A grammar formalism for dependency parsing with word order domains

Ralph Debusmann
Advisors: Denys Duchier and Joachim Niehren

September 19, 2002

Abstract

Recently, Duchier (1999) presented a new, constraint-based axiomatization of dependency parsing. We propose to develop an abstract grammar formalism for Duchier's axiomatization. In addition, we propose to formulate on a high level of abstraction statements for expressing constraints on linear precedence. We believe that we can solve this part by adapting for dependency parsing Reape's (e.g. Reape 1994) notion of word order domains, which has already been thoroughly investigated in the context of HPSG for German.

Contents

1 Introduction

My thesis pursues two major goals[1]:

1. The development of a grammar formalism for Duchier's (1999) dependency parser, excluding issues of word order (section 3). Duchier's dependency parser and the accompanying grammar formalism form what we call the *ID framework*[2].

2. The development of a theory of word order, integrated into Duchier's (1999) axiomatization of dependency grammar (sections 4 and 5). We call this part of our proposal the *LP framework*[3].

Both are enhancements of Duchier's axiomatization of dependency parsing, which we summarize in section 2.

1.1 Goal 1: Towards a declarative grammar formalism

A practical shortcoming of (Duchier 1999) is the lack of a formalism for declarative grammar development. Therefore, the first contribution of my thesis will be the development of a grammar formalism made up of two languages: One for

[1] A more detailed list of goals can be found in appendix A.
[2] *ID* stands for "immediate dominance".
[3] *LP* stands for "linear precedence".

specifying the lexicon and another for stating grammatical conditions. Section 3 presents an overview of the grammar formalism envisaged.

A further goal is to write a parser generator (using the *Gump* parser generator) that takes a grammar specification (in our grammar formalism) as its input and returns an executable dependency parser for this grammar.

1.2 Goal 2: A new formulation of word order domains

Duchier's (1999) formulation of dependency parsing lacks a theory of word order. Constraints on word order are stated in an ad-hoc fashion to make the parser undergenerate. A finer-grained account of word order is needed to be able to describe word order more adequately. Therefore, the second contribution of my thesis is an adaptation of Reape's (e.g. Reape 1994) notion of word order domains for dependency grammar. Word order domains have proven to be an adequate means for describing German word order within HPSG. For instance, (Müller 1999) and (Kathol 1995) both make use of Reape's word order domains in their theories of German grammar. An introduction to the concept of word order domains is given in section 4.

Our notion of word order domains is described in section 5. It not only fits into Duchier's axiomatization of dependency parsing but also makes finer-grained distinctions. We will demonstrate that phenomena such as partial VP extraposition and relative clause extraposition are problematic for Reape's but not for our refined notion of word order domains.

2 Duchier's dependency parser

In this section, we outline Duchier's (1999) formulation of dependency parsing. The two major contributions of my thesis are both built onto Duchier's dependency parser.

First, let us introduce the basic concepts behind dependency grammar.

2.1 Dependency grammar

Modern dependency grammar has been pioneered by the French linguist Lucien Tesnière (1959). The key idea is this: In a natural language sentence, all but one word *depend* on other words. The word that does not depend on any other is called the *root* of the sentence. A word depends on another if it is a complement or a modifier of the latter.

Unlike traditional phrase structure grammar, dependency grammar postulates only lexical nodes but no phrasal nodes. In addition, many flavors of dependency grammar allow *non-projective* analyses (i.e. allow crossing edges), which make dependency grammar particularly well suited to account for languages with a high degree of word order variation and discontinuity such as German.

2.2 A non-projective dependency grammar

Duchier (1999) proposes such a non-projective dependency grammar. He regards dependency grammar as a 7-tuple of finite sets:

(1) $DG = \langle \text{Words}, \text{Cats}, \text{Agrs}, \text{Comps}, \text{Mods}, \text{Lexicon}, \text{Rules} \rangle$

Words is a set of strings of word forms. Cats is a set of lexical categories such as `v` for verb and `n` for noun. Agrs is a set of agreement tuples such as ⟨`masc sing 3 nom`⟩ for "masculine singular third person nominative".

Each complement or modifier in the dependency tree fills a certain *role*. We distinguish roles by their *role type*. For instance, the subject of a finite verb fills the role subj. The disjoint union of the sets Comps of complement role types and Mods of modifier role types (e.g. adv for "adverb") forms the set Roles of all role types.

Lexicon is a set of lexical entries which are represented by attribute-value-matrices like the one shown in (2):

(2)
$$\begin{bmatrix} \text{string} & \text{"liebt"} \\ \text{cat} & \texttt{v} \\ \text{agr} & \langle \texttt{masc sing 3 nom} \rangle \\ \text{comps} & \{\text{subj}, \text{obj}\} \end{bmatrix}$$

Here, the string-feature maps to a string from the set Words of words. cat maps to a category from the set Cats of categories and agr to an agreement tuple from the set Agrs of agreement tuples. comps maps to subsets of the set Comps of complement role types.

Rules (also called *role constraints*) is a family of binary predicates Γ_ρ for all $\rho \in$ Roles. Role constraints express grammatical conditions that must hold between two words connected by a dependency edge in the dependency tree.

2.3 The dependency tree

We will now introduce the formal notion of a dependency tree. As in Duchier (1999), we assume a finite set $\mathcal{V}$ of nodes, representing the words of the input sentence, and identify it with the set of integers $\{1\,..\,n\}$. We label edges between two nodes by role types $\rho \in$ Roles. The set of edges in the dependency tree is $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V} \times$ Roles and we write $w \xrightarrow{\rho} w'$ for $\langle w, w', \rho \rangle \in \mathcal{E}$. Now $\langle \mathcal{V}, \mathcal{E} \rangle$ is a directed graph with labeled edges, and a dependency tree T can be defined as:

$$T = \langle (\mathcal{V}, \mathcal{E}), \text{entry} \rangle$$

where entry is a function that maps each node in $\mathcal{V}$ to a lexical entry in Lexicon:

$$\text{entry} : \text{Nodes} \mapsto \text{Lexicon}$$

Dependency trees are subject to the following treeness constraints:

1. A node has at most one mother.
2. There is precisely one node which has no mother. This node is called root.
3. There are no cycles.

The following conditions for well-formedness of dependency trees must be satisfied. First, any complement required by a node's valency must be realized precisely once:

(3) $\forall w \in \mathcal{V}, \forall \rho \in \mathsf{comps}(\mathsf{entry}(w)) \quad \exists! w' \in \mathcal{V}, w \xrightarrow{\rho} w' \in \mathcal{E}$

Second, if there is an edge emanating from w, then it must be labeled either by a complement type in w's valency or by a modifier type:

(4) $\forall w \xrightarrow{\rho} w' \in \mathcal{E} \quad \rho \in \mathsf{comps}(\mathsf{entry}(w)) \cup \mathrm{Mods}$

Third, whenever there is an edge $w \xrightarrow{\rho} w'$, then the *role constraint* (grammatical condition) $\Gamma_\rho(w, w')$ for $\Gamma_\rho \in \mathrm{Rules}$ must be satisfied in T:

(5) $\forall w \xrightarrow{\rho} w' \in \mathcal{E} \quad T \models \Gamma_\rho(w, w')$

2.4 Role constraints

A role constraint is a grammatical condition which licenses an edge $w \xrightarrow{\rho} w'$ in the dependency tree. We write $\Gamma_\rho(w, w')$ for a role constraints licensing an edge $w \xrightarrow{\rho} w'$. Role constraints are largely used to control agreement and assign grammatical functions to dependents.

For instance, the dependency edge $w \xrightarrow{\mathsf{subj}} w'$ is licensed by the role constraint $\Gamma_{\mathsf{subj}}(w, w')$ iff w' is a noun (`n`) or a pronoun (`pro`), agrees with w and has nominative case:

$$\begin{aligned} \Gamma_{\mathsf{subj}}(w, w') \quad \equiv \quad & \mathsf{cat}(w') \in \{\mathtt{n}, \mathtt{pro}\} \wedge \\ & \mathsf{agr}(w) = \mathsf{agr}(w') \wedge \\ & \mathsf{agr}(w') \in \mathrm{NOM} \end{aligned}$$

where NOM represents the set of all agreement tuples Gender $\times$ Number $\times$ Person $\times \{\mathtt{nom}\}$.

Another example. Here, the role constraint Γ_{adj} ensures that an adjective `adj` may modify nouns only and agrees with the noun:

$$\begin{aligned} \Gamma_{\mathsf{adj}}(w, w') \quad \equiv \quad & \mathsf{cat}(w) = \mathtt{n} \wedge \\ & \mathsf{cat}(w') = \mathtt{adj} \wedge \\ & \mathsf{agr}(w) = \mathsf{agr}(w') \end{aligned}$$

2.5 Lexical economy

Duchier (1999) improves lexical economy by doing two things: First, he collapses together entries that differ only in the values of their agreement-tuples. For this reason, he replaces the attribute agr by agrs. The values of agrs are sets of agreement tuples. In a similar manner, Duchier replaces the cat-attribute by cats, the values of cats being sets of categories.

Another source for redundancy in the lexicon are optional complements. Therefore, as a second measure, Duchier proposes to model the valency of entries e with two sets $\mathsf{compsReq}(e)$ and $\mathsf{compsOpt}(e)$ for required and optional complements respectively instead of using just one set $\mathsf{comps}(e)$.

Duchier calls the resulting representation a *lexicon* entry to distinguish it from a *lexical* entry as the one in (2). *Lexicon* entries generate *lexical* entries. The lexical entries generated by (6) are of the form of (7).

