	einen	Roman	verspricht,	zu	schreiben	
(that)	Maria	a	novel(acc)	promises,	to	write	(8.12)

"(that) Maria promises to write a novel"

In the corresponding ID/LP analysis below, the extraposed partial VP *zu schreiben* lands in the vxf of *verspricht* and its object *Roman* climbs up into the mf of *verspricht*:


subj
vzu
obj
zu
det
(dass) Maria einen Roman verspricht zu schreiben
mf
mf
vxf
nvf
pf
n
n
n
v
v
v
(dass) Maria einen Roman verspricht zu schreiben

(8.13)

8.2.5 Intraposition

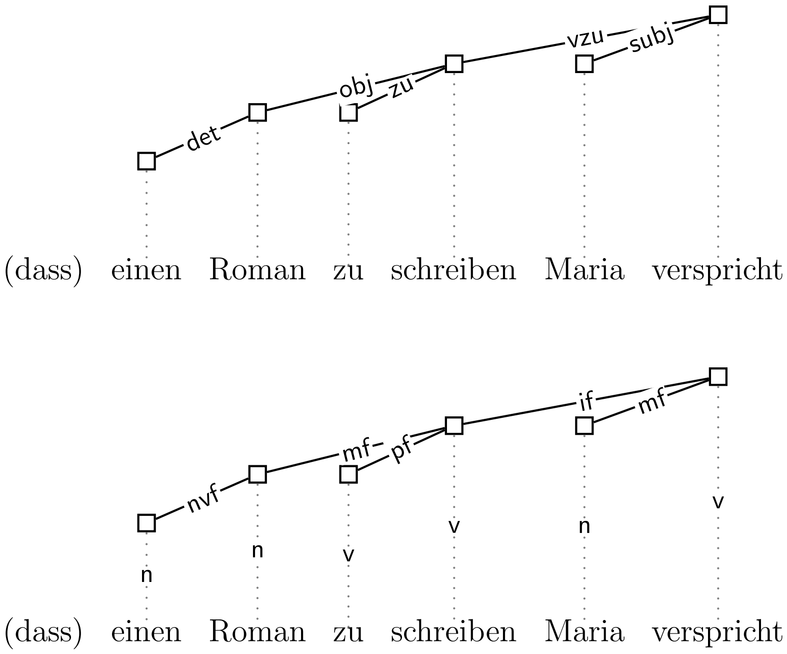
VPs headed by a *zu*-infinitive can also be dislocated to the left:

(dass)	einen	Roman	zu	schreiben	Maria	verspricht	
(that)	a	novel(acc)	to	write	Maria	promises	(8.14)

“(that) Maria promises to write a novel”

The corresponding ID/LP analysis is depicted in (8.15) below. Here, the infinitival complement *zu schreiben* lands in the intraposition field (if) and takes its object

Roman along in its mf:


det
obj
zu
vzu
subj
(dass) einen Roman zu schreiben Maria verspricht
nvf
mf
pf
if
mf
n
n
v
v
n
v
(dass) einen Roman zu schreiben Maria verspricht

(8.15)

About the possibility of partial VP intraposition there is no general agreement: Meurers & Kuthy (2001) for instance argue that partial VPs can never be intraposed, whereas Askedahl (1983) presents a set of examples which are marginal but still grammatical. In the present fragment, we do allow for partial intraposition: verbs in non-canonical position (including if) block the roles adv and pp but e.g. objects can climb through such verbs.

However, our framework also allows us to accomodate Meurers & Kuthy's (2001) view that intraposition must be treated differently from fronting and extraposition. To this end, we might change the lexical type *t_intra_lp* for verbs in intraposed position. To prohibit partial intraposition, we would make the lexical type for intraposed verbs block all roles, with the result that no node may climb through such a verb and only complete VPs can be intraposed:

$$t\_intra\_lp \quad = \quad t\_noncan\_lp \sqcap \begin{bmatrix} \mathsf{labels}_{\mathrm{LP}} & : & \{\mathsf{if}\} \\ \mathsf{blocks} & : & \mathcal{R} \end{bmatrix} \tag{8.16}$$

Because we think, following Haider (1985), that pied piping should be treated analogously to intraposition, the lexical type *t_pied_lp* for pied piped verbs would also block all roles:

$$t\_pied\_lp \quad = \quad t\_noncan\_lp \sqcap \begin{bmatrix} \mathsf{labels}_{\mathrm{LP}} & : & \{\mathsf{rvf}\} \\ \mathsf{blocks} & : & \mathcal{R} \end{bmatrix} \tag{8.17}$$

The lexical types *t_front_lp* for fronted verbs and *t_extra_lp* for extraposed verbs would remain as specified in chapter 7, still allowing for partial fronting and partial extraposition respectively.

8.3 Auxiliary flip

The auxiliary flip construction was first described by Bech (1955) using the term *Oberfeldumstellung.* More recently, Hinrichs & Nakazawa (1994) coined the name *auxiliary flip* in their account of the phenomenon in the HPSG-framework. In an auxiliary flip construction, the verbal complement of an auxiliary such as *werden* or *haben* follows the auxiliary rather than preceding it. The auxiliary is said to have 'flipped' to the left periphery of the verb cluster. We will elaborate on five aspects of the phenomenon: obligatory head-final placement, optional auxiliary flip, obligatory auxiliary flip, V-projection raising and VC-split.

8.3.1 Obligatory head-final placement

Verbal complements typically precede their governors. Here are two examples:

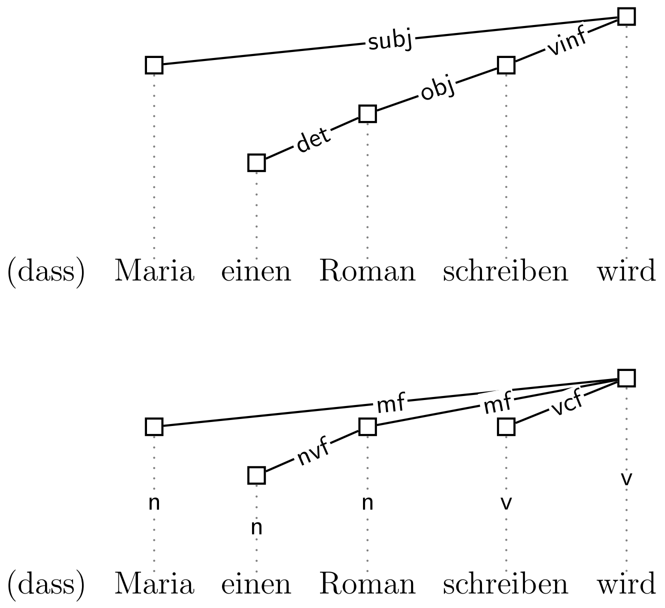
(dass)	Maria	einen	Roman	schreiben	wird	
(that)	Maria	a	novel(acc)	write	will	(8.18)

"(that) Maria will write a novel"

*	(dass)	Maria	einen	Roman	wird	schreiben	(8.19)
	(that)	Maria	a	novel(acc)	will	write	

In (8.18), the infinitival complement *schreiben* precedes its governor *wird.* Linearization (8.19) is ungrammatical because *schreiben* follows its governor rather than preceding it.

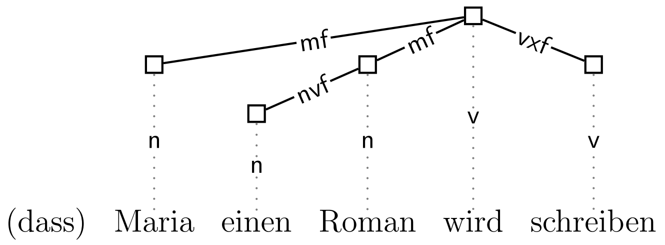
An ID/LP analysis of the grammatical linearization (8.18) is shown below:


subj
vinf
obj
det
(dass) Maria einen Roman schreiben wird
mf
mf
vcf
nvf
n
n
n
v
v
(dass) Maria einen Roman schreiben wird

(8.20)

Here, *schreiben* lands in canonical position (vcf) to the left of *wird*. The subject *Maria* lands in the Mittelfeld (mf) of *wird* and so does the object *Roman*.

A hypothetical LP tree for the ungrammatical linearization (8.19) is shown in (8.21) below. The ID tree remains the same as in (8.20).



(8.21)

Here, *schreiben* lands in the verb extraposition field (vxf) of its governor *wird*. How do we exclude incorrect analyses like (8.21)? By stipulating in the lexicon that main verbs such as *schreiben* can only land in canonical position (vcf) but not in extraposed position (vxf). That is, we pose only a lexical entry for *schreiben* which inherits from lexical type *t_can_lp*, but no lexical entries for *schreiben* inheriting from e.g. *t_extra_lp*:

$$(\mathit{schreiben} \quad , \quad \mathit{t\_vinf\_id} \sqcap \mathit{t\_tr\_id} \sqcap \mathit{t\_can\_lp}) \tag{8.22}$$

8.3.2 Optional auxiliary flip

The future-auxiliary *wird* gives rise to optional auxiliary flip in combination with a class of bare infinitival complements including modal verbs. Either of the two linearizations glossed below are grammatical:

(dass)	Maria	einen	Roman	schreiben	können	wird
(that)	Maria	a	novel(acc)	write	can	will

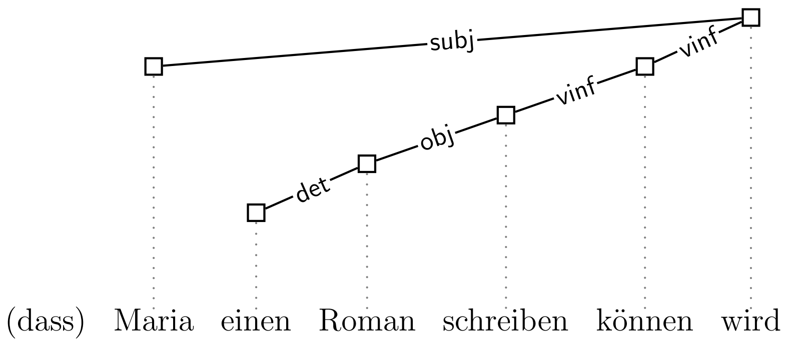
"(that) Maria will be able to write a novel" (8.23)

(dass)	Maria	einen	Roman	wird	schreiben	können
(that)	Maria	a	novel(acc)	will	write	can

"(that) Maria will be able to write a novel" (8.24)

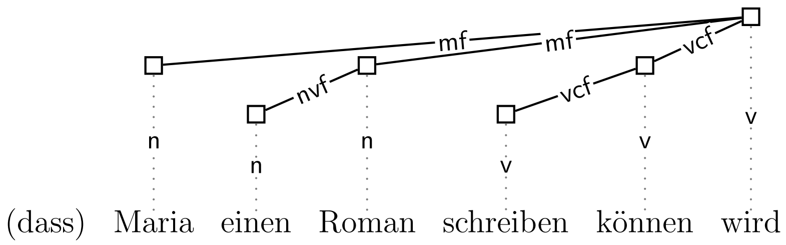
In (8.23), the bare infinitival modal verb complement *können* is in canonical position to the left of its governor *wird*, whereas in (8.24), *können* occurs to the right of *wird*. In the latter case, the auxiliary *wird* seems to have 'flipped' to the left of the verb cluster.

The ID tree analyses of (8.23) and (8.24) are the same:

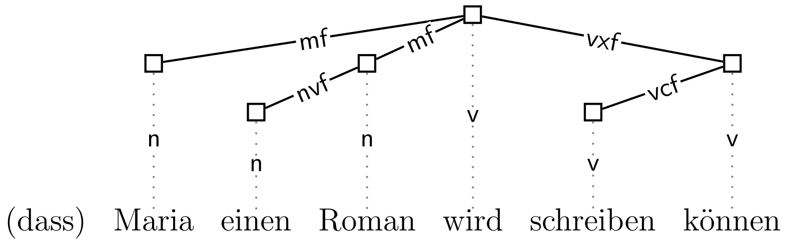


(8.25)

The LP tree analyses are given in (8.26) and (8.27) respectively:



(8.26)



(8.27)

In (8.26), the modal verb *können* lands in canonical position (vcf) to the left of its auxiliary governor *wird*. Contrarily, in (8.27) *können* lands in the verb extraposition field (vxf) to the right of *wird*. Thus, we do not assume that the auxiliary 'flips' to the left but rather that the bare infinitival complement of the auxiliary is extraposed to the right (into a non-canonical position).

Only the bare infinitival forms of a small class of words can be extraposed, including modal verbs. We propose for such words two lexical entries: one inherits from *t_can_lp* and one from *t_extra_lp*. For instance, here are the two lexical entries for the infinitival form of the modal verb *können*:

$$(\textit{können} \quad , \quad \textit{t\_vinf\_id} \sqcap \textit{t\_infc\_id} \sqcap \textit{t\_can\_lp}) \tag{8.28}$$

$$(\textit{können} \quad , \quad \textit{t\_vinf\_id} \sqcap \textit{t\_infc\_id} \sqcap \textit{t\_extra\_lp}) \tag{8.29}$$

(8.28) is selected if *können* lands in canonical position to the left of its governor and (8.29) if *können* lands in non-canonical, extraposed position to the right of its governor.

8.3.3 Obligatory auxiliary flip

Auxiliary flip is obligatory if the perfect-auxiliary *haben* is the head of a so-called *Ersatzinfinitiv* (substitute infinitive). A substitute infinitive exhibits bare infinitival inflection, yet syntactically acts as a past participle. Only modals, raising-to-object verbs such as *sehen* and the verb *helfen* can act as a substitute infinitive. Here are two examples, including the substitute infinitive *können* as the complement of the perfect-auxiliary *hat*:

(dass)	Maria	einen	Roman	hat	schreiben	können
(that)	Maria	a	novel(acc)	has	write	can

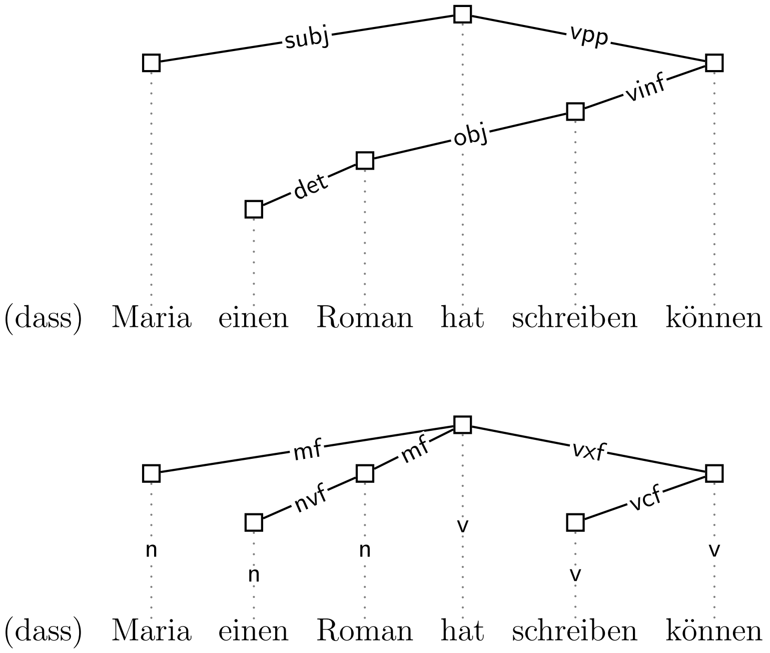
"(that) Maria has been able to write a novel" (8.30)

*	(dass)	Maria	einen	Roman	schreiben	können	hat
	(that)	Maria	a	novel(acc)	write	can	has

(8.31)

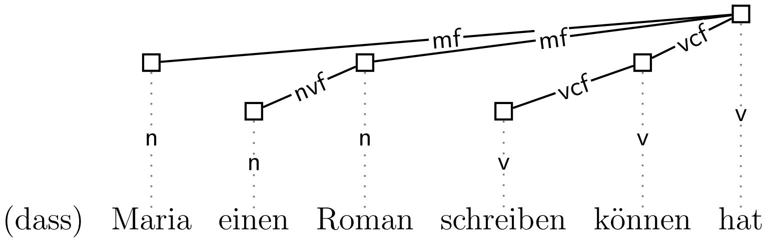
(8.30) is an auxiliary flip construction where the *hat* precedes its verbal complement *können*. Contrarily, *hat* follows *können* in the ungrammatical (non-auxiliary flip) construction shown in (8.31).

The ID/LP tree analysis of the grammatical linearization (8.30) is exhibited below:


subj
vpp
vinf
obj
det
(dass) Maria einen Roman hat schreiben können
mf
mf
vxf
nvf
vcf
n
n
n
v
v
v
(dass) Maria einen Roman hat schreiben können

(8.32)

And in (8.33), we show a hypothetical LP tree for the ungrammatical linearization (8.31). The ID tree remains the same as in ID/LP analysis (8.32).



(8.33)

Here, the substitute infinitive *können* lands in canonical position (vcf) to the left of its governor *hat*. We exclude this analysis by requiring that substitute infinitives such as *können* can only land in extraposed position (vxf) but not in canonical position (vcf).

We display the lexical entry for the substitute infinitival form of *können* below:

$$(\textit{können} \quad , \quad \textit{t\_vpp\_id} \sqcap \textit{t\_infc\_id} \sqcap \textit{t\_extra\_lp}) \tag{8.34}$$

Notice that contrary to the lexical entry for the bare infinitival form of *können* in (8.29) above, *können* in substitute infinitival form inherits from *t_vpp_id* (and not from *t_vinf_id*) because it syntactically acts as a past participle and not as a bare infinitive. Its set of accepted labels hence includes vpp instead of vinf.

