

A Comparative Introduction To Extensible Dependency Grammar

Denys Duchier

Ralph Debusmann

This booklet collects together some hopefully helpful reading material to complement our lectures:

1. “Approaches to linear precedence” (by Ralph Debusmann, chapter 3 of his Diploma thesis 2001 “A declarative grammar formalism for dependency grammar”)
2. “A Meta-Grammatical Framework for Dependency Grammar” (by Ralph Debusmann and Denys Duchier 2003), the first description of XDG
3. “A Relational Syntax-Semantics Interface Based on Dependency Grammar” (by Ralph Debusmann and Denys Duchier and Alexander Koller and Marco Kuhlmann and Gert Smolka and Stefan Thater for COLING 2004)
4. “Extensible Dependency Grammar: A New Methodology“ (by Ralph Debusmann and Denys Duchier and Geert-Jan Kruijff for the DG workshop at COLING 2004)
5. “A Parser System for Extensible Dependency Grammar” (by Ralph Debusmann for the Nancy/Saarbrücken workshop 2003)
6. “Topological Dependency Trees: A Constraint-Based Account of Linear Precedence” (by Denys Duchier and Ralph Debusmann for ACL 2001)
7. “Configuration Of Labeled Trees Under Lexicalized Constraints And Principles” (by Denys Duchier for the Journal of Research on Language and Computation 2003)

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).

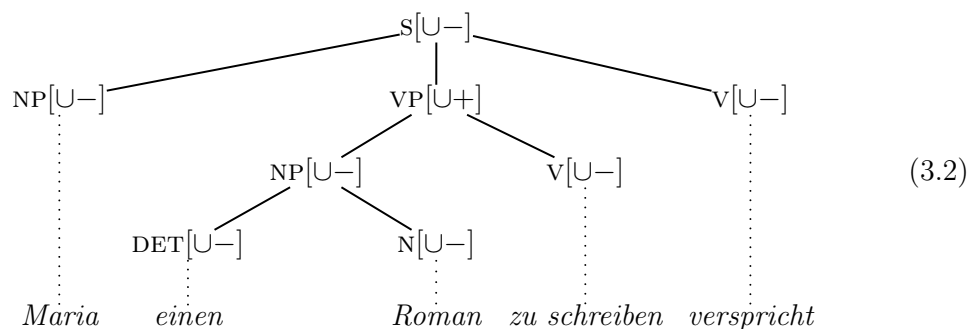
Reape's basic idea is to dissociate structures dealing with immediate dominance (ID) from those dealing with linear precedence (LP). Reape's terminology is however different: he distinguishes a level of syntax (ID) and a level of word order domains¹ (LP). In our presentation of Reape's work, we use the terms ID and LP. Each analysis is represented by two tree structures instead of one: an ID tree and

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

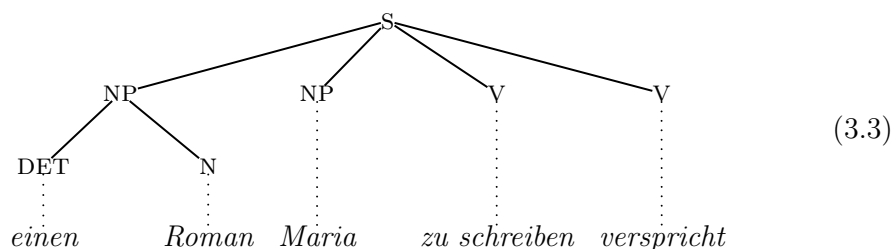
an LP tree. The ID tree is a non-ordered phrase structure analysis, and the LP tree is a flattened and ordered phrase structure analysis. As an example, consider the following German sentence:

(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:



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



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 $\langle \text{NP}, \text{NP}, \text{V}, \text{V} \rangle$ 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 $\langle \text{DET}, \text{N} \rangle$ of the determiner *einen* and the noun *Roman*. Reape constrains the order of elements in a word order domain by *linear precedence rules* like $\text{DET} \prec \text{N}$, which requires that determiners precede nouns and $\text{NP} \prec \text{V}$, 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 $[\text{U}+]$, it ‘contributes’ its daughters

²We write **unioned** = + as $[\text{U}+]$ and **unioned** = – as $[\text{U}-]$.