(6)
$$\begin{bmatrix} \mathsf{string} & \mathrm{S} \\ \mathsf{cats} & \mathrm{C} \\ \mathsf{agrs} & \mathrm{A} \\ \mathsf{compsReq} & \mathrm{Req} \\ \mathsf{compsOpt} & \mathrm{Opt} \end{bmatrix}$$

(7)
$$\left\{ \begin{bmatrix} \mathsf{string} & \mathrm{S} \\ \mathsf{cat} & c \\ \mathsf{agr} & a \\ \mathsf{comps} & \mathrm{Comps} \end{bmatrix} \;\middle|\; \begin{array}{l} c \in \mathrm{C} \wedge a \in \mathrm{A} \wedge \\ \mathrm{Req} \subseteq \mathrm{Comps} \subseteq \mathrm{Req} \cup \mathrm{Opt} \end{array} \right\}$$

2.6 An example analysis

As an example, look at the dependency tree of sentence (8) in Figure 1. The upper part shows nodes represented as boxes, connected by directed edges. The integers appearing in the boxes reflect the position of the corresponding word in the input sentence.

(8) *(dass) ein Buch Maria zu lesen verspricht.*
(that) a book Maria to read promises.
"(that) Maria promises to read a book."

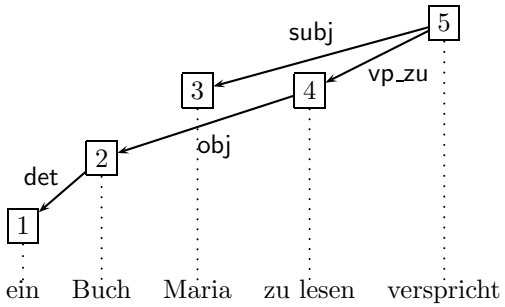


Figure 1: Example dependency tree

Figure 2 depicts the lexical entry assigned to each node of the dependency tree by the entry-function. We write the underscore "_" in the lexical entry corresponding to "zu lesen" to indicate that all assignments of values (e.g. all agreement tuples) are licensed by the grammar.

2.7 Constraint-based axiomatization and parsing

(Duchier 1999) describes a constraint model for the formal model outlined above, which we do not flesh out here. The constraint model uses finite domain and finite set variables to encode the quantities and mappings defined, and formulates constraints on them that precisely capture the conditions the formal model stipulates. By having this reduction at hand, Duchier's axiomatization of dependency parsing naturally yields executable dependency parsers.

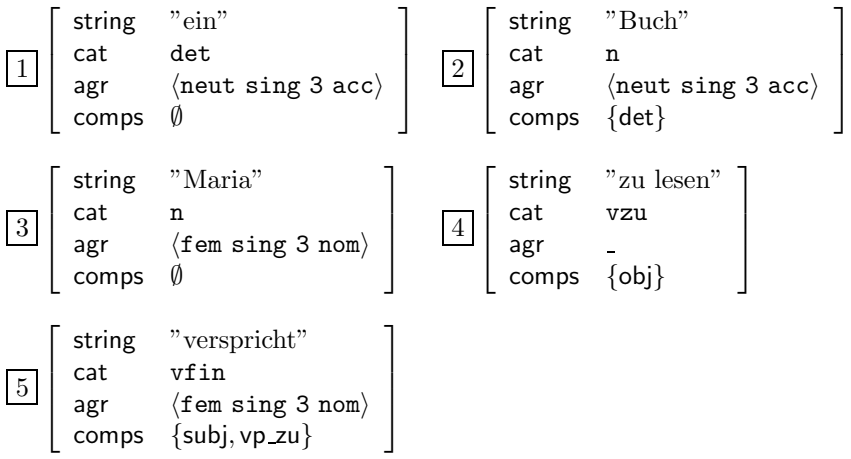


Figure 2: Entries corresponding to the nodes in Figure 1

2.8 Evaluation

Duchier developed several prototype parsers for both German and English using the techniques described in (Duchier 1999). The prototype parsers support PPs (in the case of ambiguous attachment possibilities, all of them are enumerated), infinitive clauses, separable verb prefixes (in German), and relative clauses. The treatment of relative clauses seems a little ad-hoc though and indicates that the linguistic theory behind Duchier's dependency parser needs some enhancement.

Additionally, Duchier's dependency parsers cover the following phenomena: topicalization, fronting (including partial fronting), extraposition and scrambling. However, many important notions such as coordination and nested extraposition fields are missing.

3 Towards a declarative grammar formalism

The first major contribution of my thesis is a declarative grammar formalism for Duchier's axiomatization dependency parsing. Duchier's axiomatization and the grammar formalism make up what we call the *ID framework*. The grammar formalism is made up of two languages: one for specifying the lexicon and one for stating grammatical conditions (role constraints). In this proposal, we present just an outline of the *abstract syntax* of the grammar formalism that we envisage. The development of a *concrete syntax* for it is a further goal of the thesis but not attempted herein.

3.1 The lexicon

The first building block of our grammar formalism is an expression language for specifying the lexicon. Expressions of this language are inductively defined

below:

E	::=	$\_$	don't care symbol
	|	c	finite domain constant
	|	c_b	base type constant (integer, string)
	|	$f(E)$	feature access
	|	$\langle E_1 \ldots E_n \rangle$	tuple
	|	$\{E_1 \ldots E_n\}$	set
	|	$E_1 \cap E_2$	set intersection
	|	$E_1 \cup E_2$	set union
	|	$\overline{E}$	set complementation
	|	$[f_1 : E_1 \ldots f_n : E_n]$	avm
	|	$\Uparrow E$	set generator
	|	$E_1 \wedge E_2$	set generator intersection
	|	$E_1 \vee E_2$	set generator union
	|	$\neg E$	set generator complementation

The *set generator* allows us to write for instance $\Uparrow\langle \texttt{masc} \vee \texttt{neut}, \texttt{sing}, 3, \_ \rangle$ to denote the set of all agreement tuples where gender is "masculine" or "neuter", number "singular" and person 3rd, and where case is not restricted, as indicated by the *don't care symbol* "$\_$".

The expression language we envisage is statically typed. However, we do not flesh out the type system in this proposal.

3.2 An example lexicon specification

We now specify a lexicon using the expression language outlined above. We start with specifying finite domain constants for gender, number, person and case.

$$\begin{array}{lcl} \text{Gender} & = & \{\texttt{masc}, \texttt{fem}, \texttt{neut}\} \\ \text{Number} & = & \{\texttt{sg}, \texttt{pl}\} \\ \text{Person} & = & \{1, 2, 3\} \\ \text{Case} & = & \{\texttt{nom}, \texttt{gen}, \texttt{dat}, \texttt{acc}\} \end{array}$$

We define agreement tuples Agr as being made up of the four domains specified above:

$$\text{Agr} = \langle \text{Gender}, \text{Number}, \text{Person}, \text{Case} \rangle$$

Next we define finite domain constants for categories Cat:

$$\text{Cat} = \{\texttt{adj}, \texttt{adv}, \texttt{det}, \texttt{n}, \texttt{vfin}, \texttt{vzu}\}$$

Similarly, we define the finite domains of complements Comp and modifiers Mods:

$$\begin{array}{lcl} \text{Comp} & = & \{\texttt{dative}, \texttt{det}, \texttt{obj}, \texttt{subj}, \texttt{vp\_zu}\} \\ \text{Mod} & = & \{\texttt{adj}, \texttt{adv}\} \end{array}$$

From the domains defined above we can now build sets of finite domain constants:

$$\begin{array}{lcl} \text{Agrs} & = & \{\text{Agr}\} \\ \text{Cats} & = & \{\text{Cat}\} \\ \text{Comps} & = & \{\text{Comp}\} \\ \text{Mods} & = & \{\text{Mod}\} \end{array}$$

And here is an example lexicon entry, abiding the specifications above:

$$\begin{bmatrix} \mathsf{string} & \text{"verspricht"} \\ \mathsf{cats} & \{\mathtt{vfin}\} \\ \mathsf{agrs} & \Uparrow\langle \_,\ \mathtt{sing},\ \mathtt{3},\ \mathtt{nom}\rangle \\ \mathsf{compsReq} & \{\mathtt{subj}, \mathtt{vp\_zu}\} \\ \mathsf{compsOpt} & \{\mathtt{dative}\} \end{bmatrix}$$

where $\Uparrow\langle \_,\ \mathtt{sing},\ \mathtt{3},\ \mathtt{nom}\rangle$ denotes all agreement tuples in:

$$\text{Gender} \times \{\mathtt{sing}\} \times \{\mathtt{3}\} \times \{\mathtt{nom}\}$$

3.3 Role constraints

Role constraints are the second building block in our grammar formalism. They are used to express grammatical conditions on pairs of words to be connected by a dependency edge. We propose to develop an abstract constraint language on top of the expression language defined above for specifying the lexicon:

$$\begin{array}{rcll} C & ::= & E_1 = E_2 & \text{equal} \\ & | & E_1 \neq E_2 & \text{inequal} \\ & | & E_1 \in E_2 & \text{element} \\ & | & E_1 \notin E_2 & \text{not element} \\ & | & E_1 \subseteq E_2 & \text{subset} \\ & | & E_1 \parallel E_2 & \text{disjoint} \\ & | & C_1 \wedge C_2 & \text{conjunction} \end{array}$$

We now present an example of a role constraint, expressed in terms of the abstract syntax defined above.

$$\begin{array}{rcl} \Gamma_{\mathtt{adv}}(w, w') & \equiv & \mathsf{cat}(w) \in \{\mathtt{vfin}, \mathtt{vzu}\} \wedge \\ & & \mathsf{cat}(w') = \mathtt{adv} \end{array}$$

Here is another example role constraint:

$$\begin{array}{rcl} \Gamma_{\mathtt{subj}}(w, w') & \equiv & \mathsf{cat}(w') \in \{\mathtt{n}\} \wedge \\ & & \mathsf{agr}(w) = \mathsf{agr}(w') \wedge \\ & & \mathsf{agr}(w') \in \Uparrow\langle \_,\ \_,\ \_, \mathtt{nom}\rangle \end{array}$$

where $\Uparrow\langle \_,\ \_,\ \_, \mathtt{nom}\rangle$ denotes the set of agreement tuples in:

$$\text{Gender} \times \text{Number} \times \text{Person} \times \{\mathtt{nom}\}$$

4 An introduction to Reape's word order domains

The grammar formalism for Duchier's axiomatization of dependency parsing is not equipped with a means to state constraints on word order. The second contribution of my thesis is the development of a theory of word order which fits into Duchier's formulation of dependency parsing and is called *LP framework*. Our theory of word order is an adaptation of Reape's notion of word order domains for dependency parsing.