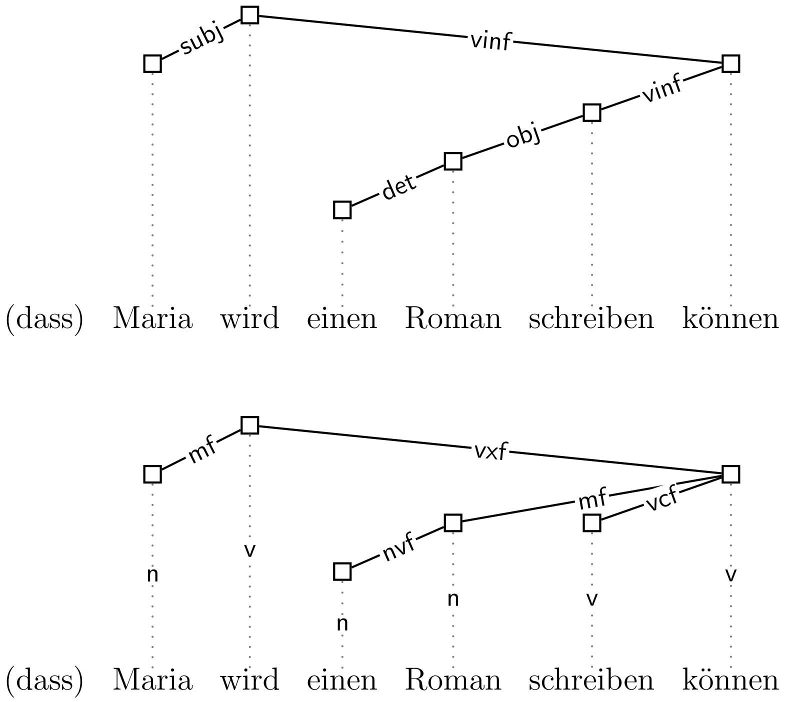
8.3.4 V-projection raising

V-projection raising describes a phenomenon where non-verbal material is interspersed in the verb cluster. In the example below, the NP *einen Roman* is positioned between *wird* and *schreiben können* in the verb cluster:

(dass)	Maria	wird	einen	Roman	schreiben	können	
(that)	Maria	will	a	novel(acc)	write	can	(8.35)

"(that) Maria will be able to write a novel"

In the corresponding ID/LP analysis, the extraposed bare infinitive *können* offers a mf-position on its own by virtue of landing in a non-canonical position, and the object *einen Roman* of *schreiben* climbs up to this position:

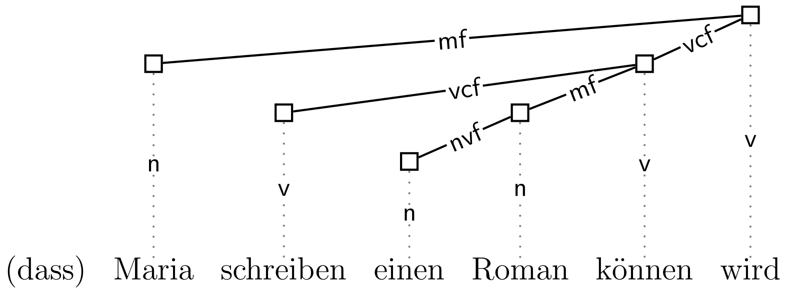

subj
vinf
vinf
obj
det
(dass) Maria wird einen Roman schreiben können
mf
vxf
mf
vcf
nvf
n
v
n
n
v
v
(dass) Maria wird einen Roman schreiben können

(8.36)

Notice that although we allow non-verbal material to be interspersed in the verb cluster as in (8.36), this does not lead to overgeneration. In the following, we show how we exclude various ungrammatical linearizations with non-verbal material interspersed in the verb cluster. For instance consider the linearization below where the NP *einen Roman* follows *schreiben* and precedes *können* and *wird*:

∗	(dass)	Maria	schreiben	einen	Roman	können	wird	(8.37)
	(that)	Maria	write	a	novel(acc)	can	will	

We show a hypothetical LP tree below:



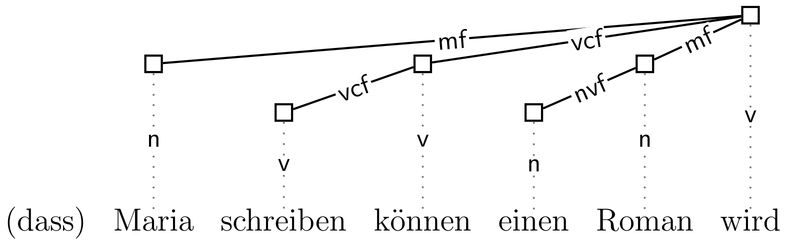
(8.38)

(8.38) is excluded by the order principle: *schreiben* lands in the vcf of *können* preceding *Roman* in the mf of *können*, but mf must precede vcf.

Another example is the following ungrammatical linearization:

∗	(dass)	Maria	schreiben	können	einen	Roman	wird	(8.39)
	(that)	Maria	write	can	a	novel(acc)	will	

of which we show a hypothetical LP tree analysis below:



(8.40)

This LP tree is also excluded by the order principle: *können* lands in the vcf of *wird*, preceding *Roman* in the mf of *wird*. But again, mf must precede vcf.

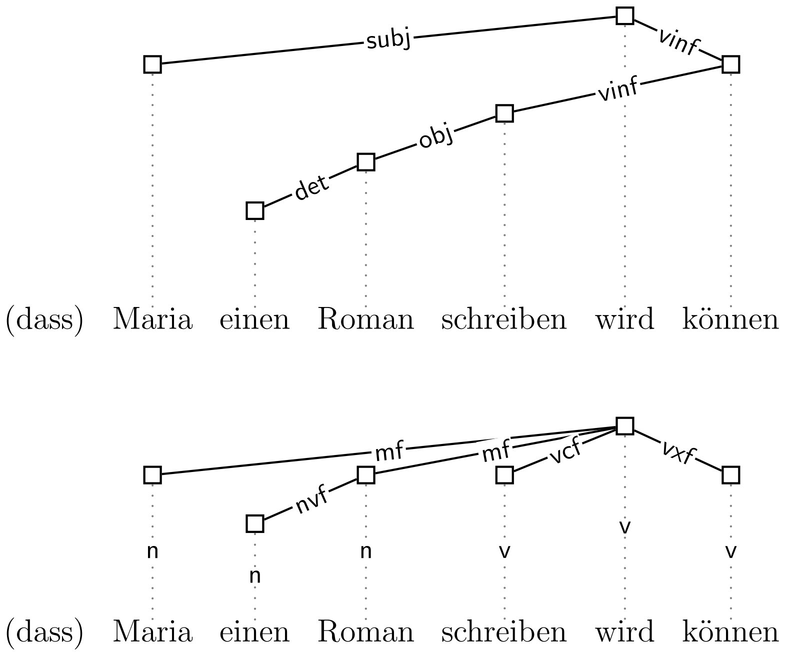
8.3.5 VC-split

Zwischenstellung (VC-split) is a construction first described by Meurers (1994). Here, the auxiliary is not 'flipped' to the left but into the middle of the verb-cluster. In other words, the verb-cluster is ‘split’. An example is given below, where the auxiliary *wird* lands in between *schreiben* and *können*:

(dass)	Maria	einen	Roman	schreiben	wird	können	
(that)	Maria	a	novel(acc)	write	will	can	(8.41)

“(that) Maria will be able to write a novel”

We display the corresponding ID/LP analysis below:


subj
vinf
vinf
obj
det
(dass) Maria einen Roman schreiben wird können
mf
mf
vcf
vxf
nvf
n
n
n
v
v
v
(dass) Maria einen Roman schreiben wird können

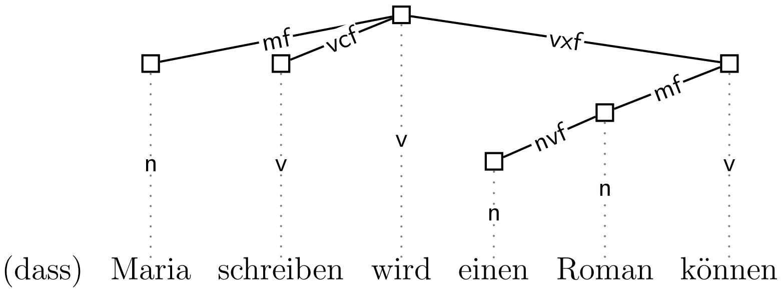
(8.42)

Here, *schreiben* climbs into the vcf of *wird*. By virtue of being in canonical position, *schreiben* offers no mf and therefore forces its object *Roman* to climb up (into the mf of *wird*). The modal verb *können* lands in the vxf of *wird*.

Below, we demonstrate that licensing the VC-split construction does not lead to overgeneration. E.g. we do not obtain the ungrammatical linearization below:

∗	(dass)	Maria	schreiben	wird	einen	Roman	können	(8.43)
	(that)	Maria	write	will	a	novel(acc)	can	

because the corresponding hypothetical LP tree is not licensed by our theory:



(8.44)

(8.44) violates the subtrees principle: *schreiben* has climbed up into the vcf of *wird* but has left its dependent object *Roman* behind lower in the tree (in the mf of *können*).

8.4 Relative clauses

In this section, we discuss two phenomena associated with relative clauses. The first is pied piping and the second relative clause extraposition.[1]

8.4.1 Pied Piping

In a pied piping ('Rattenfänger') construction, the relative pronoun at the left of a relative clause takes along its governor. We discern three cases of pied piping: NP pied piping, PP pied piping and VP pied piping.

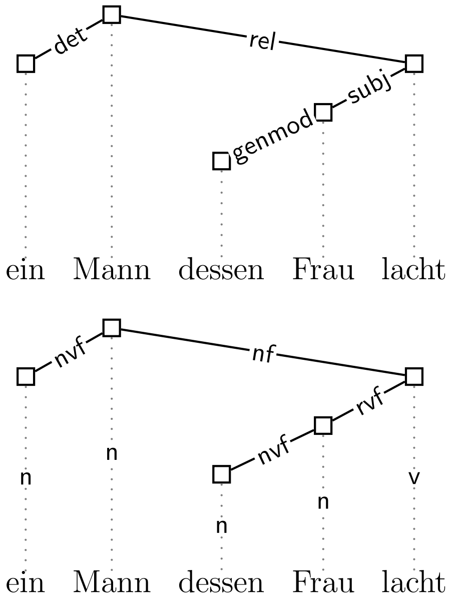
NP pied piping

Below, we present an example for NP pied piping, where the relative pronoun *dessen* takes along its nominal governor *Frau*:

ein	Mann,	dessen	Frau	lacht
a	man,	whose	woman	laughs

"a man whose woman laughs" (8.45)

[1] Notice that the grammar fragment as exhibited in chapter 7 gives rise to overgeneration with respect to relative clauses. We outline a possibility to tackle this by the *relative clause principle* in appendix A. Informally, the relative clause principle states that the relative pronoun must be in the yield of the relative clause Vorfeld (rvf).



(8.46)

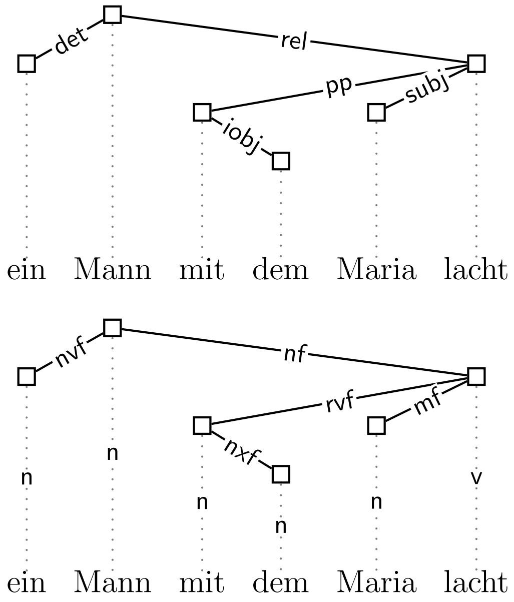
In (8.46), the noun *Frau* lands in the relative clause Vorfeld (rvf) of the finite verb *lacht*. The genitival relative pronoun *dessen* lands in the nvf of the noun.

PP pied piping

Here is an example for PP pied piping, where the relative pronoun *dem* takes along its prepositional governor *mit*:

ein	Mann,	mit	dem	Maria	lacht
a	man,	with	whom	Maria	laughs

"a man with whom Maria laughs" (8.47)



(8.48)

Here, the preposition *mit* lands in the rvf of the finite verb *lacht*, and the relative pronoun *dem* in the nxf of the preposition.

VP pied piping

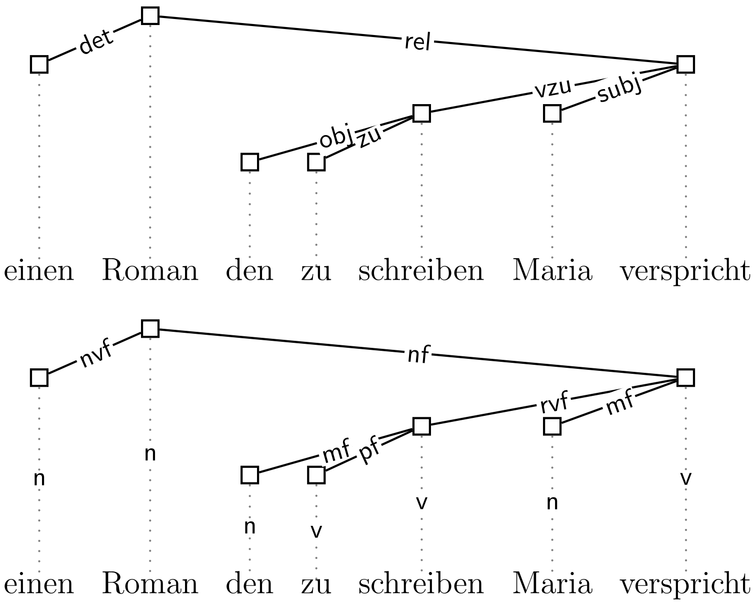
Another case of pied piping is VP pied piping. Here, the relative pronoun takes along its governing verb. In the example below, the relative pronoun *den* takes along the *zu*-infinitive *zu schreiben*:

einen	Roman,	den	zu	schreiben	Maria	verspricht	
a	novel(acc),	which(acc)	to	write	Maria	promises	(8.49)

"a novel which Maria promises to write"

Haider (1985) observed that pied piping can be regarded as an instance of intraposition: the same class of verbs (viz. *zu*-infinitives) can be intraposed as well as pied piped, whereas VPs headed by a bare infinitive or a past participle can neither be intraposed nor pied piped.

Here is an ID/LP analysis of (8.49):


det
rel
obj
zu
vzu
subj
einen Roman den zu schreiben Maria verspricht
nvf
nf
mf
pf
rvf
mf
n
n
n
v
v
n
v
einen Roman den zu schreiben Maria verspricht

(8.50)

In (8.50), the head *schreiben* of the pied piped VP *den zu schreiben* lands in the relative clause Vorfeld (rvf) of the finite verb *verspricht* and the relative pronoun *den* lands in the mf of *schreiben*. Notice that this analysis proceeds analogous to the analysis given in (8.15) for the intraposition example. The difference is that the VP lands in the relative clause Vorfeld rvf here and in the intraposition field if in (8.15).

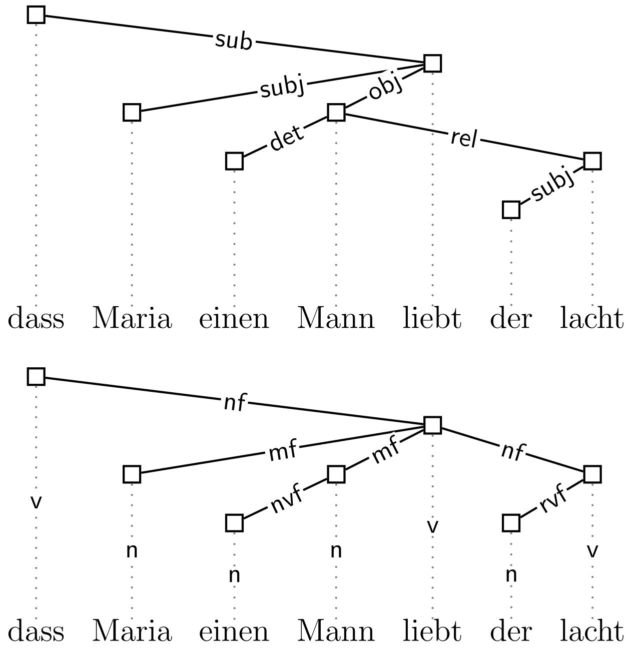
8.4.2 Relative clause extraposition

In German, relative clauses can always be extraposed to the right as in the example below:

(dass)	Maria	einen	Mann	liebt,	der	lacht
(that)	Maria	a	man(acc)	loves,	who	laughs

"(that) Maria loves a man who laughs" (8.51)

In the ID/LP analysis of (8.51), the relative clause *der lacht* climbs up into the nf of the finite verb *liebt*:


sub
subj
obj
det
rel
subj
dass Maria einen Mann liebt der lacht
nf
mf
mf
nf
nvf
rvf
v
n
n
n
v
n
v
dass Maria einen Mann liebt der lacht

(8.52)

8.5 Summary

We gave a number of examples of how the grammar fragment presented in the preceding chapter is able to elegantly handle difficult phenomena in German syntax. We began with scrambling, and turned to constructions which were related to VP-dislocation, including extraposition, intraposition and fronting. Moreover, we showed how to tackle auxiliary flip and related constructions, and gave analyses of constructions associated with relative clauses such as relative clause extraposition and pied piping.

The following chapter provides a formalization of the TDG grammar formalism.

Chapter 9

Formalization

In this chapter, we formalize the notion of a TDG grammar and give formal renditions of the ID, LP and ID/LP principles, largely based on (Duchier 2000) and (Duchier 2001). Making use of the formalizations of the principles, we can formally define the notions of ID, LP and ID/LP analyses.

9.1 TDG grammar

A TDG grammar is characterized by the following 8-tuple:

$$TDG \quad = \quad (\mathcal{R}, \mathcal{F}_{\mathsf{E}}, \mathcal{F}_{\mathsf{N}}, \prec, \mathcal{E}, \mathcal{A}, (\Gamma_\rho), (\Gamma_f)) \tag{9.1}$$

where $\mathcal{R}$ is a set of grammatical roles, $\mathcal{F}_{\mathsf{E}}$ the set of topological fields, $\mathcal{F}_{\mathsf{N}}$ the set of node labels and $\prec$ is a total order on $\mathcal{F} = \mathcal{F}_{\mathsf{E}} \uplus \mathcal{F}_{\mathsf{N}}$. $\mathcal{E}$ is the set of lexical entries and $\mathcal{A}$ the set of lexical attributes. (Γ_ρ) and (Γ_f) denote the sets of ID and LP edge constraints respectively.

For each TDG grammar, $\mathcal{A}$ includes at least the following lexical attributes:

$$\mathcal{A} \quad = \quad \{\mathsf{labels}_{\mathrm{ID}}, \mathsf{valency}_{\mathrm{ID}}, \mathsf{labels}_{\mathrm{LP}}, \mathsf{labels}_{\mathrm{N}}, \mathsf{valency}_{\mathrm{LP}}, \mathsf{blocks}\} \tag{9.2}$$

The definitions of the lexical attributes and their corresponding lattices remain as in chapter 6.

We turn now to the principles restricting the number of admissible ID, LP and ID/LP analyses, and start with the ID principles.

9.2 ID principles

In this section, we formalize the set of ID principles. We begin with formalizing the general principles and then turn to the lexicalized principles.

9.2.1 General principles

The set of general principles in the set of ID principles includes only the treeness principle.

Treeness principle

Principle 9.1 *Each analysis is a tree.*

The starting point for our formalization is an axiomatization of finite labeled graphs. From this set of finite labeled graphs we then characterize the set of finite labeled graphs which are also trees.

We assume an infinite set of nodes $\mathcal{V}$ and a finite set of edge labels $\mathcal{L}$. A directed labeled edge is an element of $\mathcal{V} \times \mathcal{V} \times \mathcal{L}$. $\mathbf{G}(\mathcal{V}, \mathcal{L})$ is the set of finite labeled graphs $G = (V, E)$ formed from a finite set of nodes $V \subseteq \mathcal{V}$ and the finite set of edges $E \subseteq V \times V \times \mathcal{L}$. Note that since E is a set, we only consider graphs without duplicate edges. $\mathbf{G}(V, \mathcal{L})$ is the set of finite graphs in $\mathbf{G}(\mathcal{V}, \mathcal{L})$ with node set V.

We write $w{-}\ell{\rightarrow}w'$ for a labeled edge (w, w', ℓ). Given a graph $G = (V, E)$, we write $w{-}\ell{\rightarrow}_G w'$ for $w{-}\ell{\rightarrow}w' \in E$, and we define the successor relation $\rightarrow_G$ as:

$$\rightarrow_G \;=\; \cup\{-\ell\rightarrow_G \mid \ell \in \mathcal{L}\} \tag{9.3}$$

$w\rightarrow_G^+ w'$ is the transitive closure and $w\rightarrow_G^* w'$ the reflexive transitive closure of $\rightarrow_G$. We define $\rightarrow_G^+$ as being the smallest relation such that:

$$\begin{array}{rcl} w\rightarrow_G w' & \Rightarrow & w\rightarrow_G^+ w' \\ w\rightarrow_G^+ w' \;\wedge\; w'\rightarrow_G^+ w'' & \Rightarrow & w\rightarrow_G^+ w'' \end{array} \tag{9.4}$$

and $\rightarrow_G^*$ as being the smallest relation such that:

$$\begin{array}{rcl} & & w\rightarrow_G^* w \\ w\rightarrow_G w' & \Rightarrow & w\rightarrow_G^* w' \\ w\rightarrow_G^* w' \;\wedge\; w'\rightarrow_G^* w'' & \Rightarrow & w\rightarrow_G^* w'' \end{array} \tag{9.5}$$

The edges of a graph G induce the functions ℓ_G for each $\ell \in \mathcal{L}$, $\mathsf{daughters}_G$, $\mathsf{mothers}_G$, down_G, eqdown_G, up_G and equp_G of type $V \to 2^V$ and roots_G of type 2^V. We define these functions below.

$\ell_G(w)$ denotes the set of ℓ-daughters w' of a node w whose incoming edge is labeled with ℓ:

$$\ell_G(w) = \{w' \mid w{-}\ell{\rightarrow}_G w'\} \tag{9.6}$$

$\mathsf{daughters}_G(w)$ denotes the set of daughters of a node w:

$$\mathsf{daughters}_G(w) = \{w' \mid w{\rightarrow}_G w'\} \tag{9.7}$$

$\mathsf{mothers}_G(w')$ denotes the set of mothers of a node w':

$$\mathsf{mothers}_G(w') = \{w \mid w{\rightarrow}_G w'\} \tag{9.8}$$

$\mathsf{down}_G(w)$ denotes the set of nodes below a node w:

$$\mathsf{down}_G(w) = \{w' \mid w{\rightarrow}^+_G w'\} \tag{9.9}$$

$\mathsf{eqdown}_G(w)$ denotes the set of nodes equal or below w:

$$\mathsf{eqdown}_G(w) = \{w' \mid w{\rightarrow}^*_G w'\} \tag{9.10}$$

$\mathsf{up}_G(w')$ denotes the set of nodes above w':

$$\mathsf{up}_G(w') = \{w \mid w{\rightarrow}^+_G w'\} \tag{9.11}$$

$\mathsf{equp}_G(w')$ denotes the set of nodes equal or above w':

$$\mathsf{equp}_G(w') = \{w \mid w{\rightarrow}^*_G w'\} \tag{9.12}$$

roots_G denotes the set of roots:

$$\mathsf{roots}_G = \{w' \mid \neg\exists w : w{\rightarrow}_G w'\} \tag{9.13}$$

A finite labeled graph $G = (V, \mathcal{L})$ is also a finite labeled tree if and only if it satisfies the following three treeness conditions:

1. Each node has at most one incoming edge.

2. There is precisely one node (the root) with no incoming edge.
3. There are no cycles.

We formalize these conditions using the functions defined on G above. The first treeness condition requires that each node has at most one incoming edge: for any node w'', there exists at most one $\ell \in \mathcal{L}$ and one $w \in V$ such that $w'' \in \ell_G(w)$, or, equivalently:

$$\forall \ell, \ell' \in \mathcal{L} : \forall w, w' \in V \quad : \quad (\ell \neq \ell' \vee w \neq w') \;\Rightarrow\; \ell_G(w) \parallel \ell'_G(w') \tag{9.14}$$

where $\parallel$ expresses disjointness.

The second treeness condition states that there must be one unique root. That is, the cardinality of the set of roots of G is 1:

$$|\mathsf{roots}_G| = 1 \tag{9.15}$$

Finally, the third treeness condition forbids cycles, i.e. w must never be a successor of itself:

$$\forall w \in V \quad : \quad w \notin \mathsf{down}_G(w) \tag{9.16}$$

$\mathbf{G}_{\mathsf{T}}(\mathcal{V}, \mathcal{L})$ is the subset of $\mathbf{G}(\mathcal{V}, \mathcal{L})$ satisfying conditions (9.14), (9.15) and (9.16). $\mathbf{G}_{\mathsf{T}}(V, \mathcal{L})$ is the set of trees in $\mathbf{G}_{\mathsf{T}}(\mathcal{V}, \mathcal{L})$ whose node set is V.

9.2.2 Lexicalized principles

We turn now to the formalization of the lexicalized ID principles. We define lexicalized principles with respect to a lexicon $(\mathcal{E}, \mathcal{A})$ consisting of a finite set $\mathcal{E}$ of lexical entries and a finite set $\mathcal{A}$ of lexical attributes. Recall that a lexical entry is a pair (s, e_t) of a string s from the set *Strs* of strings and a lexical type e_t from the set of lexical types $\mathcal{E}_t$:

$$\mathcal{E} \quad \subseteq \quad \mathit{Strs} \times \mathcal{E}_t \tag{9.17}$$

Lexical attributes $\alpha \in \mathcal{A}$ are functions $\alpha : \mathcal{E}_t \to D_\alpha$ mapping lexical types to values in some domain D_α. For each lexical attribute $\alpha \in \mathcal{A}$ with type $\alpha : \mathcal{E}_t \to D_\alpha$, we introduce the overloaded function $\alpha : V \to D_\alpha$ defined as:

$$\forall w \in V \quad : \quad \alpha(w) = \alpha(\pi_2(\varepsilon(w))) \tag{9.18}$$

where $\pi_2(\varepsilon(w))$ denotes the second projection of $\varepsilon(w)$.

Given a lexicon $(\mathcal{E}, \mathcal{A})$, a graph $(V, E) \in \mathbf{G}(V, \mathcal{L})$ and a lexical assignment $\varepsilon : V \to \mathcal{E}$ of lexical entries to nodes, we call (V, E, ε) an attributed graph. We state all lexicalized principles with respect to attributed graphs.

Accepted edge labels principle

Principle 9.2 *Each node must accept the label of its incoming edge if any.*

To restrict the label of a node's incoming edge, we propose the lexical attribute $\mathsf{labels} : \mathcal{E}_t \to 2^{\mathcal{L}}$ mapping lexical types to sets of edge labels. We define the accepted edge labels principle as follows:

$$w{-}\ell{\to}_G w' \in E \quad \Rightarrow \quad \ell \in \mathsf{labels}(w') \tag{9.19}$$

and we write $\mathbf{G}_{\mathsf{E}}(V, \mathcal{L}, \mathcal{E}, \mathcal{A})$ for the set of attributed graphs $G = (V, E, \varepsilon)$ that are well-formed under the accepted edge labels principle and lexicon $(\mathcal{E}, \mathcal{A})$.

Valency principle

Principle 9.3 *Each node's outgoing edges must precisely fulfill the node's valency.*

Symmetrically to the accepted edge labels principle, the valency principle states constraints on a node's outgoing edges. We define the valency principle with respect to a language of *valency specifications* v. The language of valency specifications for edge label set $\mathcal{L}$ is $v(\mathcal{L})$. A valency specification $\ell' \in v(\mathcal{L})$ is defined by the following abstract syntax:

$$\ell' \quad ::= \quad \ell \mid \ell? \mid \ell* \tag{9.20}$$

We pose the function $\mathsf{valency} : \mathcal{E}_t \to 2^{v(\mathcal{L})}$ mapping lexical types to sets of valency specifications. Writing $\ell(w)$ for the set of dependents of node w with edge label ℓ, we formalize the valency principle as follows:

$$\begin{array}{rcl} \ell \in \mathsf{valency}(w) & \Rightarrow & |\ell(w)| = 1 \\ \ell? \in \mathsf{valency}(w) & \Rightarrow & |\ell(w)| \leq 1 \\ \ell* \in \mathsf{valency}(w) & \Rightarrow & |\ell(w)| \geq 0 \\ \text{otherwise} & \Rightarrow & |\ell(w)| = 0 \end{array} \tag{9.21}$$

and we write $\mathbf{G}_{\mathsf{V}}(V, \mathcal{L}, \mathcal{E}, \mathcal{A})$ for the set of attributed graphs $G = (V, E, \varepsilon)$ that are well-formed under the valency principle and lexicon $(\mathcal{E}, \mathcal{A})$.

Edge constraints

Principle 9.4 *Each edge must be licensed by the corresponding edge constraint.*

Edge constraints are a family (Γ_ℓ) of binary predicates indexed by edge labels $\ell \in \mathcal{L}$. In order for an edge $w{-}\ell{\rightarrow}_G w'$ to be licensed, $\Gamma_\ell(w, w')$ must be satisfied.

We express edge constraints in a constraint language whose abstract syntax is depicted in (9.22). w and w' are variables ranging over nodes, a denotes an arbitrary element of a domain and D an arbitrary finite domain. α is a lexical attribute in $\mathcal{A}$. A binary predicate Γ_ℓ must be of the form $\lambda w, w' \cdot C$, i.e. a P-expression:

$$\begin{array}{lcl} E & ::= & a \mid D \mid \alpha(w) \mid \\ C & ::= & C \wedge C' \mid E = E' \mid E \neq E' \mid E \in E' \mid E \notin E' \mid E \subseteq E' \mid E \parallel E' \\ P & ::= & \lambda w, w' \cdot C \end{array} \tag{9.22}$$

We classify edge constraints under the set of lexicalized principles since they are usually defined with respect to lexical attributes. An example is our treatment of agreement illustrated in appendix A.

Here is the definition of the edge constraints principle:

$$w{-}\ell{\rightarrow}_G w' \in E \quad \Rightarrow \quad G \models \Gamma_\ell(w, w') \tag{9.23}$$

where $G \models \Gamma_\ell(w, w')$ means that G satisfies $\Gamma_\ell(w, w')$ and satisfaction is defined in the obvious Tarskian fashion.

We write $\mathbf{G}_{\mathsf{C}}(V, \mathcal{L}, \mathcal{E}, \mathcal{A}, (\Gamma_\ell))$ for the set of attributed graphs $G = (V, E, \varepsilon)$ that are well-formed under lexicon $(\mathcal{E}, \mathcal{A})$ and edge constraints (Γ_ℓ).

9.2.3 ID analyses

ID analyses are subject to the general treeness principle and the lexicalized principles of accepted edge labels, valency and edge constraints. Thus, we define the set of well-formed ID analyses as follows:

$$\begin{array}{l} \mathbf{G}_{\mathsf{T}}(V, \mathcal{R}) \cap \\ \mathbf{G}_{\mathsf{E}}(V, \mathcal{R}, \mathcal{E}, \mathcal{A}) \ \cap \ \mathbf{G}_{\mathsf{V}}(V, \mathcal{R}, \mathcal{E}, \mathcal{A}) \ \cap \ \mathbf{G}_{\mathsf{C}}(V, \mathcal{R}, \mathcal{E}, \mathcal{A}, (\Gamma_\rho)) \end{array} \tag{9.24}$$

where $\mathcal{R}$ is the finite set of grammatical roles ρ. An ID analysis is a tuple $(V, E_{\text{ID}}, \varepsilon)$ where E_{ID} is the set of ID tree edges and ε a lexical assignment.

9.3 LP principles

In this section, we formalize the set of LP principles. Again, we begin with the general principles and then turn to the lexicalized principles.

9.3.1 General principles

The general LP principles include the treeness principle, the order principle and the projectivity principle.

Treeness principle

The treeness principle remains as in section 9.2.1.

Order principle

Principle 9.5 *Each analysis must be well-ordered.*

We distinguish a set $\mathcal{L}_{\mathsf{E}}$ of edge labels ℓ and $\mathcal{L}_{\mathsf{N}}$ of node labels n and pose a total order $\prec$ on the set $\mathcal{L}_{\mathsf{E}} \uplus \mathcal{L}_{\mathsf{N}}$ of edge and node labels. Given a graph (V, E), a lexical assignment ε, a total order $<$ on V and a node label assignment $\mathsf{label}_{\mathrm{N}} : V \to \mathcal{F}_{\mathsf{N}}$, we say that $G = (V, E, \varepsilon, <, \mathsf{label}_{\mathrm{N}})$ is a well-ordered graph iff it satisfies the following conditions:

$$w{-}\ell_1{\rightarrow}_G w_1 \;\wedge\; w{-}\ell_2{\rightarrow}_G w_2 \;\wedge\; \ell_1 \prec \ell_2 \;\Rightarrow\; w_1 < w_2 \tag{9.25}$$

$$w_1{\rightarrow}^*_G w'_1 \;\wedge\; w_2{\rightarrow}^*_G w'_2 \;\wedge\; w_1 < w_2 \;\Rightarrow\; w'_1 < w'_2 \tag{9.26}$$

$$w{-}\ell{\rightarrow}_G w_1 \;\wedge\; \ell \prec \mathsf{label}_{\mathrm{N}}(w) \;\Rightarrow\; w_1 < w \tag{9.27}$$

$$w{-}\ell{\rightarrow}_G w_1 \;\wedge\; \mathsf{label}_{\mathrm{N}}(w) \prec \ell \;\Rightarrow\; w < w_1 \tag{9.28}$$

(9.25) orders the daughters of a node w with respect to each other. (9.26) states that if w_1 precedes w_2, then all nodes in the yield of w_1 must also precede all nodes in the yield of w_2. (9.27) and (9.28) orders mothers with respect their daughters using node labels.

We write $\mathbf{G}_{\mathsf{O}}(V, \mathcal{L}_{\mathsf{E}}, \mathcal{L}_{\mathsf{N}}, \mathcal{E}, \mathcal{A}, \prec)$ for the set of graphs $G = (V, E, \varepsilon, <, \mathsf{label}_{\mathrm{N}})$ which are well-formed under the order principle, lexicon $(\mathcal{E}, \mathcal{A})$ and the total order $\prec$.

Projectivity principle

Principle 9.6 *Each analysis must be projective.*

The projectivity principle requires ordered graphs to be *projective*, i.e. the yield of each node must cover a contiguous substring. We formalize the projectivity principle as:

$$\forall w \in V : \quad \mathsf{convex}(\mathsf{eqdown}(w)) \tag{9.29}$$

The declarative semantics of $\mathsf{convex}(S)$ is that for all $w_1, w_2 \in S$, if $w_1 < w_2$, then for all w such that $w_1 < w < w_2$, also $w \in S$. That is, $\mathsf{convex}(S)$ requires that the interval S contains no holes.

We write $\mathbf{G}_{\mathsf{P}}(V, \mathcal{L}_{\mathsf{E}})$ for the set of ordered graphs $G = (V, E, <)$ satisfying the projectivity principle.

9.3.2 Lexicalized principles

The set of lexicalized LP principles includes the accepted edge labels principle, the valency principle and the accepted node labels principle.

Accepted edge labels principle

The accepted edge labels principle remains as in section 9.2.2.

Valency principle

The valency principle remains as in section 9.2.2.

Accepted node labels principle

Principle 9.7 *Each node must accept its node label.*

This principle makes use of the lexical attribute $\mathsf{labels}_{\mathrm{N}} : \mathcal{E}_t \to 2^{\mathcal{L}_{\mathsf{N}}}$ and the node label assignment $\mathsf{label}_{\mathrm{N}} : V \to \mathcal{L}_{\mathsf{N}}$. $\mathsf{label}_{\mathrm{N}}(w)$ assigns to w precisely one node label from the set of node labels $\mathsf{labels}_{\mathrm{N}}(w)$:

$$\forall w \in V : \quad \mathsf{label}_{\mathrm{N}}(w) \in \mathsf{labels}_{\mathrm{N}}(w) \tag{9.30}$$

We write $\mathbf{G}_{\mathsf{N}}(V, \mathcal{L}_{\mathsf{E}}, \mathcal{L}_{\mathsf{N}}, \mathcal{E}, \mathcal{A})$ for the set of attributed graphs $G = (V, E, \varepsilon, \mathsf{label}_{\mathrm{N}})$ that are well-formed under the accepted node labels principle and lexicon $(\mathcal{E}, \mathcal{A})$.

Edge constraints

The edge constraints principle remains as in section 9.2.2.