This section serves to motivate the intuitions behind Reape's word order domains, in order to make it easier to see the commonalities and differences to our adaptation for dependency grammar (section 5).

4.1 Motivation

As Reape's word order domains have proven highly successful as an ingredient of theories of German grammar (e.g. Müller 1999, Kathol 1995), we concentrate on describing German word order phenomena too in this proposal.

In German, the order among nominal complements of verbs (this sequence of words is often called *Mittelfeld*) is almost arbitrary. As an example, consider the sentences below[4]:

(9) *(dass) Maria ein Buch liest.*
(that) Maria a book reads.
"(that) Maria reads a book."

(10) *(dass) ein Buch Maria liest.*
(that) a book Maria reads.
"(that) Maria reads a book."

In sentence (9), the nominal arguments "Maria" and "ein Buch" are in *canonical* order, i.e. the subject ("Maria") of the governing verb "liest" precedes the object ("ein Buch"). In (10), the object precedes the subject of "liest", resulting in a non-canonical ordering, which is called *scrambling*.

4.1.1 A traditional phrase structure analysis

How would traditional phrase structure syntax analyses of the two example sentences look like? For (9), one could immediately come up with the analysis shown in Figure 3:

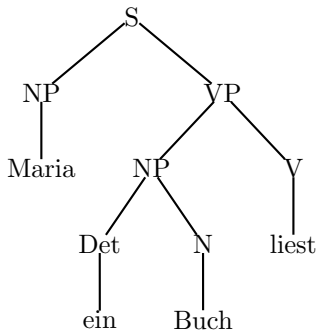


Figure 3: Phrase structure tree for sentence (9)

But what about (10)? Note that traditionally, phrase structure trees are ordered trees, i.e. the surface order of the analysed string must be equal to the concatenation of the leaves of the parse tree from left to right. Holding on to the same analysis as in Figure 3 then, we would end up in postulating an analysis as in Figure 4.

Figure 4 contains a crossing edge. We call analyses with crossing edges *non-projective*, and analyses without crossing edges *projective*. Non-projective analyses are traditionally avoided in most phrase structure-based theories of grammar. Two leading arguments against them are:

[4]Note that we only consider subordinate (verb-last) sentences in all of this proposal for the sake of keeping things as simple as possible.

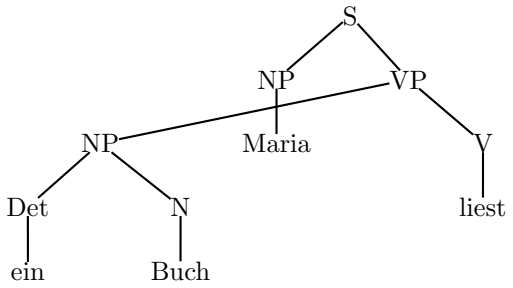


Figure 4: Phrase structure tree for sentence (10)

- Crossing edges result in a combinatory explosion during parsing if linear precedence is not properly controlled.
- Crossing edges go against the intuition of *constituents*, namely that they should be continuous or *convex* sequences of words.

4.1.2 A revised phrase structure analysis

Revised phrase structure analyses of sentences (9) and (10) that do not fall foul to requiring non-projective analyses are depicted in Figures 5 and 6:

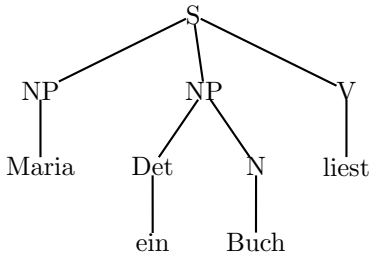


Figure 5: Revised phrase structure tree for sentence (9)

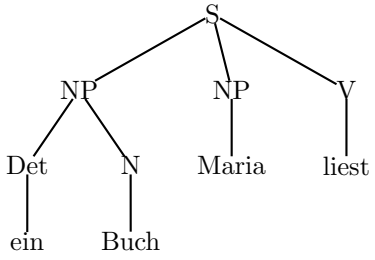


Figure 6: Revised phrase structure tree for sentence (10)

This kind of *flatter* analysis seems perfectly suited to account for the almost arbitrary order among nominal complements in German. In addition, by separating immediate dominance (ID) and linear precedence (LP), the number

of schemata required to account for all possible permutations of e.g. nominal complements can be reduced to a minimum: The idea is to leave the order of daughters of each node unrestricted a priori and then employ LP-statements to exclude unacceptable daughter sequences. For instance, the LP-statement NP $\prec$ V would constrain NPs to precede Vs at each node, correctly excluding sentences such as the ones given below:

(11) * (dass) Maria liest ein Buch.

(12) * (dass) ein Buch liest Maria.

(13) * (dass) liest Maria ein Buch.

(14) * (dass) liest ein Buch Maria.

Still, with the kind of analysis exemplified above we do not get very far. Consider sentences with embedded non-finite verbs, forming with their finite verbal heads so-called *verb clusters* or *verb complexes*. For instance, the finite verb "verspricht" and its non-finite complement "zu lesen" below form such a *verb cluster*:

(15) *(dass) Maria ein Buch zu lesen verpricht.*
(that) Maria a book to read promises.
"(that) Maria promises to read a book."

(16) *(dass) ein Buch Maria zu lesen verspricht.*
(that) a book Maria to read promises.
"(that) Maria promises to read a book."

Phrase structure analyses in the spirit of the revised analyses shown in Figure 5 and 6 would lead to analyses presented in Figure 7 and 8:

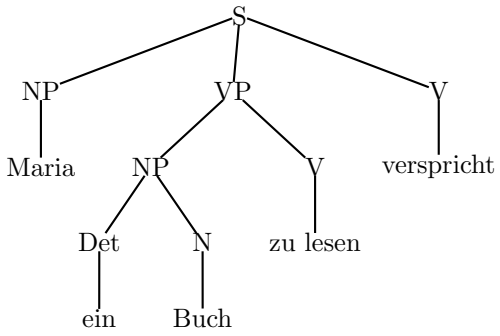


Figure 7: Phrase structure tree for sentence (15)

Again, we have to postulate analyses with crossing edges in the non-canonical case (Figure 8), something we thought we could avoid by adopting a revised, flatter phrase structure analysis.

What choices do we have left then? One approach that has been evaluated by (Netter 1991) is known under the name of *clause union*, and is understood as a merging of the subcategorization frames of verbs in a verb cluster. The result is a complex predicate with a composite subcategorization frame.

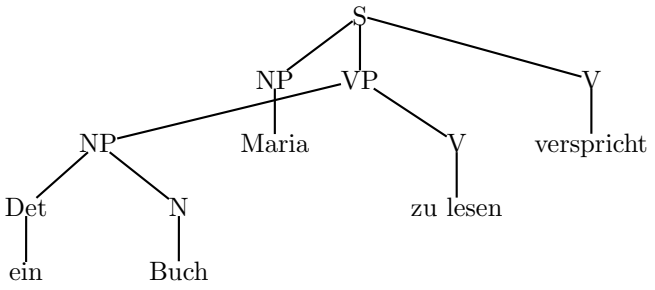


Figure 8: Phrase structure tree for sentence (16)

Another choice posits even flatter phrase structure analyses, as proposed by e.g. Uszkoreit (1987) in the GPSG ID/LP-framework (Gazdar, Klein, Pullum & Sag 1985). Figure 9 illustrates such an analysis for sentence (16):

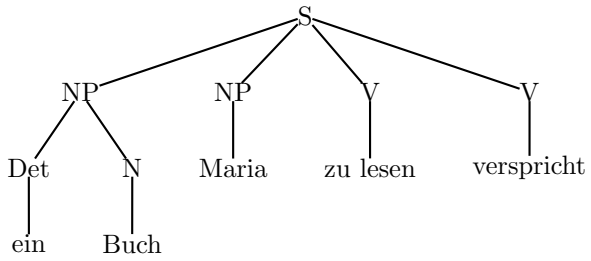


Figure 9: Flat phrase structure tree for sentence (16)

But while such an analysis enables us to account for *scrambled* sentences with embedded verbs without having to permit crossing edges, it does not reflect much more than the linear order of the words in the sentence. Analyses as in Figure 9 lack much of the information about dominance that trees such as Figure 7 incorporate. For instance, in Figure 7, the V "zu lesen" c-commands only its complement NP "ein Buch", whereas in Figure 9, it c-commands all other constituents. Hence, adopting flat analyses, c-command can not be used as an indicator about head-complement relations anymore.

4.2 Introducing word order domains

As argued above, traditional phrase structure analyses run into difficulties when confronted with discontinuous constituents (for instance, *scrambled* nominal complements). We presented two possible solutions:

- Postulating non-projective analyses. Problems: run against the intuition of what a *constituent* is and lead to combinatory explosion.
- Postulating *flat* phrase structure analyses such as in Figure 9. Problem: lack of dominance information.

Reape's (e.g. Reape 1994) solution is to assume two analyses instead of one: his *ID tree*[5] (ID for "Immediate Dominance") is a phrase structure analysis that breaks with tradition in that it is *unordered*. His *LP tree* (LP for "Linear Precedence") is a flattened version of the corresponding ID tree and is *ordered*. Reape uses a binary operation termed *sequence union* to obtain a set of ordered LP trees from an unordered ID tree. These correspond to the licensed surface orders.

4.3 Sequence union

We define *sequence union* as the deterministic function "$\bullet$":

$$\bullet : \text{Nodes}^* \times \text{Nodes}^* \mapsto 2^{\text{Nodes}^*}$$

where Nodes^* is a sequence of ID tree nodes.