9.3.3 LP analyses

LP analyses are subject to the general treeness principle, the order principle and the projectivity principle. Each LP analysis must also satisfy the lexicalized principles of accepted edge labels, valency, accepted node labels and edge constraints. Thus, we define the set of well-formed LP analyses as follows:

$$\begin{array}{l} \mathbf{G}_{\mathsf{T}}(V, \mathcal{F}_{\mathsf{E}}) \;\cap\; \mathbf{G}_{\mathsf{O}}(V, \mathcal{F}_{\mathsf{E}}, \mathcal{F}_{\mathsf{N}}, \mathcal{E}, \mathcal{A}, \prec) \;\cap\; \mathbf{G}_{\mathsf{P}}(V, \mathcal{F}_{\mathsf{E}}) \cap \\ \mathbf{G}_{\mathsf{E}}(V, \mathcal{F}_{\mathsf{E}}, \mathcal{E}, \mathcal{A}) \;\cap\; \mathbf{G}_{\mathsf{V}}(V, \mathcal{F}_{\mathsf{E}}, \mathcal{E}, \mathcal{A}) \;\cap\; \mathbf{G}_{\mathsf{N}}(V, \mathcal{F}_{\mathsf{E}}, \mathcal{F}_{\mathsf{N}}, \mathcal{E}, \mathcal{A}) \cap \\ \mathbf{G}_{\mathsf{C}}(V, \mathcal{F}_{\mathsf{E}}, \mathcal{E}, \mathcal{A}, (\Gamma_f)) \end{array} \tag{9.31}$$

where $\mathcal{F}_{\mathsf{E}}$ is the set of LP edge labels and $\mathcal{F}_{\mathsf{N}}$ the set of LP node labels. An LP analysis is a tuple $(V, E_{\text{LP}}, \varepsilon, <, \mathsf{label}_{\text{N}})$ where E_{LP} is the set of LP tree edges, ε a lexical assignment, $<$ a total order on V and $\mathsf{label}_{\text{N}}$ a node label assignment.

9.4 ID/LP principles

A TDG ID/LP analysis is subject to general and lexicalized ID/LP principles.

9.4.1 General principles

The set of general ID/LP principles includes the climbing principle and the subtrees principle.

Climbing principle

Principle 9.8 *A node must land on a transitive head.*

We formalize the *climbing principle* by stating that the set of topological heads of node w is a subset of the set of nodes equal or above w in the ID tree:[1]

$$\forall w \in V : \quad \mathsf{mothers}_{\text{LP}}(w) \subseteq \mathsf{equp}_{\text{ID}}(w) \tag{9.32}$$

[1] The ID- and LP-subscripts indicate whether the respective function is defined on the ID or the LP tree.

Subtrees principle

Principle 9.9 *A node must land on, or climb higher than its syntactic head.*

We formalize the *subtrees principle* by requiring that the set of nodes above w in the LP tree is a subset of the set of nodes equal or above the syntactic head in the LP tree:

$$\forall w \in V : \quad \mathsf{up}_{\text{LP}}(w) \ \subseteq \ \cup\{\mathsf{equp}_{\text{LP}}(w') \mid w' \in \mathsf{mothers}_{\text{ID}}(w)\} \tag{9.33}$$

9.4.2 Lexicalized principles

The set of lexicalized ID/LP principles includes only the barriers principle.

Barriers principle

Principle 9.10 *A node may not climb through a barrier.*

The *barriers principle* makes use of the lexical attribute $\mathsf{blocks} : \mathcal{E}_t \to 2^{\mathcal{R}}$ mapping lexical types to sets of roles. A node w is blocked by another node w' if w has role ρ and $\rho \in \mathsf{blocks}(w')$. Here is a formalization of the barriers principle:

The nodes which w must climb through are all nodes between w and the topological head of w:

$$\mathsf{through}(w) = \quad \mathsf{up}_{\text{ID}}(w) \ \cap \ \cup\{\mathsf{down}_{\text{ID}}(w') \mid w' \in \mathsf{mothers}_{\text{LP}}(w)\} \tag{9.34}$$

We pose the function $\mathsf{label}_{\text{ID}} : V \to \mathcal{R}$ which assigns to each node the label of its incoming edge:

$$\forall w \in V : \quad \mathsf{label}_{\text{ID}}(w) \in \mathsf{labels}_{\text{ID}}(w) \tag{9.35}$$

Whenever the label of w's incoming edge is ρ, then ρ must not be blocked by any of the nodes in $\mathsf{through}(w)$:

$$\forall \rho \in \mathcal{R} : \forall w \in V : \quad \rho = \mathsf{label}_{\text{ID}}(w) \ \Rightarrow \ \rho \notin \cup\{\mathsf{blocks}(w') \mid w' \in \mathsf{through}(w)\} \tag{9.36}$$

9.4.3 ID/LP analyses

An ID/LP analysis is a tuple $(V, E_{\mathrm{ID}}, E_{\mathrm{LP}}, \varepsilon, <, \mathsf{label}_{\mathrm{N}})$ such that $(V, E_{\mathrm{ID}}, \varepsilon)$ is a well-formed ID analysis and $(V, E_{\mathrm{LP}}, \varepsilon, <, \mathsf{label}_{\mathrm{N}})$ is a well-formed LP analysis and the ID/LP principles are satisfied.

9.5 Summary

We provided a formalization of the TDG grammar formalism. We started out with defining the notion of a TDG grammar, and continued with stating the principles for admissible ID, LP and ID/LP analyses. Starting from the formalizations of the principles, we could define the notions of ID, LP and ID/LP analyses. An ID/LP analysis is only licensed if all principles are satisfied.

The following chapter sketches the parser implementation done for the thesis.

Chapter 10

Implementation

Using a reduction into a constraint satisfaction problem (CSP) stated in terms of finite sets described in (Duchier 2000), the formalization presented in the previous chapter yields an efficient parser implementation. In this chapter, we review the TDG parser implementation done in the context of this thesis. Grammars for the parser can be written using a concrete grammar specification language, and we have also created a graphical user interface (GUI) to facilitate grammar development and grammar debugging.

10.1 Parser

In the formalization of TDG in the previous chapter, most of the principles were stated in terms of sets. As an example, consider the accepted edge labels principle:

$$w{-}\ell{\rightarrow}_G w' \in E \quad \Rightarrow \quad \ell \in \mathsf{labels}(w') \tag{10.1}$$

which is stated on the sets E of edges and $\mathsf{labels}(w')$ of accepted edge labels for node w'. However, there are also principles, such as the order principle, which for simplicity we formalized in a more classical fashion. Their reformulation as set constraints is beyond the scope of this thesis, but follows directly from (Duchier 2000, Duchier 2001). Using his ideas we obtain a characterization of valid ID/LP analyses as solutions of a constraint satisfaction problem (CSP) which is entirely stated in terms of set variables. The central importance given to sets, in both the formalization and the implementation, is a novel and distinguishing aspect of our approach.

The Mozart-Oz programming language (*Mozart* 1998) provides all the means necessary to turn the CSP into a constraint program. These means include a sophisticated constraint system with support for finite set variables. Consequently, the CSP can be converted straightforwardly into an Oz program. Parsing amounts to searching for solutions of the CSP which describes the valid ID/LP analyses for a given input. This process follows the *propagate and distribute* paradigm of constraint programming and alternates steps of deterministic inference through constraint propagation with steps of non-deterministic choice variously called 'distribution' or 'labeling'. With the help of the *selection constraint* developed in (Duchier 1999) and (Duchier 2000), we achieve very strong propagation. In fact, in many cases no search at all is necessary, in spite of substantial lexical and structural ambiguity.

The parser is highly efficient. Although parsing with TDG is NP-complete (Alexander Koller, pc), it seems to be polynomial for practical applications. As an example, we can find the first parse for a 10 word sentence in 200ms, for 20 words in 700ms and for 50 words in 5 seconds on a 700MHz x86-machine. Notice that we attain this high degree of efficiency without any kind of optimization.

The parser implementation is completely modular and allows to add additional principles using a *plugin system.* One example of such an additional principle is the *relative clause principle* described in appendix A.

To facilitate grammar debugging, the parser can be switched into *generate mode.* In generate mode, it does not parse a sequence of words but tries to find all ID/LP analyses for a bag of words. This can be used to check whether the grammar licenses undesired linearizations. Another useful feature for grammar debugging is the possibility to individually toggle each of the ID, LP and ID/LP principles on and off.

The latest version of the parser (including the GUI) can be found in the *MOzart Global User Library* (`http://www.mozart-oz.org/mogul`)

10.2 Concrete grammar specification language

Grammars for the TDG parser are written in a concrete grammar specification language. The specification language has two advantages over writing grammars directly in Oz:

- it abstracts away from the syntax of Mozart-Oz
- it specifically supports our formal presentation of TDG
- it is statically typed

A grammar specification written in the concrete grammar specification language consists of a sequence of *definition sections* introduced by keywords. We display a list of the sections in Table 10.1.

section	defines
`deftypes`	types
`defentry`	lexical attributes
`deforder`	total order on $\mathcal{F} = \mathcal{F}_{\mathsf{E}} \uplus \mathcal{F}_{\mathsf{N}}$
`defattributes`	node attributes
`defnode`	node constraints
`defedges id`	ID edge constraints
`defedges lp`	LP edge constraints
`defdistribute`	distribution strategy
`defword`	lexical hierarchy and entries

Table 10.1: Concrete grammar specification language: definition sections

10.2.1 Constants and variables

In the concrete grammar specification language, we distinguish *constants Const* and *variables Var*:

$$\begin{array}{rcl} \mathit{Const} & ::= & \texttt{[a-z][A-Za-z0-9\_-]*} \\ & & \texttt{'([\^{}\textbackslash n']|\textbackslash\textbackslash.)*'} \\ & & \texttt{[0-9]*} \end{array} \tag{10.2}$$

$$\mathit{Var} \quad ::= \quad \texttt{[A-Z][A-Za-z0-9\_-]*} \tag{10.3}$$

Variables must start with an upper-case letter [A−Z], while constants must start with a lower-case letter [a−z]. Subsequent characters can either be upper case letters, lower case letters, digits (0−9), underscores (_) or dashes (−).[1] Additionally, a constant may also be quoted '([^\n']|\\.)*' so that they may contain unusual characters or in order to distinguish them from keywords. Integers are also constants.

Examples

Here are some example variables:

$$\texttt{ROLE EXT TesT Hello} \tag{10.4}$$

[1] The asterisks following the closing square brackets indicate that constants and variables may consist of an arbitrarily long sequence of these characters.

And below, we show some example constants:

$$\texttt{rOLE}\quad\texttt{subj}\quad\texttt{'def'}\quad\texttt{4711} \tag{10.5}$$

10.2.2 Types

Types can be either sets or lattices defined on sets. We define types in the `deftypes`-section:

$$\begin{array}{lccc} \texttt{deftypes}\,\{ & & & \\ & \mathit{Var}_1 & : & \mathit{Typeexpr}_1 \\ & & \ldots & \\ & \mathit{Var}_n & : & \mathit{Typeexpr}_n \\ & \} & & \end{array} \tag{10.6}$$

Expressions of type *Typeexpr* are defined as follows:

$$\begin{array}{lcl} \mathit{Typeexpr} & ::= & \mathit{Typeexpr2}_1 * \ \ldots\ * \mathit{Typeexpr2}_n \\ & \mid & \mathit{Typeexpr2} \end{array} \tag{10.7}$$

A *Typeexpr* is either a cartesian product where the individual factors $\mathit{Typeexpr2}_i$ are separated by asterisks[2] or just an expression of type *Typeexpr2*. *Typeexpr2* is defined as follows:

$$\begin{array}{lcl} \mathit{Typeexpr2} & ::= & \mathit{Var} \\ & \mid & \{\mathit{Const}_1\ \ldots\ \mathit{Const}_n\} \\ & \mid & \mathit{Typeexpr2}\ \texttt{set} \\ & \mid & \mathit{Typeexpr2}\ \texttt{aset} \\ & \mid & \mathit{Typeexpr2}\ \texttt{valency} \end{array} \tag{10.8}$$

where `set` corresponds to lattice type $\mathcal{L}_\cap$, `aset` to $\mathcal{L}_\cup$ and `valency` to $\mathcal{L}_\cup \circ \upsilon$.

Example

Here is an example where we define the types `ROLE`, `EXT` and `INT` and, in addition, the type `AGR` which is defined as the cartesian product of the types `PERSON`, `GENDER`,

[2] Notice that the asterisks here do not belong to the metalanguage as in the definition of constants and variables above but to the concrete grammar specification language itself.

`NUMBER`, `DEF` and `CASE`:

$$\begin{array}{rcl}
\multicolumn{3}{l}{\texttt{deftypes}\ \{} \\
\texttt{ROLE} & : & \{\texttt{adj adv} \ldots \texttt{zu}\} \\
\texttt{EXT} & : & \{\texttt{if mf} \ldots \texttt{vxf}\} \\
\texttt{INT} & : & \{\texttt{n v12 v}\} \\
& & \\
\texttt{PERSON} & : & \{\texttt{1 2 3}\} \\
\texttt{GENDER} & : & \{\texttt{masc fem neut}\} \\
\texttt{NUMBER} & : & \{\texttt{sg pl}\} \\
\texttt{DEF} & : & \{\texttt{def indef undef}\} \\
\texttt{CASE} & : & \{\texttt{nom gen dat acc}\} \\
\texttt{AGR} & : & \texttt{PERSON} * \texttt{GENDER} * \texttt{NUMBER} * \texttt{DEF} * \texttt{CASE} \\
\}
\end{array} \tag{10.9}$$

`ROLE` corresponds to the set $\mathcal{R}$ of grammatical roles, `EXT` to the set of topological fields $\mathcal{F}_{\mathsf{E}}$ and `INT` to the set of node labels $\mathcal{F}_{\mathsf{N}}$. These three types *must* be defined in each grammar specification. `AGR` corresponds to the set *Agr* defined in (A.35) in appendix A where we show how we handle agreement in TDG.

10.2.3 Lexical attributes

In the `defentry` section, we assign lattices to lexical attributes:

$$\begin{array}{l}
\texttt{defentry}\ \{ \\
\qquad \mathit{Typefeat}_1 \\
\qquad \ldots \\
\qquad \mathit{Typefeat}_n \\
\}
\end{array} \tag{10.10}$$

Typefeat is defined as:

$$\mathit{Typefeat} \ ::= \ \mathit{Const} : \mathit{Typeexpr} \tag{10.11}$$

where *Const* is a lexical attribute and *Typeexpr* a lattice.

Example

In chapter 6, we assigned the following lattices to the ID, LP and ID/LP attributes:

$$\begin{bmatrix} \mathsf{labels}_{\mathrm{ID}} & : & \mathcal{L}_{\cap}(\mathcal{R}) \\ \mathsf{valency}_{\mathrm{ID}} & : & \mathcal{L}_{\cup}(\upsilon(\mathcal{R})) \\ \mathsf{labels}_{\mathrm{LP}} & : & \mathcal{L}_{\cap}(\mathcal{F}_{\mathsf{E}}) \\ \mathsf{labels}_{\mathrm{N}} & : & \mathcal{L}_{\cap}(\mathcal{F}_{\mathsf{N}}) \\ \mathsf{valency}_{\mathrm{LP}} & : & \mathcal{L}_{\cup}(\upsilon(\mathcal{F}_{\mathsf{E}})) \\ \mathsf{blocks} & : & \mathcal{L}_{\cup}(\mathcal{R}) \end{bmatrix} \tag{10.12}$$

Below, we show how to do the same in the concrete grammar specification language. In addition, we assign to the new lexical attribute `agrs` the lattice `AGR set`:

```
defentry {
   labelsID  :  ROLE set
  valencyID  :  ROLE valency
   labelsLP  :  EXT set
  labelsINT  :  INT set
  valencyLP  :  EXT valency                  (10.13)
     blocks  :  ROLE aset

       agrs  :  AGR set
          }
```

Here, `labelsID` corresponds to lexical attribute $\mathsf{labels}_{\mathrm{ID}}$, `valencyID` to $\mathsf{valency}_{\mathrm{ID}}$, `labelsLP` to $\mathsf{labels}_{\mathrm{LP}}$, `labelsINT` to $\mathsf{labels}_{\mathrm{N}}$, `valencyLP` to $\mathsf{valency}_{\mathrm{LP}}$ and `blocks` to blocks. All the lexical attributes contained in (10.13) *must* be defined in each grammar specification.

10.2.4 Total order

The total order on $\mathcal{F} = \mathcal{F}_{\mathsf{E}} \uplus \mathcal{F}_{\mathsf{N}}$ is stated in the `deforder`-section:

$$\begin{array}{l} \texttt{deforder}\ \{ \\ \qquad \mathit{Cons}_1\ \ldots\ \mathit{Cons}_n \\ \qquad \} \end{array} \tag{10.14}$$

where the constants Cons_i must either be of type `EXT` or `INT`.

Example

As an example, we repeat the total order on the set $\mathcal{F} = \mathcal{F}_{\mathsf{E}} \uplus \mathcal{F}_{\mathsf{N}}$ defined for the grammar fragment in chapter 7:

$$\mathsf{nvf} \prec \mathsf{nmf} \prec \mathsf{n} \prec \mathsf{nxf} \prec \mathsf{vf} \prec \mathsf{v12} \prec \mathsf{rvf} \prec \mathsf{if} \prec \mathsf{mf} \prec \mathsf{vcf} \prec \mathsf{pf} \prec \mathsf{v} \prec \mathsf{vxf} \prec \mathsf{nf} \tag{10.15}$$

And below, we show how to state this total order using the concrete grammar specification language:

```
deforder {
            nvf nmf n nxf vf v12 rvf if mf vcf pf v vxf nf
         }
```
(10.16)

10.2.5 Node attributes

We can also define so-called *node attributes*. Node attributes are functions with domain V. We define node attributes in the `defattributes`-section:

$$\begin{array}{ll}\texttt{defattributes}\ \{ & \\ & \mathit{Typefeat}_1 \\ & \ldots \\ & \mathit{Typefeat}_n \\ \} & \end{array} \tag{10.17}$$

Example

Here is an example where we define the node attribute `agr` of type `AGR`:

```
defattributes {
                agr  :  AGR
              }
```
(10.18)

10.2.6 Node constraints

The licensed values of node attributes can be constrained using *node constraints*. We define node constraints in the `defnode`-section:

$$\begin{array}{l} \texttt{defnode}\ \{ \\ \qquad \mathit{Nodecons}_1 \\ \qquad \quad \ldots \\ \qquad \mathit{Nodecons}_n \\ \} \end{array} \tag{10.19}$$

We define expressions of type *Nodecons* as shown below:

$$\begin{array}{rcl} \mathit{Nodecons} & ::= & \mathit{Nodeexpr}_1 = \mathit{Nodeexpr}_2 \\ & \mid & \mathit{Nodeexpr}_1\ \texttt{in}\ \mathit{Nodeexpr}_2 \\ & \mid & \mathit{Nodeexpr}_1\ \texttt{subset}\ \mathit{Nodeexpr}_2 \\ & \mid & \mathit{Nodeexpr}_1\ \texttt{disjoint}\ \mathit{Nodeexpr}_2 \end{array} \tag{10.20}$$

where $=$ denotes equality, `in` denotes $\in$, `subset` denotes $\subseteq$ and `disjoint` denotes $\|$ (disjointness).

Expressions of type *Nodeexpr* are defined as:

$$\begin{array}{rcl} \mathit{Nodeexpr} & ::= & \mathit{Scoreexpr} \\ & \mid & \mathit{Conexpr} \end{array} \tag{10.21}$$

We define *Scoreexpr* as:

$$\begin{array}{rcl} \mathit{Scoreexpr} & ::= & \_[\mathit{Const}] \\ & \mid & \_\mathit{Dot}_1 \ldots \mathit{Dot}_n \end{array} \tag{10.22}$$

where the underscore denotes the current node. $\_[\mathit{Const}]$ denotes the value of node attribute *Const* at the current node.

We define *Dot* as:

$$\mathit{Dot} \ ::= \ .\mathit{Const} \tag{10.23}$$

using expressions of type *Dot* we can access features of so-called *signs* which are Oz record structures introduced for each node by the parser. A sign is defined as follows:

```
                sign(
                  lex  :  o(
                            index  :  Index
                             word  :  Word
                            entry  :  Entry
                            )                          (10.24)
                   id  :  NodeID
                   lp  :  NodeLP
            attribute  :  AttributeRecord
                     )
```

where `Index` is the index of the selected entry, `Word` the string corresponding to the node and `Entry` the selected lexical entry itself. `NodeID` holds the information for the occurrence of the node in the ID tree and `NodeLP` the corresponding information for its occurrence in the LP tree. `AttributeRecord` is a record containing the node attributes. For instance to access the feature `labelsID` of the lexical entry at the current node, we write `_.lex.entry.labelsID`. To access node attribute `agr` at the current node, we can either write `_.attribute.agr` or use the more convenient notation _[`agr`].

We define *Conexpr* as:

$$\begin{array}{rcl} \mathit{Conexpr} & ::= & \mathit{Const} \\ & | & \{\,\} \\ & | & \{\ \mathit{Wild_1}\ \ldots\ \mathit{Wild_n}\ \} \\ & | & \$\ \mathit{Genexpr} \end{array} \tag{10.25}$$

i.e. a *Conexpr* is either a constant (*Const*), the empty set ({ }), a set of wildcard expressions ({ $\mathit{Wild_1}$ … $\mathit{Wild_n}$ }) or a set generator expression ($ *Genexpr*).

We define a wildcard expression *Wild* as:

$$\begin{array}{rcl} \mathit{Wild} & ::= & \mathit{Const} \\ & | & \mathit{Const}? \\ & | & \mathit{Const}* \end{array} \tag{10.26}$$

Wildcard expressions are used to specify the valency features `valencyID` and `valencyLP`.

We define a set generator expression *Genexpr* as:

$$\begin{array}{rcl} \mathit{Genexpr} & ::= & \mathit{Genexpr2} \\ & | & \mathit{Genexpr} \mid \mathit{Genexpr2} \end{array} \tag{10.27}$$

Genexpr2 is defined as follows:

$$\begin{array}{lcl} \mathit{Genexpr2} & ::= & \mathit{Genexpr3} \\ & \mid & \mathit{Genexpr2}\ \&\ \mathit{Genexpr3} \end{array} \tag{10.28}$$

and *Genexpr3* is defined as follows:

$$\begin{array}{lcl} \mathit{Genexpr3} & ::= & (\ \mathit{Genexpr}\ ) \\ & \mid & \mathit{Const} \end{array} \tag{10.29}$$

Set generator expressions are boolean expressions generating values for the corresponding types. For example

$$\$\ \mathtt{3}\ \&\ \mathtt{sg}\ \&\ (\mathtt{nom} \mid \mathtt{acc}) \tag{10.30}$$

generates the set of all agreement tuples that are third person (3), singular (sg) and nominative or accusative case ((nom | acc)).

Example

Here is an example node constraint:

$$\begin{array}{l} \mathtt{defnode}\ \{ \\ \quad \_[\mathtt{agr}]\ \ \mathtt{in}\ \ \_.\mathtt{lex.entry.agrs} \\ \qquad \} \end{array} \tag{10.31}$$

_[agr] denotes the value of node attribute agr and _.lex.entry.agrs the value of the lexical attribute agrs. Notice that this node constraint is the counterpart of the constraint given in (A.40) in appendix A.

10.2.7 ID edge constraints

ID edge constraints are defined in the *defedges id*-section:

$$\begin{array}{l} \mathtt{defedges\ id}\ \{ \\ \qquad\qquad \mathit{Edgedef}_1 \\ \qquad\qquad \quad \ldots \\ \qquad\qquad \mathit{Edgedef}_n \\ \qquad \} \end{array} \tag{10.32}$$

Edgedef is defined as follows:

$$\begin{array}{lll} \textit{Edgedef} & ::= & \\ & & \textit{Lab}\ \{ \\ & & \quad \textit{Edgecons}_1 \\ & & \quad \ldots \\ & & \quad \textit{Edgecons}_n \\ & & \} \end{array} \tag{10.33}$$

Lab is defined as follows:

$$\begin{array}{lll} \textit{Lab} & ::= & \textit{Cons} \\ & | & \texttt{\_\_} \end{array} \tag{10.34}$$

where *Cons* must be of type `ROLE` for ID edge constraints and of type `EXT` for LP edge constraints and `__` is a special symbol matching any edge label.

Edgecons is defined as follows:

$$\begin{array}{lll} \textit{Edgecons} & ::= & \textit{Edgeexpr}_1 = \textit{Edgeexpr}_2 \\ & | & \textit{Edgeexpr}_1\ \texttt{in}\ \textit{Edgeexpr}_2 \\ & | & \textit{Edgeexpr}_1\ \texttt{subset}\ \textit{Edgeexpr}_2 \\ & | & \textit{Edgeexpr}_1\ \texttt{disjoint}\ \textit{Edgeexpr}_2 \end{array} \tag{10.35}$$

Edgeexpr is defined as follows:

$$\begin{array}{lll} \textit{Edgeexpr} & ::= & \textit{Scoreexpr} \\ & | & \textit{Conexpr} \\ & | & \textit{Caretexpr} \\ & | & \texttt{\_\_} \end{array} \tag{10.36}$$

Caretexpr is defined as follows:

$$\begin{array}{lll} \textit{Caretexpr} & ::= & \hat{}\,[\textit{Const}] \\ & | & \hat{}\,\textit{Dot}_1 \ldots \textit{Dot}_n \end{array} \tag{10.37}$$

Example

Here are example ID edge constraints written in the concrete grammar specification language:

```
defedges id {
        subj {
              _[agr] = ^[agr]
              _[agr] in & nom
             }                                  (10.38)
        obj {
              _[agr] in & acc
             }
       }
```

The ID edge constraint for grammatical role `subj` corresponds to the edge constraint given in (A.41) in appendix A and the ID edge constraint for role `vpref` to the one given in (A.42). Notice that we have chosen the notation using underscores (_) and carets (ˆ) to hint at a similarity to LFG: in LFG and also here, the underscore can be read as ‘down’ (this node) and the caret as ‘up’ (mother).

10.2.8 LP edge constraints

LP edge constraints are defined precisely like ID edge constraints except that the section is headed by `defedges lp` instead of `defedges id`:

```
defedges lp {
        Edgedef_1
          ...                                   (10.39)
        Edgedef_n
       }
```

10.2.9 Distribution strategy

The distribution strategy used by the parser can be adjusted in the `defdistribute`-section:

```
defdistribute {
        Scoreexpr_1
          ...                                   (10.40)
        Scoreexpr_n
       }
```

Each $Scoreexpr_i$ expression identifies a feature of the signs which should be determined next using distribution (aka labeling).

Example

The following distribution strategy determines the values of all set variables in a typical TDG grammar:

```
defdistribute {
                 _.id.mothers
                 _.id.daughterSet
                 _.lp.mothers
                 _.lp.daughterSet             (10.41)
                 _.lp.labelINT
                 _.lp.pos
              }
```

In other words, the strategy first determines the ID mother for each sign. After this step of the strategy, every node knows its ID mother or is root. While each node in the ID tree now knows its mother, it may not yet know the label on its incoming edge. The second step takes care of this by determining all daughter sets. After this step, the ID tree is entirely known. The third and fourth steps do the same for the LP tree. The fifth step takes care of the node label assignment, and the last step determines the order (in case we were executing in 'generate' mode).

10.2.10 Lexical hierarchy and entries

Lexical types and lexical entries are both defined using the keyword `defword`:

$$\begin{array}{l} \texttt{defword}\ \mathit{Const_1}\ \ldots\ \mathit{Const_n}\ \{ \\ \qquad \mathit{Featexpr_1} \\ \qquad \ldots \\ \qquad \mathit{Featexpr_n} \\ \} \end{array} \tag{10.42}$$

Featexpr is defined as follows:

$$\mathit{Featexpr} \ ::=\ \mathit{Const} : \mathit{Conexpr} \tag{10.43}$$

Example

As an example of how to specify a lexical type, we repeat the type *t_cnoun_id* for common nouns from chapter 7:

$$t\_cnoun\_id \quad = \quad t\_noun\_id \sqcap \left[\ \mathsf{valency}_{\text{ID}} \ : \ \{\mathsf{det}?, \mathsf{adj}*, \mathsf{genmod}?, \mathsf{pp}?, \mathsf{rel}?\} \ \right] \tag{10.44}$$

Here is how we write it in the concrete grammar specification language:

```
defword  t_cnoun_id  t_noun_id  {
                          valencyID  :  {det? adj * genmod? pp? rel?}
                                     }
```

(10.45)

That is, the first argument of `defword` is the name of the lexical type to be defined (here: `t_cnoun_id`), followed by an arbitrary number of lexical types from which to inherit (here: only `t_noun_id`). In curly brackets, the definition of another lexical type is given from which the defined type inherits.

Lexical entries are defined identically. As an example, we repeat here the lexical entries for the past participle *gegeben* in chapter 7:

$$(gegeben \ , \ t\_vpp\_id \sqcap t\_ditr\_id \sqcap t\_can\_lp) \tag{10.46}$$

$$(gegeben \ , \ t\_vpp\_id \sqcap t\_ditr\_id \sqcap t\_front\_lp) \tag{10.47}$$

In the concrete grammar specification language, we obtain these lexical entries as follows:

```
defword  gegeben  t_vpp_id  t_ditr_id  t_can_lp  { }
defword  gegeben  t_vpp_id  t_ditr_id  t_front_lp  { }
```

(10.48)

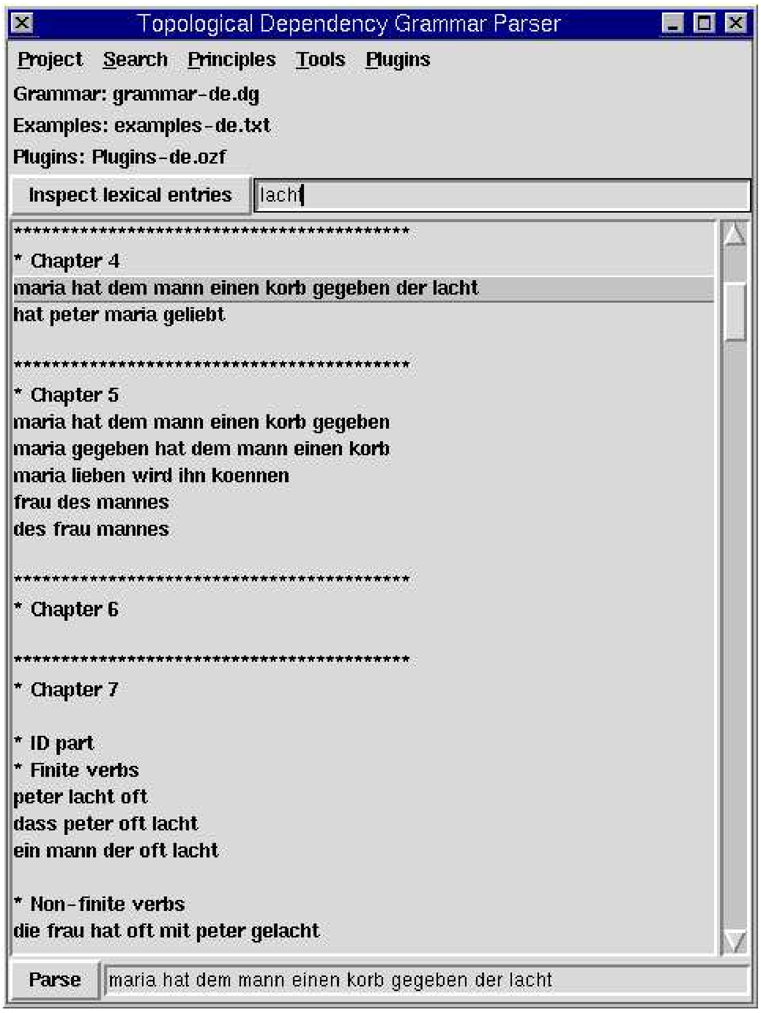
I.e. we define the lexical entry `gegeben` which inherits from the lexical types `t_vpp_id` and `t_ditr_id`. `gegeben` also inherits from `t_can_lp` (upper entry) and `t_front_lp` (lower entry).

10.3 Graphical user interface

We have also implemented a graphical user interface (GUI) to facilitate grammar development and grammar debugging. We describe the GUI in the following.

10.3.1 Main window

We display the main window of the graphical user interface (GUI) in Figure 10.1. The main window is divided into five parts. The first part is the topmost row of pulldown-menus, consisting of *Project*, *Search*, *Principles*, *Tools* and *Plugins*.



```
Grammar: grammar-de.dg
Examples: examples-de.txt
Plugins: Plugins-de.ozf
```

```
lacht
```

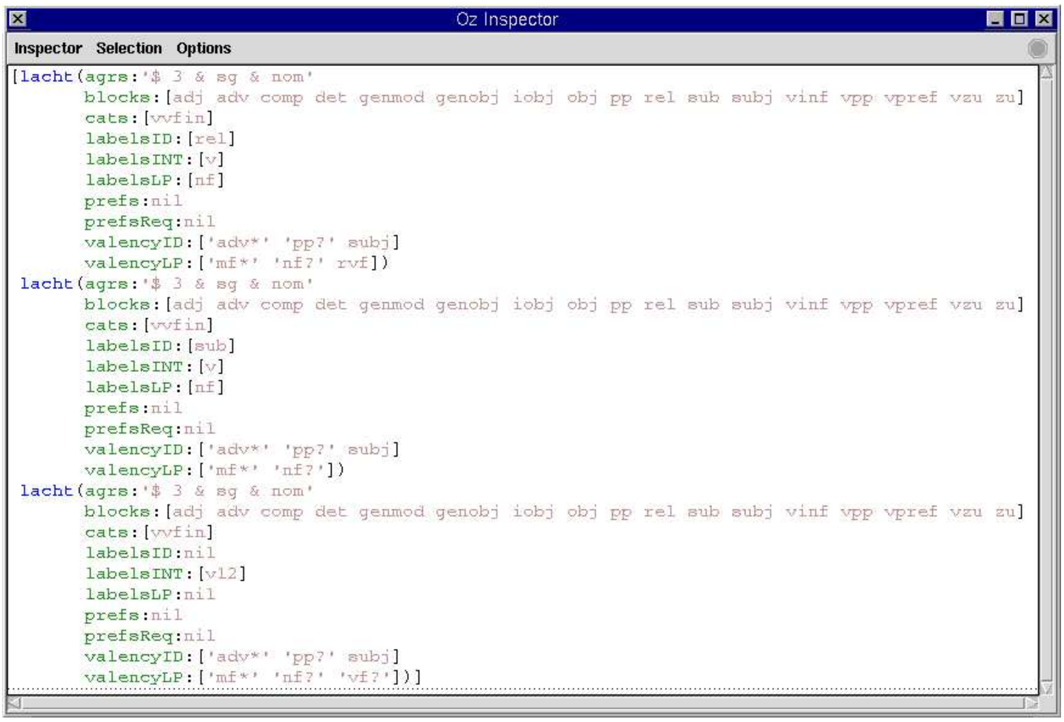
```
*********************************************
* Chapter 4
maria hat dem mann einen korb gegeben der lacht
hat peter maria geliebt

*********************************************
* Chapter 5
maria hat dem mann einen korb gegeben
maria gegeben hat dem mann einen korb
maria lieben wird ihn koennen
frau des mannes
des frau mannes

*********************************************
* Chapter 6

*********************************************
* Chapter 7

* ID part
* Finite verbs
peter lacht oft
dass peter oft lacht
ein mann der oft lacht

* Non-finite verbs
die frau hat oft mit peter gelacht
```

```
maria hat dem mann einen korb gegeben der lacht
```

Figure 10.1: GUI main window

The second part is a status display showing which grammar file is currently active (*Grammar: grammar-de.dg*), which example file is currently loaded (*Examples: examples-de.dg*) and which plugins file (*Plugins: Plugins-de.ozf*).

The third part is made up of a button labeled *Inspect lexical entries* and an adjacent text entry field. Here, a list of words (including also names of lexical types) can be entered (separated by spaces) whose corresponding lexical entries shall be depicted using the Oz Inspector (Brunklaus 2000). The function is triggered either by pressing the *Inspect lexical entries*-button or by pressing the return key while the text entry field is active. An example output of this function for the lexical entry *lacht* is given in Figure 10.2.



```
[lacht(agrs:'$ 3 & sg & nom'
       blocks:[adj adv comp det genmod genobj iobj obj pp rel sub subj vinf vpp vpref vzu zu]
       cats:[vvfin]
       labelsID:[rel]
       labelsINT:[v]
       labelsLP:[nf]
       prefs:nil
       prefsReq:nil
       valencyID:['adv*' 'pp?' subj]
       valencyLP:['mf*' 'nf?' rvf])
 lacht(agrs:'$ 3 & sg & nom'
       blocks:[adj adv comp det genmod genobj iobj obj pp rel sub subj vinf vpp vpref vzu zu]
       cats:[vvfin]
       labelsID:[sub]
       labelsINT:[v]
       labelsLP:[nf]
       prefs:nil
       prefsReq:nil
       valencyID:['adv*' 'pp?' subj]
       valencyLP:['mf*' 'nf?'])
 lacht(agrs:'$ 3 & sg & nom'
       blocks:[adj adv comp det genmod genobj iobj obj pp rel sub subj vinf vpp vpref vzu zu]
       cats:[vvfin]
       labelsID:nil
       labelsINT:[v12]
       labelsLP:nil
       prefs:nil
       prefsReq:nil
       valencyID:['adv*' 'pp?' subj]
       valencyLP:['mf*' 'nf?' 'vf?'])]
```

Figure 10.2: Lexical entries displayed by the Mozart-Oz Inspector

The fourth part of the main window is a scrollable listview of example strings. Once clicking on an example string copies it into the text entry field below, and double clicking indicates that the string shall be parsed.