For two sequences of ID tree nodes d_1 and d_2, $d_1 \bullet d_2$ returns the set D of sequences satisfying the following constraints:

1. d contains all elements in d_1 and d_2.
2. The respective order of elements in d_1 and d_2 is preserved in $d \in D$.
3. Other than that, the order of the elements in $d \in D$ is unconstrained.

As an example, let $d_1 = [a, b]$ and $d_2 = [c, d]$:

$$[a, b] \bullet [c, d] = \quad \{\ [a, b, c, d] \\ [a, c, b, d] \\ [a, c, d, b] \\ [c, a, b, d] \\ [c, a, d, b] \\ [c, d, a, b]\ \}$$

Additionally, we overload the function symbol "$\bullet$" with a function that takes sets of sequences as arguments:

$$\bullet : 2^{\text{Nodes}^*} \times 2^{\text{Nodes}^*} \mapsto 2^{\text{Nodes}^*}$$

Here is its definition:

$$S_1 \bullet S_2 \quad \equiv \quad \cup\{d_1 \bullet d_2 \mid d_1 \in S_1,\ d_2 \in S_2\}$$

4.4 Obtaining LP trees

How does Reape obtain word order domains or *LP trees* from *ID trees*?

First, every phrasal (non-lexical) ID tree node has an attribute dom, whose value is an ordered sequence of nodes called the *domain* of that node:

$$\textsf{dom} : \text{Nodes} \mapsto \text{Nodes}^*$$

Additionally, every node bears a boolean attribute unioned. We write:

[5] Reape calls *ID trees syntax trees* in his papers.

(17) $w[\cup -]$ for $\mathsf{unioned}(w) = -$

(18) $w[\cup +]$ for $\mathsf{unioned}(w) = +$

The domain $\mathsf{dom}(w)$ of a node w is obtained from the contributions $\uparrow w'$ of the daughters $w' \in \mathsf{dtrs}(w)$. If $w'[\cup -]$, the contribution is $[w']$, and if $w'[\cup +]$, the contribution is $\mathsf{dom}(w')$:

(19) $\mathsf{dom}(w) \in \bullet\{\uparrow w' \mid w' \in \mathsf{dtrs}(w)\}$

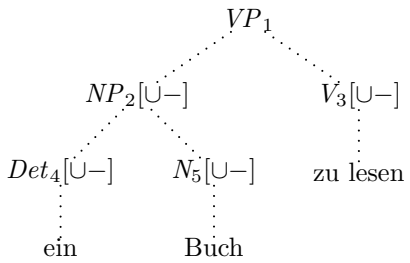
(20) $\uparrow w' = \begin{cases} [w'] & \text{if } w'[\cup -] \\ \mathsf{dom}(w') & \text{if } w'[\cup +] \end{cases}$

When $w'[\cup -]$, we say that the ID tree daughter w' is *inserted* into the domain of its mother w, and that it is *merged* when $w'[\cup +]$. We assume that lexical nodes are always $[\cup -]$.

4.5 A first example

The unordered ID tree[6] in (21) is a first example of how to construct LP trees from ID trees. We draw ID tree edges dotted to indicate that ID trees are unordered. The values of the unioned-attributes of the nodes are annotated to the right of their node labels.

(21)



We assume that lexical nodes are always $[\cup -]$ in this proposal, hence Det_4, N_5 and V_3 are $[\cup -]$ in (21). The same goes for (27) below. We specify NP_2 as $[\cup -]$ in (21) and $[\cup +]$ in (27) arbitrarily in order to illustrate what LP trees are licensed by which kind of ID trees.

Now let us try to build the LP tree at node NP_2 in ID tree (21). As both daughters of NP_2 are $[\cup -]$, they must be *inserted* into the domain of their mother:

(22) $\mathsf{dom}(NP_2) \in \uparrow Det_4 \bullet \uparrow N_5 = [Det_4] \bullet [N_5] = \{[Det_4, N_5], [N_5, Det_4]\}$

We can exclude the second sequence $[N_5, Det_4]$ by enforcing a rule of linear precedence that requires determiners to precede nouns (Det $\prec$ N) and arrive at:

(23) $\mathsf{dom}(NP_2) = [Det_4, N_5]$

When building the LP tree at VP_1, we also have to *insert* both daughters into their mother domain (both are $[\cup -]$):

[6] We label phrasal ID tree nodes with their cat-value and subscript them with distinct numbers.

(24) $\mathsf{dom}(VP_1) \in \uparrow NP_2 \bullet \uparrow V_3 = [NP_2] \bullet [V_3] = \{[NP_2, V_3], [V_3, NP_2]\}$

We exclude the second sequence $[V_3, NP_2]$ by requiring that NPs must precede Vs (NP $\prec$ V) and arrive at:

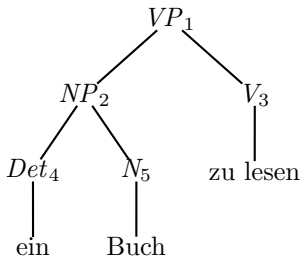
(25) $\mathsf{dom}(VP_1) = [NP_2, V_3]$

These steps license only one ordered LP tree, which has the same structure as the ID tree from which it was derived. Note that we draw solid LP tree edges as opposed to dotted ID tree edges to indicate that LP trees are ordered. The LP tree dt is obtained from the domains of ID tree nodes as shown here:

$$\mathsf{dom}(w) = [w_1, \ldots, w_n] \quad \Rightarrow \quad \mathsf{dt}(w) = w(\mathsf{dt}(w_1), \ldots, \mathsf{dt}(w_n))$$

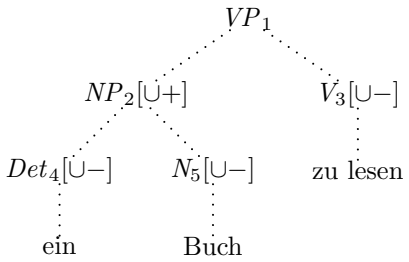
And here is the LP tree corresponding to $\mathsf{dom}(VP_1) = [NP_2, V_3]$:

(26)



Now consider another ID tree. Here, contrary to (21), NP_2 is $[\cup+]$ instead of $[\cup-]$:

(27)

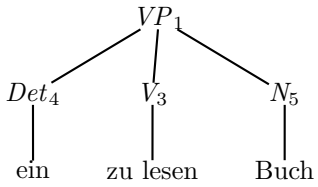


The dom-value at NP_2 is the same as above. Now, let us construct the dom-value at VP_1. As NP_2 is ($[\cup+]$), it must be *merged* into the domain of its mother:

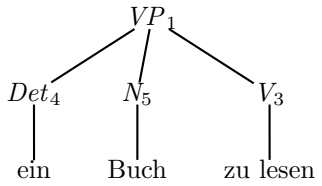
(28) $\mathsf{dom}(VP_1) \in \uparrow NP_2 \bullet \uparrow V_3 = \mathsf{dom}(NP_2) \bullet V_3 = [Det_4, N_5] \bullet [V_3] =$
$\{[Det_4, N_5, V_3], [Det_4, V_3, N_5], [V_3, Det_4, N_5]\}$

(28) licenses the following three ordered LP trees:

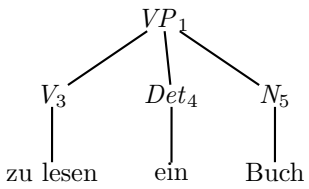
(29)



(30)



(31)



In a grammar for German, one would want to exclude such LP tree analyses. In (29) for instance, the V "zu lesen" has slipped in between the Det and the N. Reape postulates so-called *language specific principles* like (32) for German, which requires NPs to be [∪−], to prohibit insertions into NP domains:

(32) $\mathrm{NP} \Rightarrow [\cup -]$

Reape also introduces a more compact notation for LP trees which he calls *labeled bracketed string notation*. For instance the LP tree corresponding to $\mathsf{dom}(VP_1) = [Det_4, N_5, V_3]$ as shown in (30) is shown in (33) in Reape's labeled bracketed string notation:

(33) $[[\text{ein}]_{\text{Det}}\ [\text{Buch}]_{\text{N}}\ [\text{zu lesen}]_{\text{V}}]_{\text{VP}}$

4.6 Language specific principles

As already mentioned, the value of unioned is specified by what Reape calls *language specific principles*. For German, Reape assumes that only verbal projections (VPs and Ss) can be [∪+], whereas the non-verbal projections (e.g. NPs) must always be [∪−]:

(34) $w[\cup +] = + \quad \Rightarrow \quad \mathsf{cat}(w) \in \{\mathrm{VP}, \mathrm{S}\}$

(34) presumes that only verbal projections can be discontinuous in German. This premise poses a problem for his analysis of relative clause extraposition (see section 4.10.2).

A second language specific principle requires nodes which represent extraposed constituents[7] to be [∪−]:

(35) $\mathsf{extraposed}(w) = + \quad \Rightarrow \quad w[\cup -]$

We show that (35) becomes problematic when confronted with partial VP extraction (see section 4.10.1).

[7] We assume a boolean function extraposed that assigns + to nodes representing extraposed constituents and − to nodes representing non-extraposed constituents here.

4.7 Linear precedence

As Reape's *ID trees* are unordered, rules of linear precedence (LP) must be expressed in the ordered *LP tree*. A LP-rule has the form $c_1 \prec c_2$, for two categories c_1 and c_2. A domain D satisfies $c_1 \prec c_2$ iff:

$$\forall w_1, w_2 \in D \quad \mathsf{cat}(w_1) = c_1 \wedge \mathsf{cat}(w_2) = c_2 \quad \Rightarrow \quad \mathsf{index}(w_1) < \mathsf{index}(w_2)$$

where index is a function that identifies each node with its linear position in the input sentence.

For the fragment of German that we consider, we assume the following linear precedence constraints:

(36) NP $\prec$ V

(37) Det $\prec$ N

(38) N $\prec$ RelS

(36) postulates that NPs must precede verbs, (37) that determiners must precede nouns and (38) that nouns must precede relative sentences that presumably modify them.