The fifth part consists of a button labeled *Parse* and an adjacent text entry field. In this entry field, the string to parse can be edited. Parsing is triggered either by pressing the *Parse*-button or by pressing the return key whilst the text entry field is active.

10.3.2 Pulldown menus

Project

The *Project*-menu consists of the following menu entries. *About...* displays an about window. *Open all...* displays a file requestor where the user is asked to select a grammar file from which the GUI then extracts an identifier ⟨*id*⟩ such as "de" or "acl". The GUI then tries to load the grammar file "grammar-⟨*id*⟩.dg", the examples file "examples-⟨*id*⟩.txt" and the plugins file "Plugins-⟨*id*⟩.ozf". *Open grammar file...* displays a file requestor by which a new grammar file for the parser can be selected. *Reload grammar file* reloads the currently active grammar file. *Open examples...* loads a new examples file. Examples files are standard text files. *Reload examples* reloads the currently loaded text file and *Append examples...* appends an examples file to the list of examples currently loaded. *Open plugins...* loads a new plugins file. *Reload plugins* reloads the selected plugins file. *Close tree windows* closes all windows containing ID, LP or ID/LP analyses which is particularly useful in case the screen is cluttered up by them. *Quit* quits.

Search

The *Search*-menu consists of the two parts. The first allows the user to choose between the entries *First solution* and *All solutions First solution* means that search stops after the first solution (parse) has been found, and *All solutions* that all solutions are enumerated. The *Generate*-menu entry can be used to toggle the generate mode on and off. Generate mode is very useful for grammar debugging since it allows to generate all licensed linearizations of a bag of words, which means that also undesired linearizations can be comfortably spotted.

Principles

The *Principles*-menu contains a number of switches which can be used to toggle the TDG principles on and off. The first three of these principles are the set of ID principles, the second the set of LP principles and the third the set of ID/LP principles. These switches are extremely useful for grammar debugging: by switching off individual principles, one can easily find out which principles exclude ID/LP analyses of a particular string.

Tools

The *Tools*-menu consists of three menu entries: *Save trees as LaTeX...* allows the user to save all licensed ID/LP analyses of the currently selected sentence (in the *Parse* text entry field) into a LaTeX-file. The LaTeX-code utilized Denys Duchier's "dtree.sty"-style file which is also used for the dependency tree depictions throughout this thesis. Notice that the options configured in the *Search*-menu apply for this function.

The *Save test suite...*-menu entry starts a function which goes through all strings in the examples file, parses each string and writes statistics about the parse into a text file. The statistics are of the form

```
maria wird haben einen mann lieben koennen
 choices : 9 failed : 1 succeeded : 9
```
(10.49)

where the upper line contains the parsed string and the lower line information about the search tree: the number of choice points (`choice`), of failed spaces (`failed`) and of succeeded spaces (`succeeded`).

The *Inspect all linearizations*-menu entry uses the generate mode to generate all licensed linearizations of a bag of words. The linearizations are displayed in list form. Each of the displayed linearizations can be parsed by selecting the corresponding list in the Inspector and invoke the action *Parse* from the Inspector's context menu.

The *Configure pretty printing...* allows to individually toggle features on and off when using the *Pretty*-information action.

Plugins

Entries in the *Plugins*-menu are generated on the basis of plugins-specifications which are explained in the parser's online documentation.

10.3.3 Explorer

We employ the Oz Explorer to display the search space traversed during distribution. The Explorer is described in (Schulte 1997). Roughly, blue circles denote choice points in the search space, green diamonds solutions and red boxes failures. If the search space contains only red boxes, no parse for the string could be found. We show an example Explorer search tree in Figure 10.3.

Non-failed nodes in the search tree (blue circles and green diamonds) can be double-clicked to invoke a so-called information action. Information actions are e.g. the graphical display of the ID/LP analysis corresponding to the node in the search tree or a printout of the internal sign-structure of each node. The user can choose between several information actions using the Explorer's pulldown-menu (menu entry *Information Action* in the *Node*-menu), all of which we explain below.

Show prints out the list of Signs on stdio. *Browse* prints out the list of Signs using the Oz Browser. *Inspect* does the same using the Oz Inspector. The information

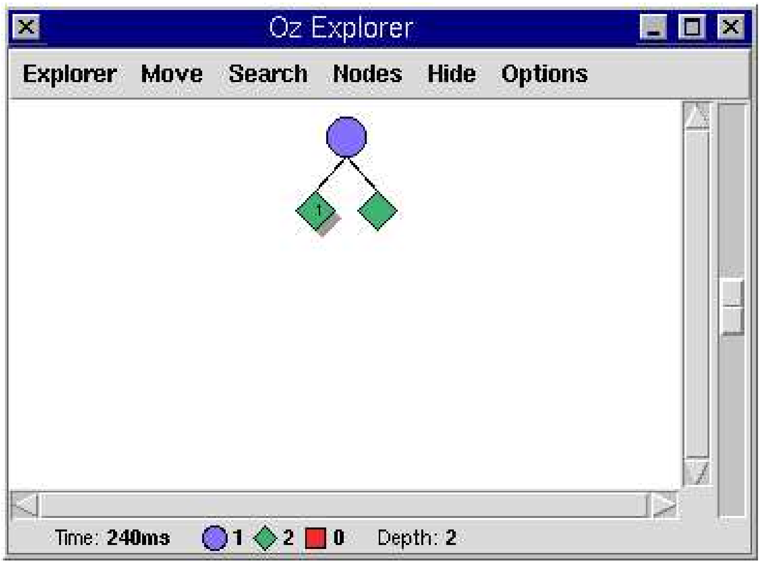


Figure 10.3: Search space displayed by the Oz Explorer

action *Pretty* prints out the list of Signs in prettified, i.e. more readable form using the Inspector. This action takes into account the options set under *Configure pretty printing...* in the *Tools*-menu. The information actions *ID tree*, *LP tree* graphically depict ID and LP trees in separate windows and *ID and LP tree* depicts both trees in one window. We show an example ID/LP tree depiction in Figure 10.4.

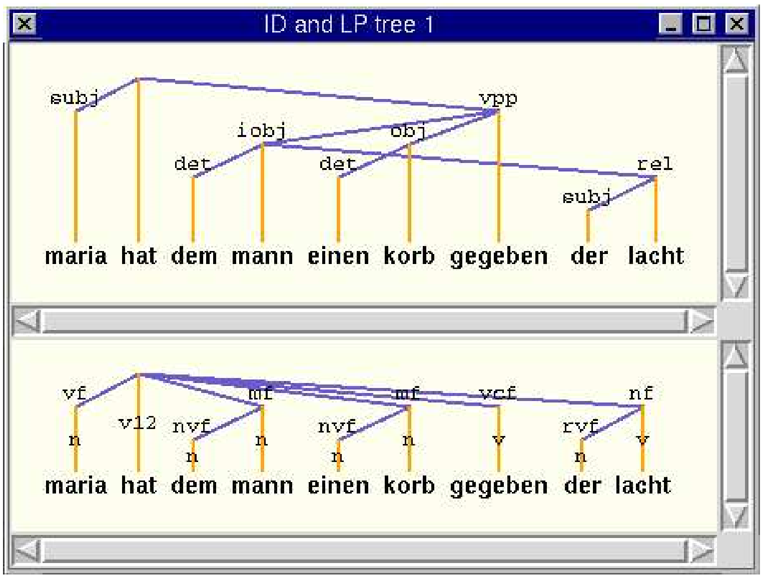

ID and LP tree 1
subj
vpp
iobj
obj
det
det
rel
subj
maria hat dem mann einen korb gegeben der lacht
vf
v12
nvf
mf
mf
vcf
nf
nvf
rvf
n
n
n
n
n
v
n
v
maria hat dem mann einen korb gegeben der lacht

Figure 10.4: An ID/LP tree depiction

10.4 Summary

We have presented a parser implementation for the TDG grammar formalism. The parser is written using state-of-the-art constraint solving technology provided by the Mozart-Oz programming language (*Mozart* 1998). The performance of the parser is very good, even without optimization. Grammars for the parser are written in a concrete grammar specification language whose concrete syntax is very similar to the set-theoretic notation used throughout the thesis. Grammars can be comfortably debugged using a graphical user interface (GUI).

The following chapter concludes this thesis and provides pointers for future research.

Chapter 11

Conclusions and future work

11.1 Summary

In the course of this thesis, we have developed a declarative grammar formalism for dependency grammar called Topological Dependency Grammar (TDG). TDG makes use of two orthogonal yet mutually constraining tree structures: the non-ordered syntactic dependency tree (ID tree) and the ordered and projective topological dependency tree (LP tree). A set of ID/LP principles based on the notion of climbing relates both trees to each other.

In chapter 2, we introduced the notion of an ID analysis. An ID analysis consists of an ID tree and a lexical assignment. ID trees are essentially non-ordered traditional dependency trees in the spirit of the dependency grammar outlined in (Duchier 1999), but modulo any constraints on word order.

In chapter 3, we presented a selection of approaches to linear precedence both in the paradigm of PSG and of DG. On the PSG-side, we reviewed Reape's (1994) theory of word order domains and Kathol's (1995) Linearization-Based Syntax. On the DG-side, we outlined Bröker's (1998) theory of word order domains in dependency grammar and Gerdes & Kahane's (2001) dependency-based theory which we called Topological Dependency Grammar′. All of these theories share the fundamental idea to dissociate immediate dominance from linear precedence, but we noticed flaws in all of them.

In chapter 4, we introduced the notion of an LP analysis. An LP analysis consists of an LP tree, a lexical assignment and an ordering on the set of nodes. LP trees represent topological structures in the spirit of topological fields theory, but from a dependency-based perspective, utilizing the notions of topological heads and topological dependents.

Chapter 5 introduced the notion of an ID/LP analysis. An ID/LP analysis is a pair of an ID analysis and an LP analysis, and both analyses are related to each other by well-formedness conditions called ID/LP principles. Climbing is the fundamental concept driving these principles: nodes can climb up from a position lower in the ID tree to a position higher up in the LP tree. As a result, the shape of the LP tree is a flattening of the ID tree's.

ID and LP analyses are subject to a number of lexicalized constraints. In chapter 6, we introduced the TDG ID/LP lexicon. In order to improve lexical economy and facilitate the statement of linguistic generalizations, we introduced a lexical type hierarchy modeled using lattices. In this lexical type hierarchy, we use lexical inheritance to obtain subtypes from more general lexical types.

In chapter 7, we presented a TDG grammar fragment for German. The focus on this chapter was on specifying the lexicon. We split the exposition of the lexicon into two parts, concerned with ID attributes and LP attributes (and ID/LP attributes) respectively.

We breathed life into the German grammar fragment in chapter 8 where we demonstrated how we are able to handle a number of difficult syntactic phenomena. In spite of its small size, the grammar fragment allowed us to treat the scrambling construction, VP-dislocation related phenomena, auxiliary flip and various phenomena associated with relative clauses.

Chapter 9 provided a formalization of the TDG grammar formalism. We precisely defined the notions of a TDG grammar and of ID, LP, and ID/LP principles. The formalization of the principles gave rise to formal renditions of the concepts of ID, LP and ID/LP analyses.

By reduction to a constraint problem following Duchier (2000), the formalization in chapter 9 lead to the implementation of a TDG parser. We outlined the parser implementation in chapter 10.

Appendix A explains a number of phenomena which not included in the grammar fragment for simplicity.

11.2 Future work

In this section, we allude to some of the open questions surrounding the TDG grammar formalism. We briefly discuss the issues of extending coverage and the application of TDG to other languages besides German and English. Then, we argue that TDG needs a morphology interface to facilitate large-scale grammar development. A syntax-semantics interface is also not yet specified, although we have already implemented a preliminary syntax-semantics interface on top of an earlier version of TDG. Finally, we discuss the incorporation of information structure.

11.2.1 Extending coverage

The grammar fragment presented in chapter 7 and also appendix A already handles a number of notorious phenomena in German syntax, but still coverage is far from complete. One future direction is thus an extension of the grammar fragment developed in the course of writing this thesis. However we think that the implemented grammar fragment demonstrates that even the most difficult phenomena in German syntax can already be treated in TDG. Thus, we are optimistic that extending coverage in order to attain a more realistic grammar should be doable.

The grammar fragment developed for this thesis was a *competence grammar* developed from a theoretical linguistics point of view. Extending such a competence grammar amounts to a high degree of linguistic research. There is however the possibility to build large-scale *performance grammars* using corpora such as the NEGRA-corpus (Uszkoreit, Brants, Duchier, Krenn, Konieczny, Oepen & Skut 1998). Christian Korthals from the NEGRA-project recently presented a first prototype for automatic lexicon extraction using the NEGRA corpus. He also successfully converted an extracted example lexicon into the TDG grammar format which means that the resulting grammar can actually be parsed. However, the lexicon extraction mechanism does not yet extract word order information from the corpus.

11.2.2 Other languages

So far, we have applied TDG only to German (this study) and English (Gabsdil et al. 2001). Another future research program might involve the application of TDG to other languages. Because of the nature of the grammar formalism (using topological fields theory), we think it should not be difficult to apply TDG for other Germanic languages such as Dutch and scandinavian languages. Especially in the HPSG community, topological fields theory has already been applied to a number of languages besides German, including Dutch and the scandinavian languages (Kathol 1995), (Kathol 2000). With respect to non-Germanic languages, Penn uses a variant of topological fields in his treatment of Serbo-Croation (Penn 1999), and Kruijff (2001) makes use of a generalized notion of topological fields in his competence grammar for tackling a number of typologically different languages with varying degrees of word order freedom. Because Penn's (1999) and Kruijff's (2001) approaches also use the fundamental notion of topological fields, we envisage that the application of TDG to other languages is realistic.

11.2.3 Morphology interface

We have not yet specified an interface to morphology in the TDG grammar formalism. At this point, the TDG lexicon is a full form lexicon listing all morphological alternatives, i.e. fully inflected morphological forms. For any sensible large-scale grammar development, we think that TDG must be augmented with an interface to a morphology module.

A first step would be to apply an already available morphology module and use it to generate a full form lexicon on the basis of a stem lexicon. A major drawback of this practice would be that the resulting full form lexicon would become very large for realistic applications. Hence, the next step would involve the integration of a morphology module to analyze full forms at parse time.

11.2.4 Syntax-semantics interface

Another direction for future work is the specification of a syntax-semantics interface for TDG. So far, we have not specified a declarative syntax-semantics interface.

Still, we are optimistic that the specification of a syntax-semantics interface is doable. Recently, we have implemented an interface to an underspecified semantics on top of an earlier version of TDG. The semantics formalism was a fragment of the Constraint Language for Lambda Structures (CLLS) (Egg, Koller & Niehren 2001). CLLS is a formalism developed for semantic underspecification in the sense of (Pinkal 1999) and has been developed in the CHORUS-project. CLLS has already been applied to model scope ambiguities, reinterpretation (Koller, Niehren & Striegnitz 2000) and ellipsis using parallelism constraints (Erk & Niehren 2000).

11.2.5 Information structure interface

Word order variation is not arbitrary. Following the Prague School of linguistics, variation in word order is a structural means to realize information structure. Thus, another issue for further work would be the integration of an interface to information structure.

Kruijff (2001) presents a multilingual grammar fragment for the DGL grammar framework which models the interaction of word order, tune and information structure. He utilizes an extended version of FGD's *Topic-Focus-Articulation* (TFA) (Sgall et al. 1986) in order to determine whether a word is contextually bounded or non-bounded. The notions topic and focus are based on the structural notions of contextual boundedness and non-boundedness which combine to effect changes in surface word order.

Kruijff's DGL is a dependency-based framework in which a categorial calculus is used to derive representations of form and meaning. He encodes the concepts of heads and dependents using *modes* in addition to the categorial notions of functor and argument. Because DGL is dependency-based, we think that adapting his theory of how to derive information structure can in principle be done. However, the theoretical basis of DGL is completely different from that of TDG: DGL is formulated on categorial type logical grounds, whereas TDG is formalized in a model-theoretic, constraint-based fashion. We do not yet see how these two views can be reconciled in order to directly make use of Kruijff's results in TDG.

Appendix A

Additions to the grammar fragment

This chapter contains additions to the grammar fragment not included in chapter 7 for simplicity. First, we discuss the issue of coherence and incoherence of verbs in German (Bech 1955). Then, we turn to the treatment of separable verb prefixes, the notion of agreement and finally the relative clause principle.

A.1 Coherence and incoherence

A.1.1 The phenomenon

Following a distinction first made by Bech (1955), we divide the set of German verbs which subcategorize for infinitival complements into three classes:

- obligatorily coherent verbs
- obligatorily incoherent verbs
- optionally coherent verbs

In a coherent construction, the infinitival complement of a governing verb lands in canonical position, i.e. in the right sentence bracket (verb-first and verb-second sentences) or to the left of the governing verb (verb-last sentences). An incoherent construction is characterized by the infinitival complement landing in non-canonical position, i.e. either intraposed (to the left periphery of the Mittelfeld) or extraposed (to the right of its governor). Obligatorily coherent verbs such as *scheinen* may only construct coherently, obligatorily incoherent verbs such as *empfehlen* may only construct incoherently, and optionally coherent verbs such as *versuchen* may construct either coherently or incoherently. Notice that regardless of their status with respect to coherence and incoherence, all non-finite verbs can be fronted into the Vorfeld in verb-second sentences.

We proceed with some examples. In (A.1) below, the infinitival complement *zu lieben* lands in canonical position to the left of the governing verb *scheint*, resulting in a coherent construction. In (A.2), *zu lieben* (in fact the entire VP *den Mann zu lieben*) is extraposed to the right of its governing verb, resulting in an incoherent construction. Since *scheint* is an obligatorily coherent verb, only the coherent construction (A.1) is grammatical. The incoherent construction (A.2) is ungrammatical:

(dass) Maria den Mann zu lieben scheint
(that) Maria the man(acc) to love seems (A.1)
"(that) Maria seems to love the man"

∗ (dass) Maria scheint, den Mann zu lieben
(that) Maria seems, the man(acc) to love (A.2)

The opposite situation occurs in constructions involving obligatorily incoherent verbs such as *empfiehlt*. Again, we show one example (A.3) where the infinitival complement *zu lieben* lands in canonical position and one (A.4) where it lands in non-canonical (extraposed) position. Hence (A.3) is a coherent and (A.4) is an incoherent construction. Because *empfiehlt* is an obligatorily incoherent verb, only the incoherent construction (A.4) is grammatical, whereas the coherent construction (A.3) is ungrammatical:

∗ (dass) Maria den Mann zu lieben empfiehlt
(that) the man(acc) to love Maria recommends (A.3)

(dass) Maria empfiehlt, den Mann zu lieben
(that) Maria recommends, the man(acc) to love (A.4)
"(that) Maria recommends to love the man"

Optionally coherent verbs such as *versucht* may give rise to either coherent or incoherent constructions: the infinitival complement may either land in canonical position (A.5) or in non-canonical position (A.6):

(dass) Maria den Mann zu lieben versucht
(that) Maria the man(acc) to love tries (A.5)
"(that) Maria tries to love the man"

(dass)	Maria	versucht,	den	Mann	zu	lieben	
(that)	Maria	tries,	the	man(acc)	to	love	(A.6)

"(that) Maria tries to love the man"

A.1.2 How we handle it

In order to distinguish between the three classes of verbs, we need to modify the grammar fragment presented in chapter 7. There, we introduced in the LP part the lexical type *t_fin_lp* for finite verbs:

$$\textit{t\_fin\_lp} \quad = \quad \begin{bmatrix} \mathsf{labels}_{\mathrm{N}} & : & \{\mathsf{v}, \mathsf{v12}\} \\ \mathsf{valency}_{\mathrm{LP}} & : & \{\mathsf{if}?, \mathsf{mf}*, \mathsf{vcf}?, \mathsf{vxf}?, \mathsf{nf}?\} \\ \mathsf{blocks} & : & \mathcal{R} \end{bmatrix} \tag{A.7}$$

But (A.7) leads to overgeneration: every finite verb inherits from *t_fin_lp*, and hence every finite verb offers the canonical field vcf for their verbal complements as well as the non-canonical fields if and vxf. Thus, all verbs may give rise to coherent and incoherent constructions. In other words, (A.7) treats all verbs as being 'optionally coherent' which is clearly wrong.

In order to remedy this problem, we redefine *t_fin_lp*, thereby changing its field valency:

$$\textit{t\_fin\_lp} \quad = \quad \begin{bmatrix} \mathsf{labels}_{\mathrm{N}} & : & \{\mathsf{v}, \mathsf{v12}\} \\ \mathsf{valency}_{\mathrm{LP}} & : & \{\mathsf{mf}*, \mathsf{nf}?\} \\ \mathsf{blocks} & : & \mathcal{R} \end{bmatrix} \tag{A.8}$$

In (A.8), we removed the fields if, vcf and vxf from the field valency of and only kept mf and nf.

In addition, we introduce three lexical types corresponding to the three classes of verbs introduced above. Obligatorily coherent verbs inherit from lexical type *t_fin_oblco_lp* given below:

$$\textit{t\_fin\_oblco\_lp} \quad = \quad \textit{t\_fin\_lp} \sqcap \begin{bmatrix} \mathsf{valency}_{\mathrm{LP}} & : & \{\mathsf{vcf}?\} \end{bmatrix} \tag{A.9}$$

(A.9) inherits from *t_fin_lp* and additionally offers the canonical field vcf. As a result, any verbal complement of such a verb must land in canonical position. Hence, verbs inheriting from this type only license coherent constructions.

Obligatorily incoherent verbs inherit from the following lexical type:

$$\textit{t\_fin\_oblinco\_lp} \quad = \quad \textit{t\_fin\_lp} \sqcap \begin{bmatrix} \mathsf{valency}_{\mathrm{LP}} & : & \{\mathsf{if}?, \mathsf{vxf}?\} \end{bmatrix} \tag{A.10}$$

Verbs of this type offer only the non-canonical positions if and vxf for their verbal complements to land in, but not the canonical position vcf. Thus, verbs inheriting from this type only license incoherent constructions.

Finally, optionally coherent verbs inherit from lexical type *t_fin_optco_lp*:

$$\mathit{t\_fin\_optco\_lp} \quad = \quad \mathit{t\_fin\_lp} \sqcap \left[\ \mathsf{valency}_{\mathrm{LP}} \ : \ \{\mathsf{if}?, \mathsf{vcf}?, \mathsf{vxf}?\}\ \right] \tag{A.11}$$

Optionally coherent verbs offer both the canonical field vcf and the non-canonical fields if and vxf.[1] Therefore, verbs of this type license both coherent and incoherent constructions.

A.1.3 Summary

The grammar fragment exhibited in chapter 7 did not distinguish between verbs which behave differently with respect to coherence and incoherence. In this section, we demonstrated how to incorporate this distinction in the fragment. We changed the lexical type for finite verbs to *t_fin_lp* and introduced the three types *t_fin_oblco_lp*, *t_fin_oblinco_lp* and *t_fin_optco_lp*, which correspond to three classes of obligatorily coherent, obligatorily incoherent and optionally coherent verbs.

A.2 Separable verb prefixes

A.2.1 The phenomenon

A German specialty is that in verb-first or verb-second sentences (but not in verb-final sentences), some verbs can 'lose' their prefixes to the right sentence bracket. Here is an example. In the verb-final sentence (A.12) below, the prefix *ein* is not separated from the finite verb *einkauft*, while in the verb-second sentence (A.13), *einkaufen* loses its prefix *ein* to the right sentence bracket:

(dass)	Maria	heute	einkauft	
(that)	Maria	today	goes shopping	(A.12)

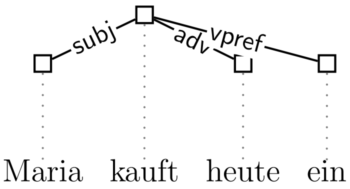
"(that) Maria goes shopping today"

Maria	kauft	heute	ein.	
Maria	goes shopping	today	(pref).	(A.13)

"Maria goes shopping today."

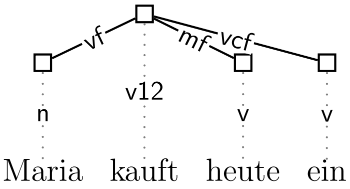
[1] Note that *t_fin_optco_lp* amounts to precisely the same type as displayed in (A.7) above.

We posit the following ID analysis for (A.13), where *Maria* is the subject, *heute* an adverb and the prefix *ein* a vpref-dependent of *kauft*:



(A.14)

The corresponding LP tree is shown in (A.15): *Maria* lands in the vf, *heute* in the mf and the verb prefix *ein* in the vcf of *kauft*:



(A.15)

A.2.2 How we handle it

We introduce new lexical types required to handle separable verb prefixes. First, we add to the set $\mathcal{R}$ of grammatical roles the role vpref for 'verb prefix'. Then, we posit the subtype *t_v12_vpref* of the lexical type *t_v12_lp* for finite verbs in verb-first or verb-second position with a separable prefix:

$$t\_v12\_vpref \quad = \quad t\_v12\_lp \sqcap \begin{bmatrix} \mathsf{valency}_{\mathrm{ID}} & : & \{\mathsf{vpref}\} \\ \mathsf{valency}_{\mathrm{LP}} & : & \{\mathsf{vcf?}\} \end{bmatrix} \tag{A.16}$$

In its role valency, *t_v12_vpref* requires a vpref-dependent, and in its field valency, it offers the field vcf for the separated prefix to land in.

The separable prefix itself inherits from lexical type *t_vpref*:

$$t\_vpref \quad = \quad \begin{bmatrix} \mathsf{labels}_{\mathrm{ID}} & : & \{\mathsf{vpref}\} \\ \mathsf{valency}_{\mathrm{ID}} & : & \emptyset \\ \mathsf{labels}_{\mathrm{LP}} & : & \{\mathsf{vcf}\} \\ \mathsf{labels}_{\mathrm{N}} & : & \{\mathsf{v}\} \\ \mathsf{valency}_{\mathrm{LP}} & : & \emptyset \\ \mathsf{blocks} & : & \emptyset \end{bmatrix} \tag{A.17}$$

(A.17) declares that verb prefixes accept only role vpref and have an empty role valency. Their only accepted field is the right sentence bracket vcf, and their node label v. Their field valency is empty, as is their set of blocked roles.

We still need to match the verb with its prefix, thereby excluding sentences like (A.18) below, where the separated prefix *zu* does not match the verb *kauft*:

$$\begin{array}{lllll} * & \text{Maria} & \text{kauft} & \text{heute} & \text{zu} \\ & \text{Maria} & \text{goes shopping} & \text{today} & \text{(pref)} \end{array} \tag{A.18}$$

To this end, we define a set *Vpref* of separable prefixes and introduce the two lexical attributes $\mathsf{vprefsReq} : \mathcal{E} \to 2^{Vpref}$ and $\mathsf{vprefs} : \mathcal{E} \to 2^{Vpref}$. vprefsReq assigns to each word a set of prefixes which the word admits and vprefs a set of prefixes which the word 'is': $\mathsf{vprefs}(e)$ is the singleton set containing only $\pi_1 \circ e$ (the string s in lexical entry $e = (s, e_t)$) if e is a verb prefix, and $\mathsf{vprefs}(e)$ is empty if e is no verb prefix.

As an example, we show how to derive the lexical entries for the finite verb *kauft* and the separable verb prefix *ein*. Here is an abbreviation of the lexical entry for *kauft*:

$$(\mathit{kauft} \quad , \quad \mathit{t\_v12\_vpref} \sqcap \ \ldots \ \sqcap \begin{bmatrix} \mathsf{vprefReq} & : & \{\mathsf{ein}\} \\ \mathsf{vprefs} & : & \emptyset \end{bmatrix}) \tag{A.19}$$

And below, we show the lexical entry for the separable verb prefix *ein*:

$$(\mathit{ein} \quad , \quad \mathit{t\_vpref} \sqcap \begin{bmatrix} \mathsf{vprefsReq} & : & \emptyset \\ \mathsf{vprefs} & : & \{\mathsf{ein}\} \end{bmatrix}) \tag{A.20}$$

We match verb prefixes and their corresponding verbs using the ID edge constraint for role vpref:

$$\Gamma_{\mathsf{vpref}}(w, w') \quad \equiv \quad \lambda w, w' \cdot (\mathsf{vprefs}(w') \subseteq \mathsf{vprefReq}(w)) \tag{A.21}$$

ID edge constraint (A.21) states that the vprefs-value of the verb prefix w' must be a subset of the vprefsReq-value of the finite verb w. The ungrammatical sentence (A.18) is now excluded because *zu* is not in the set of admitted verb prefixes for *kauft*:

$$\mathsf{vprefs}(\mathit{zu}) = \{\mathit{zu}\} \quad \not\subseteq \quad \mathsf{vprefsReq}(\mathit{kauft}) = \{\mathit{ein}\} \tag{A.22}$$

A.2.3 Summary

The grammar fragment exhibited in chapter 7 did not include a treatment of separable verb prefixes. Here, we illustrated an approach to handle separable prefixes in TDG. First, we introduced new lexical types for separable prefix verbs (*t_v12_vpref*) and separable prefixes (*t_vpref*). Then, we posed two new lexical attributes, vprefsReq and vprefs, and used ID edge constraints to match separable prefix verbs with their corresponding separable prefixes.

A.3 Agreement

A.3.1 The phenomenon

In this section, we turn to agreement in German. For one, finite verbs must agree with their subjects in person and number, and the subject must be in nominative case. (A.23) below is an example where both conditions are satisfied: the finite verb *lacht* and the subject *Mann* agree in person and number and *Mann* is in nominative case:

Der	Mann	lacht.	
The	man(nom,3rd,sg)	laughs(3rd,sg).	(A.23)

"The man laughs."

In (A.24), the finite verb does not agree with its subject in number: *lacht* is singular and *Männer* plural:

*	Die	Männer	lacht.	(A.24)
	The	men(nom,3rd,pl)	laughs(3rd,sg).	

(A.25) shows an example where the finite verb *does* agree with the subject, but the subject is not in nominative but in genitive case:

*	Des	Mannes	lacht.	(A.25)
	The	man(gen,3rd,sg)	laughs(3rd,sg).	

Within each German NP, the noun must agree with its determiner and its adjectives in gender, number, definiteness and case. In the following examples, we concentrate only on definiteness. Example (A.26) below exhibits a definite NP and (A.27) an indefinite NP:

der	reiche	Mann	
the(def)	rich(def)	Mann(def)	(A.26)

"the rich man"

ein	reicher	Mann	
a(indef)	rich(indef)	man(indef)	(A.27)

"a rich man"

If we e.g. use the indefinite adjective in the definite NP, and the definite adjective in the indefinite NP, the examples become ungrammatical:

*	der	reicher	Mann	(A.28)
	the(def)	rich(indef)	man(def)	

*	ein	reiche	Mann	(A.29)
	a(indef)	rich(def)	man(indef)	

A.3.2 How we handle it

We employ ID edge constraints to handle agreement. Although we focus here on agreement in German, we claim that the same approach can also be applied for other languages.

We distinguish between the morphological notions of *person*, *gender*, *number*, *definiteness* and *case* and capture these notions by the following five sets:

$$\begin{aligned} Person &= \{\mathsf{1},\mathsf{2},\mathsf{3}\} && \text{(A.30)}\\ Gender &= \{\mathsf{masc},\mathsf{fem},\mathsf{neut}\} && \text{(A.31)}\\ Number &= \{\mathsf{sg},\mathsf{pl}\} && \text{(A.32)}\\ Def &= \{\mathsf{def},\mathsf{indef},\mathsf{undef}\} && \text{(A.33)}\\ Case &= \{\mathsf{nom},\mathsf{gen},\mathsf{dat},\mathsf{acc}\} && \text{(A.34)} \end{aligned}$$

The set *Person* consists of 1 (first person), 2 (second person) and 3 (third person). *Gender* consists of masc (masculine), fem (feminine) and neut (neuter). The set *Number* comprises sg (singular) and pl (plural), and the set *Def* includes def (definite), indef (indefinite) and undef (no determiner).[2] Finally, the set *Case* is made up of nom (nominative), gen (genitive), dat (dative) and acc (accusative).

From these five sets, we construct the set *Agr* of agreement tuples as follows:

$$Agr = Person \times Gender \times Number \times Def \times Case \qquad \text{(A.35)}$$

For convenience, we also define the following four sets of agreement tuples in nominative, genitive, dative and accusative case:

$$\begin{aligned} NOM &= Person \times Gender \times Number \times Def \times \{\mathsf{nom}\} && \text{(A.36)}\\ GEN &= Person \times Gender \times Number \times Def \times \{\mathsf{gen}\} && \text{(A.37)}\\ DAT &= Person \times Gender \times Number \times Def \times \{\mathsf{dat}\} && \text{(A.38)}\\ ACC &= Person \times Gender \times Number \times Def \times \{\mathsf{acc}\} && \text{(A.39)} \end{aligned}$$

We introduce the lexical attribute $\mathsf{agrs} : \mathcal{E}_t \to 2^{Agr}$ mapping lexical types to sets of agreement tuples. This captures the fact that the same surface form may be consistent with a number of agreement tuples. We also posit the function $\mathsf{agr} : V \to Agr$ which maps nodes to agreement tuples. $\mathsf{agr}(w)$ picks out precisely one agreement tuple from the set $\mathsf{agrs}(w)$ of possible agreement tuples of a node w:

$$\mathsf{agr}(w) \in \mathsf{agrs}(w) \qquad \text{(A.40)}$$

[2] The corresponding German terms are *schwache Flexion* for def, gemischte Flexion for indef and *starke Flexion* for undef.

Notice that for example finite verbs do not inflect for gender, case and definiteness, and nouns (except for pronouns) do not inflect for person, yet we assign to them agreement tuples including gender, case and definiteness (for finite verbs) and person (nouns). If a word does not inflect for any of the projections in the agreement tuple, we simply assign to that projection its range.

We employ ID edge constraints to state agreement requirements. We capture the requirement that finite verbs must agree with their subjects and their subjects must be in nominative case by ID edge constraint (A.41):

$$\Gamma_{\mathsf{subj}}(w, w') \equiv \lambda w, w' \cdot \ \mathsf{agr}(w) = \mathsf{agr}(w') \ \wedge \ \mathsf{agr}(w') \in \mathit{NOM} \tag{A.41}$$

stating that the agreement values of the finite verb w and the subject w' must be equal ($\mathsf{agr}(w) = \mathsf{agr}(w')$) and that the subject must be in nominative case ($\mathsf{agr}(w') \in \mathit{NOM}$).

We also need to ensure that objects are in accusative case, indirect object in dative case and genitival dependents in genitive case. This is accomplished by the following four ID edge constraints:

$$\begin{aligned} \Gamma_{\mathsf{obj}}(w, w') &\equiv \lambda w, w' \cdot \ \mathsf{agr}(w') \in \mathit{ACC} && \text{(A.42)} \\ \Gamma_{\mathsf{iobj}}(w, w') &\equiv \lambda w, w' \cdot \ \mathsf{agr}(w') \in \mathit{DAT} && \text{(A.43)} \\ \Gamma_{\mathsf{genobj}}(w, w') &\equiv \lambda w, w' \cdot \ \mathsf{agr}(w') \in \mathit{GEN} && \text{(A.44)} \\ \Gamma_{\mathsf{genmod}}(w, w') &\equiv \lambda w, w' \cdot \ \mathsf{agr}(w') \in \mathit{GEN} && \text{(A.45)} \end{aligned}$$

And determiners and adjectives agree with their nouns, which is characterized by the following ID edge constraints:

$$\begin{aligned} \Gamma_{\mathsf{det}}(w, w') &\equiv \lambda w, w' \cdot \ \mathsf{agr}(w) = \mathsf{agr}(w') && \text{(A.46)} \\ \Gamma_{\mathsf{adj}}(w, w') &\equiv \lambda w, w' \cdot \ \mathsf{agr}(w) = \mathsf{agr}(w') && \text{(A.47)} \end{aligned}$$

A.3.3 Summary

The grammar fragment exhibited in chapter 7 did not include a treatment of agreement. In this section, we illustrated how to handle a number of agreement phenomena in TDG. We introduced the new lexical attribute agrs which assigns to each lexical type a set of possible agreement tuples. Then, we stated agreement requirements for subjects, objects, indirect objects, genitival dependents and also for determiners and adjectives using ID edge constraints.