4.8 A thorough example

We now illustrate the intuitions behind word order domains by going through the example of obtaining the LP tree for sentence (16), repeated below as (39):

(39) *(dass)* *ein* *Buch* *Maria* *zu lesen* *verspricht.*
(that) a book Maria to read promises.
"(that) Maria promises to read a book."

We assume as the *ID tree* for this sentence Figure 10. We annotate the ID tree nodes with their appropriate unioned-values: All non-verbal projections and lexical nodes (see section 4.4) are [∪−], only verbal projections can be either [∪−] or [∪+] (as VP_3 in Figure 10).

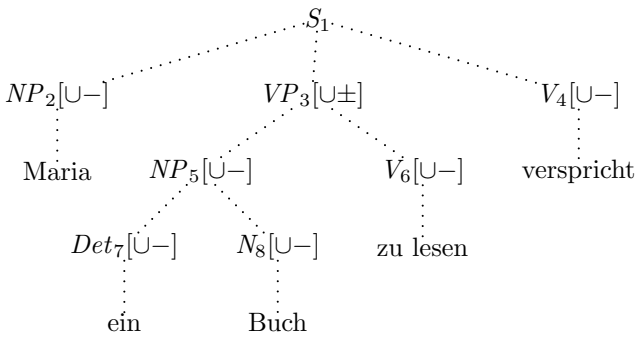


Figure 10: ID tree for sentence (39)

First of all, we enumerate the possible dom-values of node NP_5. It has two lexical daughters: Det_7 and N_8. As these are [∪−], we arrive at:

(40) $\mathsf{dom}(NP_5) \in \uparrow Det_7 \bullet \uparrow N_8 = [Det_7] \bullet [N_8] = \{[Det_7, N_8], [N_8, Det_7]\}$

The second sequence $[N_8, Det_7]$ is excluded by the linear precedence constraint Det $\prec$ N (37). Thus we end up with $\mathsf{dom}(NP_5) = [Det_7, N_8]$. (41) shows the corresponding LP tree in labeled bracketed string notation:

(41) $[[\text{ein}]_{\text{Det}}\ [\text{Buch}]_{\text{N}}]_{\text{NP}}$

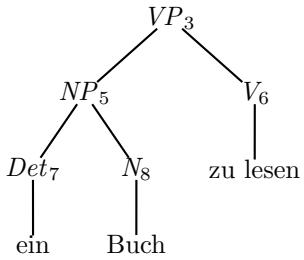
Next, we compute the dom-value of VP_3. Its daughters are NP_5 and V_6. Both are $[\cup -]$, and we arrive at:

(42) $\mathsf{dom}(\,VP_3) \in [NP_5] \bullet [\,V_6] = \{[NP_5,\, V_6], [\,V_6, NP_5]\}$

The second sequence $[\,V_6, NP_5]$ is ruled out by the linear precedence constraint NP $\prec$ V (36), thus leaving us with $[NP_5,\, V_6]$, which is depicted below in labeled bracketed string and tree notation:

(43) $[[[\text{ein}]_{\text{Det}}\ [\text{Buch}]_{\text{N}}]_{\text{NP}}\ [\text{zu lesen}]_{\text{V}}]_{\text{VP}}$

(44)



So far, the above LP subtree is isomorphic to the corresponding ID subtree from which it has been constructed. This changes at S_1, whose daughters are NP_2, VP_3 and V_4. NPs such as NP_2 and lexical nodes as V_4 must be *inserted*, but VPs such as VP_3 can be either *inserted* or *merged*. Hence these are the remaining choices for computing $\mathsf{dom}(S_1)$:

(45) $\mathsf{dom}(S_1) \in [NP_2] \bullet [\,VP_3] \bullet [\,V_4]$

(46) $\mathsf{dom}(S_1) \in [NP_2] \bullet \mathsf{dom}(\,VP_3) \bullet [\,V_4] = [NP_2] \bullet [NP_5,\, V_6] \bullet [\,V_4]$

In order to obtain a linearization where "Maria" (NP_2) occurs between "ein Buch" (NP_5) and "zu lesen"(V_6), we must *merge* VP_3 as in (46). Thus VP_3 must be $[\cup +]$, and we arrive at:

(47) $\mathsf{dom}(S_1) = [NP_5, NP_2,\, V_6,\, V_4]$

Intuitively, VP_3 must be *merged* rather than *inserted* because it represents a *discontinuous* constituent.

Finally, here is the LP tree at S_1, which is isomorphic to the flat tree given in Figure 9 (and repeated below in Figure 11):

(48) $[[[\text{ein}]_{\text{Det}}\ [\text{Buch}]_{\text{N}}]_{\text{NP}}\ [\text{Maria}]_{\text{NP}}\ [\text{zu lesen}]_{\text{V}}\ [\text{verspricht}]_{\text{V}}]_{\text{S}}$

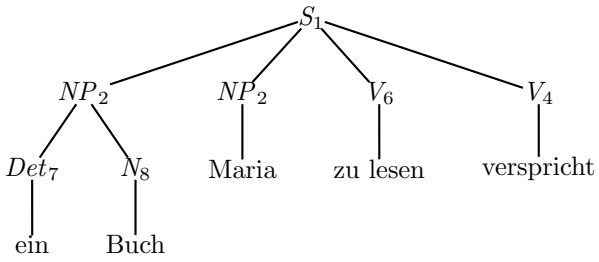


Figure 11: LP tree for sentence (39)

4.9 VP extraposition

In this section, we explain how Reape's proposal accounts for VP extraposition, as in sentence (49) below:

(49) *(dass) Maria verspricht, ein Buch zu lesen.*
(that) Maria promises, a book to read.
"(that) Maria promises to read a book."

The ID tree (Figure 12) for (49) is the same as in the example above, except for the unioned-value of VP_3. VP_3 represents an extraposed constituent, and by Reape's language specific principle (35), it is therefore $[\cup -]$.

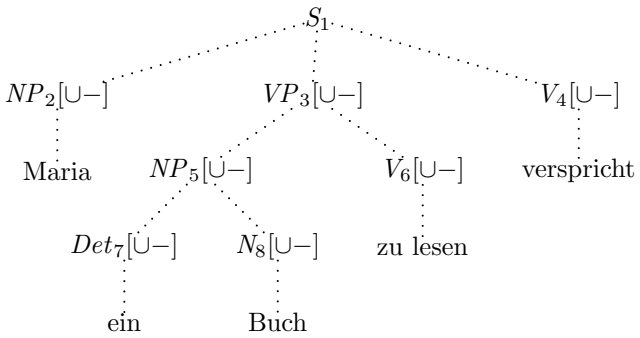


Figure 12: ID tree for sentence (49)

The domains of the nodes NP_5 and VP_3 are the same as above:

(50) $\mathsf{dom}(NP_5) = [Det_7, N_8]$

(51) $\mathsf{dom}(VP_3) = [NP_5, V_6]$

And VP_3 must be *inserted* into the domain of its mother because it is $[\cup -]$, as are the two other daughters of S_1, NP_2 and V_4:

(52) $\mathsf{dom}(S_1) \in [NP_2] \bullet [VP_3] \bullet [V_4]$

The one sequence generated by this equation that is the linearization of (49) is $[NP_2, V_4, VP_3]$. Its corresponding LP tree is shown below in labeled bracketed string and tree notation (Figure 13):

(53) $[[\text{Maria}]_{\text{NP}}\ [\text{verspricht}]_{\text{V}}\ [[[\text{ein}]_{\text{Det}}\ [\text{Buch}]_{\text{N}}]_{\text{NP}}\ [\text{zu lesen}]_{\text{V}}]_{\text{VP}}]_{\text{S}}$

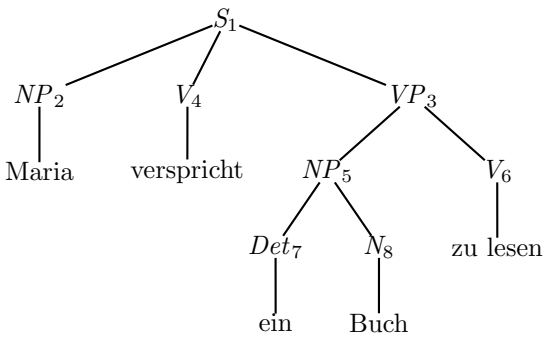


Figure 13: LP tree for sentence (49)

4.10 Problems with Reape's analysis

We show in this section that Reape's theory of word order domains as presented in the preceding sections cannot account for phenomena such as partial VP extraposition and relative clause extraposition. We argue that Reape's unioned-distinction is too crude to capture all the subtleties of word order variation in German, which is why we attempt to formulate a refined distinction in our notion of word order domains (see section 5).

4.10.1 Partial VP extraposition

The first problem with Reape's analysis concerns partial VP extraposition. The outstanding feature of partial VP extraposition is that some arguments of the extraposed verb are extraposed and some remain in their canonical positions. An example is shown below:

(54) *(dass) Maria ein Buch verspricht, zu lesen.*
(that) Maria a book promises to read.
"(that) Maria promises to read a book."

Reape's theory cannot generate an analysis for this sentence, i.e. his analysis predicts (54) is unacceptable. Why?

The ID tree for this sentence is again the same as in the example above (Figure 12). The problem in analysing (54) comes up when we construct the dom-value of S_1. As extraposed VPs are $[\cup -]$ by Reape's language specific principle (35), here is the dom-value we get at S_1:

(55) $\text{dom}(S_1) \in [NP_2] \bullet [VP_3] \bullet [V_4] =$
$\{[NP_2, V_4, VP_3], [NP_2, VP_3, V_4], [VP_3, NP_2, V_4]\}$

But none of these sequences generates the linearization of (54). We would need a sequence such as:

(56) $[NP_2, NP_5, V_4, V_6]$

To get (56), we would have to *merge* VP_3 into the domain of S_1. That is, the language specific principle (35) requiring extraposed constituents to be [∪−] would have to be refined in order to distinguish between fully and partially extraposed VPs.