A.4 Relative clause principle

A.4.1 The phenomenon

For two reasons, the treatment of relative clauses in the grammar fragment in chapter 7 leads to overgeneration:

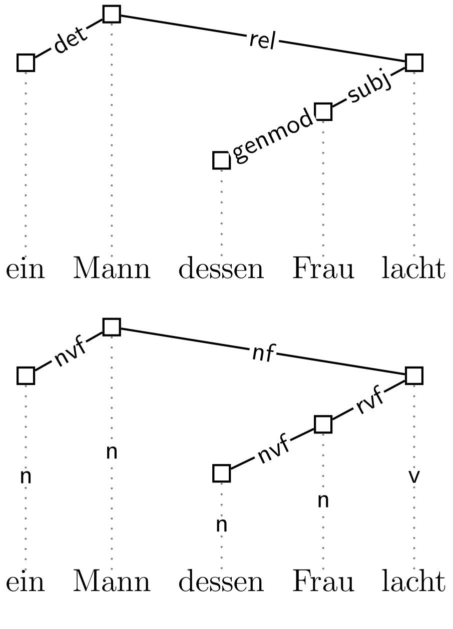
- We do not require that there must be precisely one relative pronoun per relative clause.
- We do not require that the relative pronoun agrees with the modified noun.

As an example, consider the following sentence which exhibits NP pied piping:

ein	Mann,	dessen	Frau	lacht	
a	man,	whose	woman	laughs	(A.48)

"a man whose woman laughs"

The corresponding ID/LP analysis is given below:

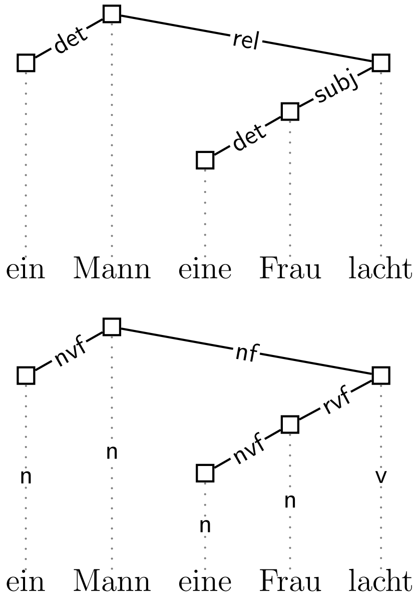


(A.49)

In (A.49), not the relative pronoun *dessen* but the nominal governor *Frau* of the relative pronoun lands in the relative clause Vorfeld rvf. But then how do we guarantee that there actually is a relative pronoun somewhere in the relative clause, or, more precisely, somewhere in the yield of the node landing in the rvf? The grammar fragment specified in chapter 7 does not guarantee that there is a relative pronoun in each relative clause: it also licenses the following ungrammatical example where the relative pronoun is missing:

$*$	ein	Mann,	eine	Frau	lacht	(A.50)
	a	man,	a	woman	laughs	

The licensed ID/LP analysis for this sentence would be:



(A.51)

On the other hand, we do not yet require that the relative pronoun agrees with the modified noun. So far, our grammar fragment does license the ungrammatical example in (A.52) below where the relative pronoun *die* has feminine gender and the corresponding noun *Mann* masculine gender:

$*$	ein	Mann,	die	lacht	(A.52)
	a	man(masc),	who(fem)	laughs	

A.4.2 How we handle it

We have shown that we need to enhance our treatment of relative clauses in order to avoid overgeneration. To this end, we introduce an additional ID/LP principle called *relative clause principle*:

Principle A.1 *Each relative clause has precisely one relative pronoun in its relative clause Vorfeld which agrees with the corresponding noun.*

We formalize the relative clause principle as follows. We introduce the function $\mathsf{rpros} : V \to 2^V$ mapping nodes to sets of nodes and the set $\mathit{Allrpros} : 2^V$ denoting the set of all relative pronouns in a sentence. $\mathsf{rpros}(w)$ denotes the singleton set containing only the relative clause's relative pronoun if w is a finite verb heading a relative clause. If w is not the head of a relative clause, $\mathsf{rpros}(w)$ denotes the empty set.

In the first part of the relative clause principle, we state that the set of all relative pronouns in a sentence is the disjoint union of the sets denoted by $\mathsf{rpros}(w)$ for each node:

$$\mathit{Allrpros} \quad = \quad \uplus\{\mathsf{rpros}(w) \mid w \in V\} \tag{A.53}$$

The second part of the relative clause principle ensures that the relative pronoun has the correct category. Therefore, we pose the function $\mathsf{cat} : V \to \mathit{Cats}$ which maps each node to its category. The category of relative pronouns is rpro. The following constraint states that a word is of category rpro if and only if it is in the set $\mathit{Allrpros}$ of all relative pronouns in the sentence:

$$\mathit{Allrpros} \quad = \quad \{w \in V \mid \mathsf{cat}(w) = \mathsf{rpro}\} \tag{A.54}$$

For the next part of the relative clause principle, we overload the functional notation ρ: $\rho(V)$ is the image of set V by function ρ, defined as:

$$\rho(V) \quad = \quad \cup\{\rho(w) \mid w \in V\} \quad \text{for } \rho \in \mathcal{R} \tag{A.55}$$

i.e. $\rho(V)$ denotes the set of all nodes whose incoming edge is labeled with ρ. We impose two further constraints: (A.56) requires that the set $\mathsf{rpros}(w)$ of relative pronouns assigned to node w has cardinality zero or one. (A.57) states that the cardinality of $\mathsf{rpros}(w)$ is one if and only if w's incoming edge is labeled with role rel, i.e. if w is the head of a relative clause:

$$\forall w \in V : \qquad |\mathsf{rpros}(w)| \le 1 \tag{A.56}$$

$$\forall w \in V : \qquad w \in \mathsf{rel}(V) \;\equiv\; |\mathsf{rpros}(w)| = 1 \tag{A.57}$$

In the fourth part of the relative clause principle, we state that the relative pronoun must be in the set of nodes reachable by traversing downwards zero or more edges from the node landing in the rvf:

$$\forall w \in V : \quad \mathsf{rpros}(w) \;\subseteq\; \to^* \circ\, \mathsf{rvf}(w) \tag{A.58}$$

Here, we make use of the function $\mathsf{rvf} : V \to 2^V$ which assigns each node to the set of its dependents in the rvf. $\to^* \circ\, \mathsf{rvf}(w)$ denotes all nodes reachable by traversing downwards zero or more edges from the node landing in the rvf.

Finally, we require that the relative pronoun agrees with the modified noun. Here, we make use of the function $\mathsf{mothers}_{\mathrm{ID}} : V \to 2^V$ mapping each node to the set of its syntactic mothers:

$$\begin{array}{rl}\forall w \in V : \ |\mathsf{rpros}(w)| = 1 \ \Rightarrow & \forall w' \in \mathsf{rpros}(w) \ \forall w'' \in \mathsf{mothers}_{\mathrm{ID}}(w) : \\ & \mathsf{agr}|_{Gender \times Number}(w') = \mathsf{agr}|_{Gender \times Number}(w'')\end{array} \tag{A.59}$$

where $\mathsf{agr}|_{Gender \times Number}$ stands for the projection of the agreement tuple to gender and number.

A.4.3 Summary

In this section, we added to TDG a new ID/LP principle called relative clause principle. The goal of this principle is twofold: on the one hand, it ensures that there is precisely one relative pronoun in each relative clause. On the other hand, it requires that the relative pronoun agrees with the modified noun in gender and number.

Bibliography

Ades, A. E. & Steedman, M. J. (1982), ‘On the order of words’, *Linguistics and Philosophy* **4**, 517–558.

Askedahl, J. O. (1983), ‘Kohärenz und Inkohärenz in deutschen Infinitfügungen: Vorschlag zur begrifflichen Klärung’, *Lingua* **59**, 177–196.

Bach, E. (1981), Discontinuous constituents in generalized categorial grammars, *in* ‘Proceedings of the 11th Annual Meeting of the Northeast Linguistic Society’, Amberst/GLSA, pp. 515–531.

Bech, G. (1955), *Studien über das deutsche Verbum Infinitum*, Det Kongelige Danske videnskabernes selskab. Historisk-Filosofiske Meddelelser, Bd. 35, Nr. 2 (1955) and Bd. 36, Nr. 6 (1957), Munksgaard, Kopenhagen/DK.

Becker, T. & Rambow, O. (1994), ‘Parsing free word-order languages in polynomial time’.

Blackburn, P. (1994), Structures, languages and translations: The structural approach to feature logic, *in* C. J. Rupp, M. A. Rosner & R. L. Johnson, eds, ‘Constraints, Language and Computation’, Academic Press, London/UK, pp. 1–27.

Bröker, N. (1998), Separating surface order and syntactic relations in a dependency grammar, *in* ‘COLING-ACL 98 - Proc. of the 17th Intl. Conf. on Computational Linguistics and 36th Annual Meeting of the ACL.’, Montreal/CAN.

Bröker, N. (1999), *Eine Dependenzgrammatik zur Kopplung heterogener Wissensquellen*, Linguistische Arbeiten 405, Max Niemeyer Verlag, Tübingen/FRG.

Brunklaus, T. (2000), Der Oz Inspector - Browsen: Interaktiver, einfacher, effizienter, Master’s thesis, Universität des Saarlandes.

Chomsky, N. (1986), *Barriers*, Linguistic Inquiry Monograph 13, MIT Press, Cambridge/MA.

Covington, M. A. (1984), *Syntactic theory in the High Middle Ages*, Cambridge University Press.

Curry, H. B. (1961), Some logical aspects of grammatical structure, *in* 'Structure of Language and its Mathematical Aspects: Proceedings of the Twelfth Symposium in Applied Mathematics', American Mathematical Society, pp. 56–68.

Debusmann, R. (2001), Movement as well-formedness conditions, *in* 'Proceedings of the Sixth ESSLLI Student Session', Helsinki/FIN.

Duchier, D. (1999), Axiomatizing dependency parsing using set constraints, *in* 'Sixth Meeting on the Mathematics of Language', Orlando/FL.

Duchier, D. (2000), 'Configuration of labeled trees under lexicalized constraints and principles', To appear in the Journal of Language and Computation.

Duchier, D. (2001), Lexicalized syntax and topology for non-projective dependency grammar, *in* 'Eighth Meeting on Mathematics of Language', Helsinki/FIN.

Duchier, D. & Debusmann, R. (2001), Topological dependency trees: A constraint-based account of linear precedence, *in* '39th Annual Meeting of the Association for Computational Linguistics (ACL 2001)', Toulouse, France.

Egg, M., Koller, A. & Niehren, J. (2001), 'The constraint language for lambda structures', *Journal of Logic, Language, and Information* .

Erdmann, O. (1886), *Grundzüge der deutschen Syntax nach ihrer geschichtlichen Entwicklung dargestellt*, Erste Abteilung, Stuttgart/FRG.

Erk, K. & Niehren, J. (2000), Parallelism constraints, *in* 'International Conference on Rewriting Techniques and Applications', Vol. 1833 of *Lecture Notes in Computer Science*, Springer-Verlag, Berlin, Norwich/UK, pp. 110–126.

Fouvry, F. & Meurers, D. (2000), Towards a platform for linearization grammars, *in* E. W. Hinrichs, D. Meurers & S. Wintner, eds, 'Proceedings of the Workshop on Linguistic Theory and Grammar Implementation', ESSLLI 2000, Birmingham, UK, pp. 153–168.

Gabsdil, M., Koller, A. & Striegnitz, K. (2001), Building a text adventure on description logic, *in* 'Proceedings of KI-2001 Workshop on Applications of Description Logics', Vienna/AUT.

Gaifman, H. (1965), 'Dependency systems and phrase-structure systems', *Information and Control* **8**(3), 304–337.

Gazdar, G., Klein, E., Pullum, G. & Sag, I. (1985), *Generalized Phrase Structure Grammar*, B. Blackwell, Oxford/UK.

Gerdes, K. & Kahane, S. (2001), Word order in german: A formal dependency grammar using a topological hierarchy, *in* '39th Annual Meeting of the Association for Computational Linguistics (ACL 2001)', Toulouse/FRA. To appear.

Greenberg, J. H. (1966), Some universals of grammar with particular reference to the order of meaningful elements, *in* J. H. Greenberg, ed., 'Universals of Language', The MIT Press, Cambridge/MA, pp. 73–114. second edition.

Haider, H. (1985), 'Der Rattenfängerei muss ein Ende gemacht werden', *wiener linguistische gazette* pp. 27–50.

Halliday, M. A. K. (1961), 'Categories of the theory of grammar'.

Halliday, M. A. K. (1994), *Introduction to functional grammar*, Edward Arnold, London/UK.

Hawkins, J. A. (1983), *Word Order Universals*, Academic Press.

Hays, D. G. (1964), 'Dependency theory: A formalism and some observations', *Language* **40**, 511–525.

Hellwig, P. (1986), Dependency unification grammar, *in* 'Proc. of the 11th Int. Conf. on Computational Linguistics', pp. 195–198.

Herling, S. (1821), 'Über die Topik der deutschen Sprache'.

Hinrichs, E. & Nakazawa, T. (1994), Linearizing AUXs in German verbal complexes, *in* J. Nerbonne, K. Netter & C. Pollard, eds, 'German in Head-Driven Phrase Structure Grammar', CSLI, Stanford/CA, pp. 11–37.

Höhle, T. (1986), Der Begriff "Mittelfeld", Anmerkungen über die Theorie der topologischen Felder, *in* W. Weiss, H. E. Wiegand & M. Reis, eds, 'Akten des 7. Internationalen Germanisten-Kongresses, Göttingen 1985', Vol. 3, Max Niemeyer Verlag, Tübingen/FRG, pp. 329–340.

Hudson, R. A. (1990), *English Word Grammar*, B. Blackwell, Oxford/UK.

Joshi, A. K. (1987), An introduction to tree-adjoining grammars, *in* A. Manaster-Ramer, ed., 'Mathematics of Language', John Benjamins, Amsterdam/NL, pp. 87–115.

Kahane, S., Nasr, A. & Rambow, O. (1998), Pseudo-projectivity: a polynomially parsable non-projective dependency grammar, *in* '36th Annual Meeting of the Association for Computational Linguistics (ACL 1998)', Montreal/CAN.

Kathol, A. (1995), Linearization-Based German Syntax, PhD thesis, Ohio State University.

Kathol, A. (2000), *Linear Syntax*, Oxford University Press.

Koller, A., Niehren, J. & Striegnitz, K. (2000), 'Relaxing underspecified semantic representations for reinterpretation', Grammars 3:2-3.

Kruijff, G.-J. M. (2001), A Categorial-Modal Architecture of Informativity, PhD thesis, Charles University, Prague/Czech Republic.

Lecerf, Y. (1960), 'Programme des conflits, modèle des conflits.', *Bulletin bimestriel de l'atala* .

Mel'čuk, I. (1988), *Dependency Syntax: Theory and Practice*, State Univ. Press of New York, Albany/NY.

Meurers, W. D. (1994), 'A modified view of the german verbal complex'. Handout of a talk given on 7. Oktober 1994 at the HPSG workshop in Heidelberg/Germany.

Meurers, W. D. & Kuthy, K. D. (2001), 'On partial constituent fronting in german', *Journal of Comparative Germanic Linguistics* .

Mozart (1998). http://www.mozart-oz.org/.

Müller, S. (1999), *Deutsche Syntax deklarativ. Head-Driven Phrase Structure Grammar für das Deutsche*, Linguistische Arbeiten 394, Max Niemeyer Verlag, Tübingen/FRG.

Penn, G. (1999), A generalized-domain-based approach to serbo-croatian second-position clitic placement, *in* G. Bouma, E. Hinrichs, G.-J. M. Kruijff & R. Oehrle, eds, 'Constraints and Resources in Natural Language Syntax and Semantics', CSLI Publications, Stanford/CA, pp. 119–136.

Pinkal, M. (1999), On underspecification, *in* 'Proceedings of the 5th International Workshop on Computational Semantics', Tilburg/NL.

Pollard, C. & Sag, I. (1994), *Head-Driven Phrase Structure Grammar*, University of Chicago Press, Chicago.

Reape, M. (1994), Domain union and word order variation in german, *in* J. Nerbonne, K. Netter & C. Pollard, eds, 'German in Head-Driven Phrase Structure Grammar', CSLI, Stanford/CA, pp. 151–197.

Robinson, J. J. (1970), 'Dependency structures and transformation rules', *Language* **46**, 259–285.

Ross, J. R. (1967), Constraints on Variables in Syntax, PhD thesis, MIT.

Schulte, C. (1997), Oz Explorer: A visual constraint programming tool, *in* L. Naish, ed., 'Proceedings of the Fourteenth International Conference on Logic Programming', The MIT Press, Leuven, Belgium, pp. 286–300.

Sgall, P., Hajicova, E. & Panevova, J. (1986), *The Meaning of the Sentence in its Semantic and Pragmatic Aspects*, D. Reidel, Dordrecht/NL.

Starosta, S. (1988), 'The case for lexicase'.

Steele, S. M. (1978), Word order variation: A typological study, *in* J. Greenberg, ed., 'Universals of Human Language', Stanford University Press, Stanford, pp. 585–624.

Tesnière, L. (1959), *Eléments de Syntaxe Structurale*, Klincksiek, Paris/FRA.

Uszkoreit, H. (1987), *Word Order and Constituent Structure in German*, CSLI, Stanford/CA.

Uszkoreit, H., Brants, T., Duchier, D., Krenn, B., Konieczny, L., Oepen, S. & Skut, W. (1998), 'Studien zur performanzorientierten Linguisitk. Aspekte der Relativsatzextraposition im Deutschen', *Kognitionswissenschaft* **7**(3).

Venneman, T. (1977), 'Konstituenz und Dependenz in einigen neueren Grammatiktheorien', *Sprachwissenschaft* **2**, 259–301.

Index