4.10.2 Relative clause extraposition

Another problem with Reape's *language specific principles* comes up when we analyse sentences exhibiting relative clause extraposition:

(57) *(dass) Maria einen Mann liebt, der schläft.*
(that) Maria a man loves, who sleeps.
"Maria loves a man who sleeps."

Figure 14 shows the ID tree that we assume for sentence (57).

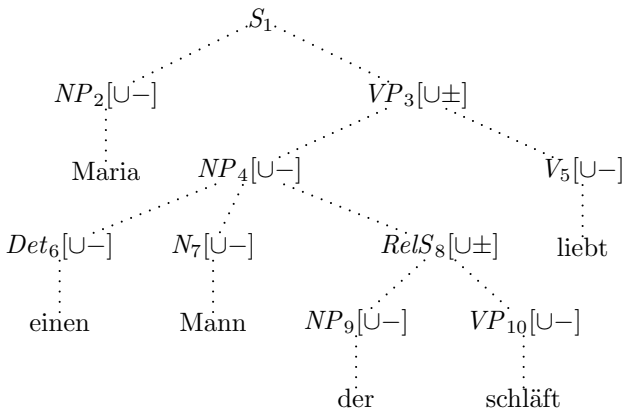


Figure 14: ID tree for sentence (57)

(58) and (59) show the respective dom-values for NP_4, depending on whether we *insert* or *merge* $RelS_8$ into the domain of NP_4.

(58) $\mathsf{dom}(NP_4) \in [Det_6] \bullet [N_7] \bullet [RelS_8] = [Det_6, N_7, RelS_8]$

(59) $\mathsf{dom}(NP_4) \in [Det_6] \bullet [N_7] \bullet \mathsf{dom}(RelS_8) = [Det_6] \bullet [N_7] \bullet [NP_9, VP_{10}] = \{[Det_6, N_7, NP_9, VP_{10}], [Det_6, NP_9, N_7, VP_{10}]\}$

Reape's approach predicts that (57) is unacceptable because regardless of whether we choose (58) or (59), the language specific principles require that NP_4 must be *inserted*:

(60) $\mathsf{dom}(VP_3) \in [NP_4] \bullet [V_5] = [NP_4, V_5]$

But this sequence does not produce the linearization of (57). What we would want Reape's theory to generate is a sequence where "liebt" (V_5) lands in between "einen Mann" (NP_4) and "der schläft" ($RelS_8$).

5 A new formulation of word order domains

As already mentioned in section 2, there is as yet no theory of linear precedence worked into Duchier's (1999) axiomatization of dependency parsing. We think that Reape's (e.g. Reape 1994) notion of word order domains can be incorporated into Duchier's axiomatization to yield such a theory of linear precedence.

However, as Reape's approach presumes an underlying phrase structure grammar framework, we must adapt Reape's notion of word order domains for dependency grammar. In doing so, we refine Reape's approach to account for phenomena like partial VP extraposition and relative clause extraposition, both of which Reape's theory does not treat properly (see section 4.10). This section gives a first sketch of our theory of linear precedence, which we call the *LP framework*.

5.1 ID and LP trees

Like Reape, we consider two analysis trees:

- An *unordered* tree for describing the immediate dominance part of the grammar. We call it *ID tree*.
- A *partially ordered* tree for describing the linear precedence part of the grammar. We call it *LP tree*.

ID trees are dependency trees as described in section 2. The edges of these trees are labeled by *role types*.

LP trees are also labeled trees, but contrary to ID trees, the set of labels of LP trees is *totally ordered*. LP trees are partially ordered, since equally labeled siblings are not ordered with respect to each other. This partial ordering characterizes the licensed linearizations of a sentence.

We distinguish two kinds of nodes in LP trees, viz. leaf nodes and non-leaf nodes. We call the former *ID nodes* and the latter *LP nodes* and consider a bijection from ID nodes to LP nodes. The set of ID nodes in the LP tree is precisely the set of nodes of the ID tree.

5.2 The intuition

Here is a first example of our notions of ID trees and LP trees. Consider the subordinate sentences (61) and (62) below:

(61) *(dass) Hans Maria liebt.*
(that) Hans Maria loves.
"(that) Hans loves Maria."

(62) *(dass) Maria Hans liebt.*
(that) Maria Hans loves.
"(that) Hans loves Maria."

Which ID and LP trees analyses does our theory produce for these sentences? First of all, since ID trees are unordered, our theory postulates only one ID tree for both sentences, which is depicted in Figure 15. We draw ID tree edges dotted to indicate that they are unordered trees.

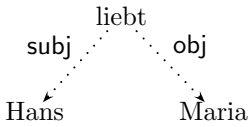


Figure 15: ID tree for (61) and (62)

We also get only one LP tree for both (61) and (62), depicted in Figure 16. Note that we draw LP tree edges squiggly, indicating that they are partially ordered trees.

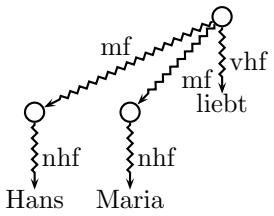


Figure 16: LP tree for (61) and (62)

For the example, we consider a set $\mathcal{F} = \{\mathsf{mhf}, \mathsf{mf}, \mathsf{vhf}\}$ of labels for LP trees. We assume the following total order on $\mathcal{F}$:

$$\mathsf{mhf} \prec \mathsf{mf} \prec \mathsf{vhf}$$

Since two edges emanate from the root node in Figure, both labeled with mf, the relative order of "Hans" and "Maria" remains unspecified and the tree is hence not totally but partially ordered. Figure 16 licenses two linearizations, one where "Hans" precedes "Maria" and one where "Maria" precedes "Hans". Both must appear left of "liebt" however, since mf precedes vhf in the total order on $\mathcal{F}$.

Notice how LP nodes are in bijective correspondence with the ID nodes. We introduce a new notation for LP trees to make this correspondence clearer. In the new notation, the subscripts LP and ID indicate whether a node is an ID node or an LP node. Figure 16 in the new notation is depicted in Figure 17.

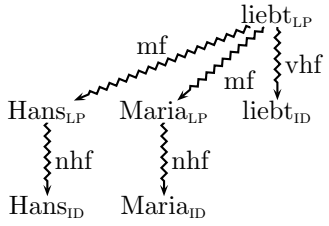


Figure 17: LP tree for (61) and (62), new notation

5.3 Tree structures

We now turn to making the above intuitions more precise. We begin with formalizing ID and LP tree structures. An ID tree T_{ID} is defined as:

$$T_{\text{ID}} = \langle (\mathcal{V}_{\text{ID}}, \mathcal{E}_{\text{ID}}), \mathsf{entry} \rangle$$

as in section 2, where $\mathcal{V}_{\text{ID}}$ is the set of ID nodes. We write w to denote ID nodes. The set of edges of the ID tree, labeled with role types $\rho \in$ Roles, is defined as:

$$\mathcal{E}_{\text{ID}} \subseteq \mathcal{V}_{\text{ID}} \times \mathcal{V}_{\text{ID}} \times \text{Roles}$$

We write $w \xrightarrow{\rho} w'$ for $\langle w, w', \rho \rangle \in \mathcal{E}_{\text{ID}}$. entry is a function assigning a lexical entry to each node.

For the LP tree, we assume a set of nodes $\mathcal{V}_{\text{LP}}$, called *LP nodes*, disjoint from $\mathcal{V}_{\text{ID}}$, and write μ to denote LP nodes. A bijection $\text{LP} : \mathcal{V}_{\text{ID}} \mapsto \mathcal{V}_{\text{LP}}$ maps ID nodes to LP nodes. The inverse function LP^{-1} is $\text{ID} : \mathcal{V}_{\text{LP}} \mapsto \mathcal{V}_{\text{ID}}$.

We pose $\mathcal{V}_{\text{ID/LP}} = \mathcal{V}_{\text{ID}} \cup \mathcal{V}_{\text{LP}}$ and define the LP-tree as

$$T_{\text{LP}} = (\mathcal{V}_{\text{ID/LP}}, \mathcal{E}_{\text{LP}})$$

where *LP edges* $\in \mathcal{E}_{\text{LP}}$ are labeled by *field types* from the set of field types $\mathcal{F}$. $\mathcal{F}$ is the disjoint union $\mathcal{H} \uplus \mathcal{D}$ of the set $\mathcal{H}$ of *head field types* and the set $\mathcal{D}$ of *daughter field types*.

The set $\mathcal{E}_{\text{LP}}$ of LP edges is defined as:

$$\mathcal{E}_{\text{LP}} \subseteq (\mathcal{V}_{\text{LP}} \times \mathcal{V}_{\text{ID}} \times \mathcal{H}) \cup (\mathcal{V}_{\text{LP}} \times \mathcal{V}_{\text{LP}} \times \mathcal{D})$$

We write $\mu \overset{h}{\rightsquigarrow} w$ for an edge from the set $\mathcal{V}_{\text{LP}} \times \mathcal{V}_{\text{ID}} \times \mathcal{H}$ and $\mu \overset{d}{\rightsquigarrow} \mu'$ for an edge from $\mathcal{V}_{\text{LP}} \times \mathcal{V}_{\text{LP}} \times \mathcal{D}$. The set of all edges emanating from an LP node forms the node's *field structure*.

We call a mother in the ID tree *ID mother* and a mother in the LP tree *LP mother*. To ensure that $\text{LP}(w)$ is the LP mother of w, we stipulate the following well-formedness constraint:

$$\forall w \in \mathcal{V}_{\text{ID}} \quad \mu \overset{h}{\rightsquigarrow} w \in \mathcal{E}_{\text{LP}} \;\Rightarrow\; \text{LP}(w) = \mu$$

5.4 Licensing conditions

What LP trees should be licensed by our theory? Since ID nodes are in bijection with LP nodes, the major question is how to connect the LP nodes together. We address this question by axiomatizing what we call *licensing conditions*. They consist of two parts:

1. which nodes to connect
2. with what label to connect them

In order to axiomatize which nodes can be connected, we employ the notions of *transitive heads* and *barriers*, which we explicate in section 5.5.

We use the concepts of *landing fields* and *field valency* to axiomatize with what labels we connect nodes in the LP tree. Landing fields and field valency are developed in section 5.6.

5.5 Transitive heads and barriers

We introduce a notion of *landing* which applies to ID nodes. When μ is the LP mother of μ', we say that $\textsc{id}(\mu')$ *lands on* $\textsc{id}(\mu)$. We call $\textsc{id}(\mu)$ the *landing site* of $\textsc{id}(\mu')$.

We require the landing site $\textsc{id}(\mu)$ of $\textsc{id}(\mu')$ to be a *transitive head* of $\textsc{id}(\mu')$:

$$\textsc{id}(\mu) \xrightarrow{\rho_1} w_1 \xrightarrow{\rho_2} \ldots \xrightarrow{\rho_n} w_n \xrightarrow{\rho} \textsc{id}(\mu')$$

When a node does not land on its ID mother, we say that it is *extracted*. Extraction of nodes corresponds to the *merge*-case in Reape's theory, and non-extraction to *insert*.

We control extraction of nodes by introducing the notion of barrier nodes or *barriers*. If a transitive head $\textsc{id}(\mu)$ of $\textsc{id}(\mu')$ is a barrier, $\textsc{id}(\mu')$ cannot be extracted beyond $\textsc{id}(\mu)$. Specifying barriers is a similar notion to Reape's language specific principles, which specify nodes of certain categories to be $[\cup -]$. In our theory however, which nodes act as barriers depends on the role ρ filled by the ID node $\textsc{id}(\mu')$ considered for extraction, not on the ID node's category. By employing this notion of barriers, we can distinguish what dependents can be extracted or not on a per role basis. This also allows us to handle cases that were problematic for Reape, e.g. relative clause extraposition (see section 5.11 below).

When we take the concepts of *transitive heads* and *barriers* together, an ID node $\textsc{id}(\mu')$ must land on a transitive head $\textsc{id}(\mu)$:

$$\textsc{id}(\mu) \xrightarrow{\rho_1} w_1 \xrightarrow{\rho_2} \ldots \xrightarrow{\rho_n} w_n \xrightarrow{\rho} \textsc{id}(\mu')$$

and none of w_i $(1 \leq i \leq n)$ are barriers for role ρ. We can express this as a constraint, but refrain from presenting it in the proposal to keep things shorter.

As an illustration of this notion of barriers consider the following. We want to ensure that determiners are not extracted but land in the field structure of their ID mother (which presumably is a noun). Therefore, we posit that nouns are barriers for words filling the det-role. Nouns are barriers for adjectives (adj), too, but they are not barriers for relative clauses (rels) since the latter can optionally be extraposed.

5.6 Landing fields and field valency

In addition to the notions of *transitive head* and *barriers*, we introduce the concepts of *landing fields* and *field valency*. Roughly, *landing fields* specify which fields are *permitted* for an ID node to land into, and *field valency* specifies which fields are *available* to land into.

Landing fields depend on the role ρ filled by the landing ID node. We capture this notion by introducing the function landingfields:

$$\mathsf{landingfields} : \text{Roles} \mapsto 2^{\mathcal{D}}$$

Here is its definition:

$$w \xrightarrow{\rho} w' \in \mathcal{E}_{\text{ID}} \quad \Rightarrow \quad \forall \mu \overset{d}{\rightsquigarrow} \textsc{lp}(w') \in \mathcal{E}_{\text{LP}} \quad \Rightarrow \quad d \in \mathsf{landingfields}(\rho)$$

In addition, we use the concept of *field valency* to specify what fields are available to land into. Field valency depends on the category of the LP mother

μ of another LP node μ'. We consider the function fieldvalency to capture this notion:

$$\mathsf{fieldvalency} : \mathcal{V}_{\text{LP}} \mapsto 2^{\mathcal{F}}$$

An LP edge $\mu \overset{f}{\rightsquigarrow} \mu'$ is licensed if f is in the field valency of μ:

$$\mu \overset{f}{\rightsquigarrow} \mu' \in \mathcal{E}_{\text{LP}} \quad \Rightarrow \quad f \in \mathsf{fieldvalency}(\mu)$$

We introduce an auxiliary function licensedfields as a mapping Cats $\mapsto 2^{\mathcal{F}}$ from categories to sets of fields, and use this function in the definition of fieldvalency below:

$$\mathsf{fieldvalency}(\mu) = \mathsf{licensedfields}(\mathsf{cat}(\textsc{id}(\mu)))$$

5.7 An example grammar

In this section, we specify an concrete instance of our theory of word order for a fragment of German. First, we specify the set $\mathcal{H} = \{\mathsf{mhf}, \mathsf{nhf}, \mathsf{vhf}\}$ of head field types. Table 1 illustrates what the names of the head field types mean:

head field type	meaning
mhf	miscellaneous head field
nhf	noun head field
vhf	verb head field

Table 1: Head field types

Second, we specify the set $\mathcal{D} = \{\mathsf{vf}, \mathsf{detf}, \mathsf{adjf}, \mathsf{mf}, \mathsf{relf}, \mathsf{vcf}, \mathsf{nf}\}$ of daughter field types. Table 2 illustrates what the names of the daughter field types mean.

daughter field type	meaning
vf	Vorfeld
detf	determiner field
adjf	adjective field
mf	Mittelfeld
relf	relative clause field
vcf	verb cluster field
nf	Nachfeld

Table 2: Daughter field types

The set $\mathcal{F}$ of field types $\mathcal{F} = \mathcal{H} \uplus \mathcal{D}$ is totally ordered:

$$\mathsf{vf} \prec \mathsf{chf} \prec \mathsf{detf} \prec \mathsf{adjf} \prec \mathsf{mhf} \prec \mathsf{nhf} \prec \mathsf{mf} \prec \mathsf{relf} \prec \mathsf{vcf} \prec \mathsf{vhf} \prec \mathsf{nf}$$

Figure 3 defines the landingfields-function and Figure 4 the licensedfields-function.

In addition, we stipulate that if an infinite verb (category vzu) is in canonical position, i.e. lands in a vcf-field, then its field valency contains only headfields:

(63) $\mu \overset{\mathsf{vcf}}{\rightsquigarrow} \mu' \quad \Rightarrow \quad \mathsf{fieldvalency}(\mu') \subseteq \mathcal{H}$

$\rho \in$ Roles	landingfields(ρ)
det	{detf}
adj	{adjf}
subj, obj, dative	{vf, mf}
vp_zu	{vf, vcf, nf}
rels	{relf, nf}

Table 3: landingfields definition

$c \in$ Cats	licensedfields(c)
det	{mhf}
adj	{mhf}
n	{detf, adjf, nhf, relf}
vzu	{mf, vcf, vhf, nf}
vfin	{vf, chf, mf, vcf, vhf, nf}

Table 4: licensedfields definition

We call constraint (63) *vcf-constraint*. It ensures that our theory does not generate LP trees like those shown in Figure 18. Figure 18 would license the unacceptable linearization "(dass) Maria zu versprechen ein Buch zu lesen versucht.". Please notice that constraints like the vcf-constraint will be worked out more thoroughly in the diploma thesis itself.

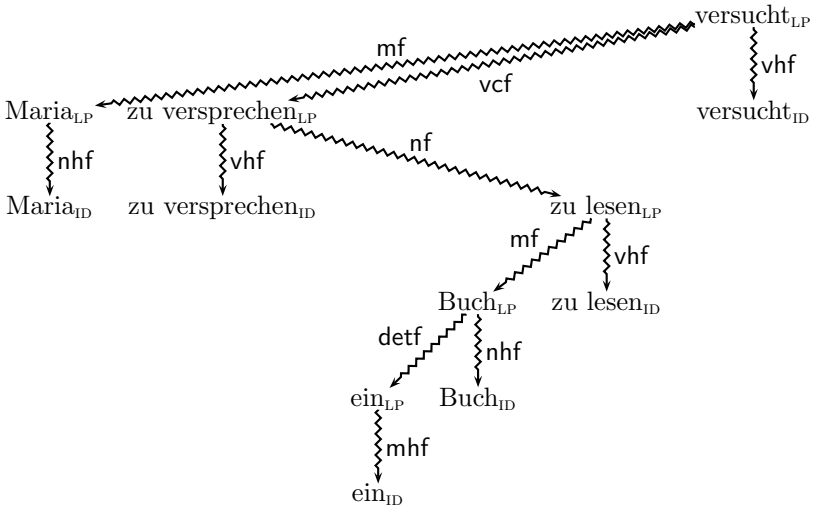


Figure 18: LP tree excluded by the vcf-constraint (63)

5.8 Scrambling

Now we turn to testing our theory on several phenomena of the German language, starting with *scrambling*:

(64) *(dass) Maria ein Buch zu lesen verspricht.*
(that) Maria a book to read promises.
"(that) Maria promises to read a book."

(65) *(dass) ein Buch Maria zu lesen verspricht.*
(that) a book Maria to read promises.
"(that) Maria promises to read a book."

Figure 19 depicts the ID tree we assume for both (64) and (65).

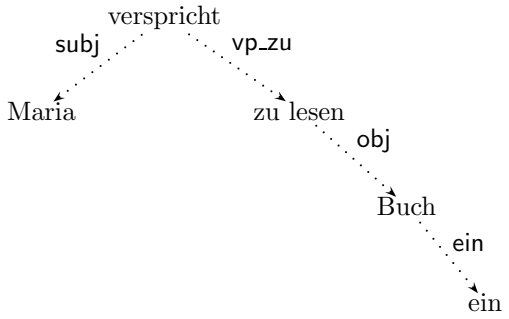


Figure 19: ID tree for (64) and (65)

Now to the corresponding LP tree analysis, given in Figure 20. As both nouns "Maria" and "Buch" land in the Mittelfeld mf, Figure 20 licenses both linearizations (64) and (65). Notice that Figure 20 is not the only LP tree that is licensed for the ID tree in Figure 19. In sections 5.9 and 5.10 we elaborate on the other two.

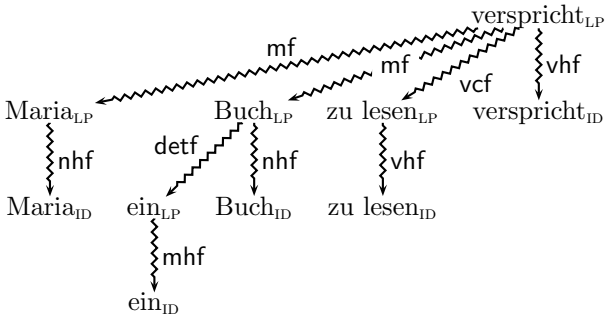


Figure 20: LP tree for (64) and (65)

5.9 VP extraposition

Now we turn to VP extraposition. The sentence of concern is (66):

(66) *(dass) Maria verspricht, ein Buch zu lesen.*
(that) Maria promises, a book to read.
"(that) Maria promises to read a book."

Figure 21 is another LP tree that is licensed for the same ID tree as in Figure 19 above. Figure 21 correctly produces (66) as the only licensed linearization.

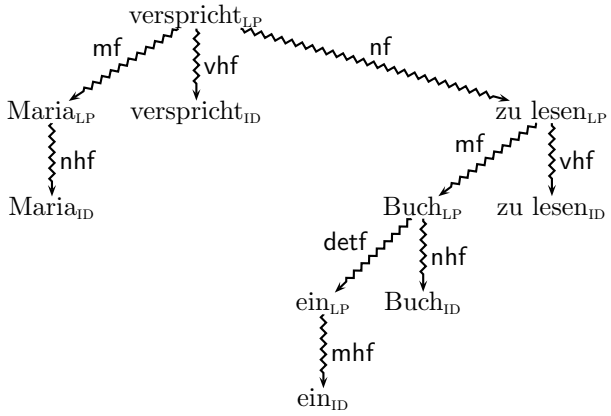


Figure 21: LP tree for (66)

5.10 Partial VP extraposition

Contrary to Reape's, our theory can also produce an analysis that yields the linearization (67):

(67) *(dass) Maria ein Buch verspricht, zu lesen.*
(that) Maria a book promises, to read.
"(that) Maria promises to read a book."

Figure 22 is an LP tree that produces this linearization[8]. It is licensed for the ID tree given in Figure 19.

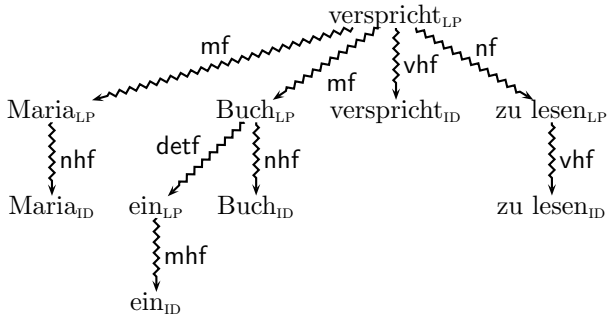


Figure 22: LP tree for (67)

[8]Observe that Figure 22 also licenses another (acceptable) linearization, viz. "(dass) ein Buch Maria verspricht zu lesen.".

5.11 Relative clause extraposition

Relative clause extraposition is one of the phenomena for which Reape's approach does not correctly account. Reape doesn't allow NPs to be discontinuous (i.e. $[\cup -]$) by one of his *language specific principles*. But that requirement prevents anything from slipping in between the noun and its modifying relative clause in an NP. In (68) for instance, the verb "liebt" slips in between "einen Mann" and "der schläft" in the NP "einen Mann, der schläft" to yield an acceptable German sentence. The non-extraposed version of (68) is given in (69).

(68) *(dass) Maria einen Mann liebt, der schläft.*
(that) Maria a man loves, who sleeps.
"Maria loves a man who sleeps."

(69) *(dass) Maria einen Mann, der schläft, liebt.*
(that) Maria a man, who sleeps, loves.
"Maria loves a man who sleeps."

Contrary to Reape's theory, our theory can not only produce the linearization (69) but also the one in (68). Figure 23 depicts the ID tree analysis for (68) and (69).

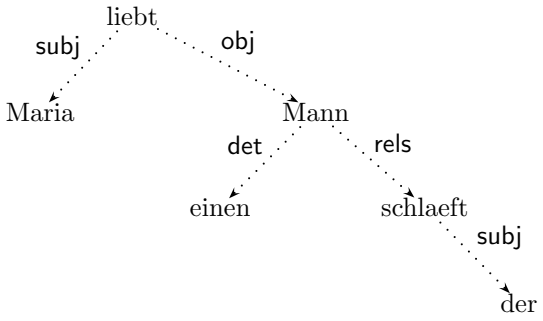


Figure 23: ID tree for (68) and (69)

Relative clauses can either be extraposed or not. In other words, a relative clause can either land in the Nachfeld of a verb (field type nf) or in the relative clause field of a noun (field type relf) (see Table 3 above). Field valency ensures that relf-field types can only appear on nouns, and nf-field types on verbs (Table 4). Additionally, nouns are a barrier for determiners (det-role) but not for relative clauses (rels-role). We further prevent the extraction of dependents of relative clause's finite verb by stipulating that finite verbs are barriers.

We arrive at two legal LP trees[9] for the extraposed and non-extraposed case, depicted in Figure 24 and Figure 25 respectively.

[9]Notice that these trees actually license two linearizations each, one where "Maria" precedes "Mann" and one where "Mann" precedes "Maria".

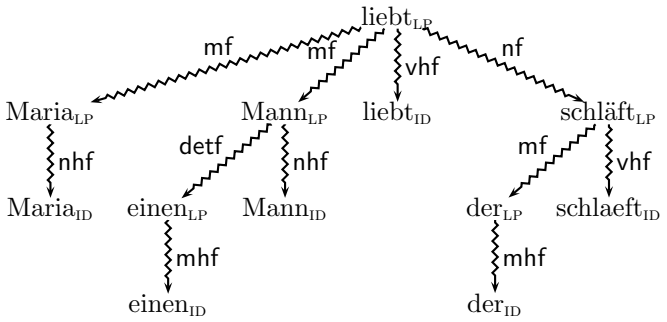


Figure 24: LP tree for (68)

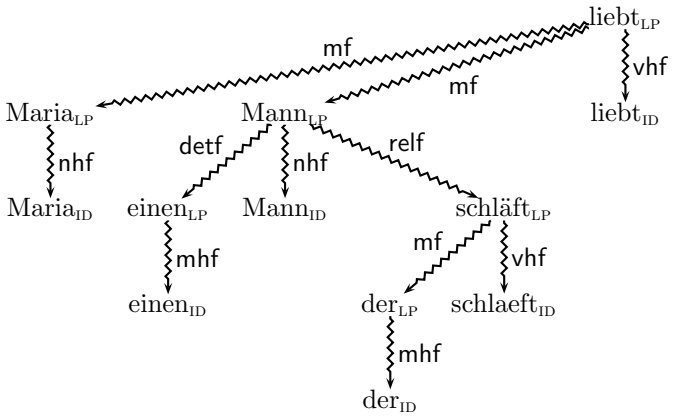


Figure 25: LP tree for (69)

A Goals

1. Development of a grammar formalism for Duchier's (1999) dependency parser, without constraining word order. Duchier's dependency parser and the grammar formalism form what we call the *ID framework*. We believe that this part is the easier one, since we know already how to do it.

   - Language-independent part
     (a) Specification of a typed lexicon (see sections 3.1 and 3.2).
     (b) Specification of a language for expressing role constraints (see section 3.3).
     (c) Implementation of a parser generator for grammars written in the grammar formalism using the *Gump* parser generator.
   - Language-dependent part
     (a) Example grammar for German. Unconstrained word order makes for a good coverage but also for overgeneration.

(b) Development of a language to add language-dependent constraints as modules to our dependency grammars.

2. Extension of Duchier's (1999) proposal with a theory of word order domains and fields, called the *LP framework*, to constrain linear precedence.

   and will probably take longer to work out.

   - Language-independent part
     (a) Constraint-based axiomatization of our notion of word order domains and fields (for a first sketch, see section 5).
     (b) Implementation of our notion of word order domains and fields within Duchier's dependency parser.
   - Language-dependent part
     (a) Example grammar for German (see also section 5). Coverage to include several non-trivial phenomena such as
        i. scrambling
        ii. partial verb phrase topicalisation and extraposition
        iii. relative clause extraposition

3. Options
   (a) Integration of CLLS semantics into the dependency parser.
   (b) Implementation of a graphical user interface for the ID framework.
   (c) Development of a notation for declaratively formulating constraints on word order.

References

Duchier, D. (1999), Axiomatizing dependency parsing using set constraints, *in* 'Sixth Meeting on Mathematics of Language', Orlando/FL.

Gazdar, G., Klein, E., Pullum, G. & Sag, I. (1985), *Generalized Phrase Structure Grammar*, B. Blackwell, Oxford/UK.

Kathol, A. (1995), Linearization-Based German Syntax, PhD thesis, Ohio State University.

Müller, S. (1999), *Deutsche Syntax deklarativ. Head-Driven Phrase Structure Grammar für das Deutsche*, number 394 *in* 'Linguistische Arbeiten', Max Niemeyer Verlag, Tübingen.

Netter, K. (1991), Clause union phenomena and complex predicates in German, Technical Report R1.1.B (Part 1), DYANA.

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.

Tesnière, L. (1959), *Eléments de Syntaxe Structurale*, Klincksiek, Paris/FRA.

Uszkoreit, H. (1987), *Word Order and Constituent Structure in German*, CSLI, Stanford/CA.