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 $\text{dtrs}_{\text{ID}} : V_{\text{ID}} \rightarrow 2^{V_{\text{ID}}}$ 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 $\text{dtrs}_{\text{LP}} : V_{\text{LP}} \rightarrow V_{\text{LP}}^*$ 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 $\text{dom} : V_{\text{ID}} \rightarrow V_{\text{LP}}^*$ 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 **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:

$$\cup^* : V_{\text{LP}}^* \times V_{\text{LP}}^* \rightarrow 2^{V_{\text{LP}}^*} \quad (3.4)$$

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:

$$\begin{aligned} \langle a, b \rangle \cup^* \langle c, d \rangle = \{ & \langle a, b, c, d \rangle \\ & \langle a, c, b, d \rangle \\ & \langle a, c, d, b \rangle \\ & \langle c, a, b, d \rangle \\ & \langle c, a, d, b \rangle \\ & \langle c, d, a, b \rangle \} \end{aligned} \quad (3.5)$$

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 $(s_1, \prec_{s_1}), (s_2, \prec_{s_2}) \in S^*$:

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

Contributions

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

$$\begin{aligned} v[\cup+] & \text{ for } \text{unioned}(v) = + \\ v[\cup-] & \text{ for } \text{unioned}(v) = - \\ v[\cup\pm] & \text{ for } \text{unioned}(v) = + \vee \text{unioned}(v) = - \end{aligned} \quad (3.7)$$

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

$$\text{contrib}(v) = \begin{cases} \langle v \rangle & \text{if } v[\cup-] \\ \text{dom}(v) & \text{if } v[\cup+] \end{cases} \quad (3.8)$$

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 $\text{dom}(v)$ at node v is in the set of word order domains licensed by the sequence union $\cup^*$ of the contributions of v 's daughters:

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

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

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

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

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

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

Language-specific principles

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

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

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

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 $\text{extra} : V \rightarrow \{+, -\}$ 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 $\text{dom}(v)$, 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:

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

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

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

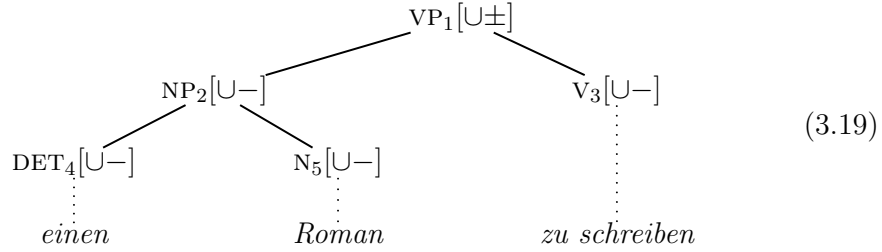
$$\text{extra}(v_1) = - \wedge \text{extra}(v_2) = + \Rightarrow v_1 \prec v_2 \quad (3.18)$$

(3.15) states that determiners precede nouns. (3.16) stipulates that NPs precede V's (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 $\text{cat} : V_{\text{ID}} \rightarrow \text{Cats}$ 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.



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 VP_1 . Proceeding from bottom to top, we start with calculating the **dom**-values at NP_2 :

$$\begin{aligned}
 \text{dom}(NP_2) &\in \text{contrib}(DET_4) \cup^* \text{contrib}(N_5) \\
 &= \langle DET_4 \rangle \cup^* \langle N_5 \rangle \\
 &= \{ \langle DET_4, N_5 \rangle, \langle N_5, DET_4 \rangle \}
 \end{aligned} \tag{3.20}$$

Both DET_4 and N_5 have been inserted into the word order domain of NP_2 because they are both $[-]$. As $\langle N_5, DET_4 \rangle$ is excluded by LP rule (3.15), the only licensed word order domain at NP_2 is $\langle DET_4, N_5 \rangle$:

$$\text{dom}(NP_2) = \langle DET_4, N_5 \rangle \tag{3.21}$$

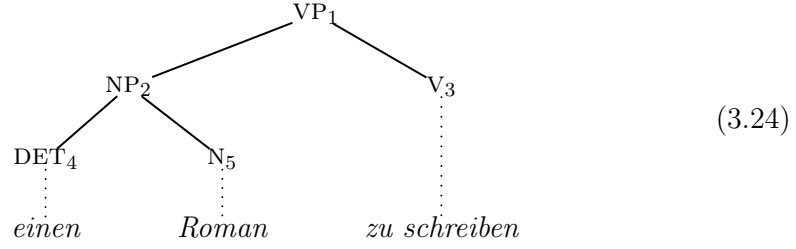
Here, we calculate the **dom**-values at root node VP_1 :

$$\begin{aligned}
 \text{dom}(VP_1) &\in \text{contrib}(NP_2) \cup^* \text{contrib}(V_3) \\
 &= \langle NP_2 \rangle \cup^* \langle V_3 \rangle \\
 &= \{ \langle NP_2, V_3 \rangle, \langle V_3, NP_2 \rangle \}
 \end{aligned} \tag{3.22}$$

Again, both nodes NP_2 and V_3 have been inserted in the domain of VP_1 . The domain $\langle V_3, NP_2 \rangle$ is excluded by LP rule (3.16), which leaves us with $\langle NP_2, V_3 \rangle$ as the only licensed word order domain at VP_1 :

$$\text{dom}(VP_1) = \langle NP_2, V_3 \rangle \tag{3.23}$$

We show the corresponding LP tree below:



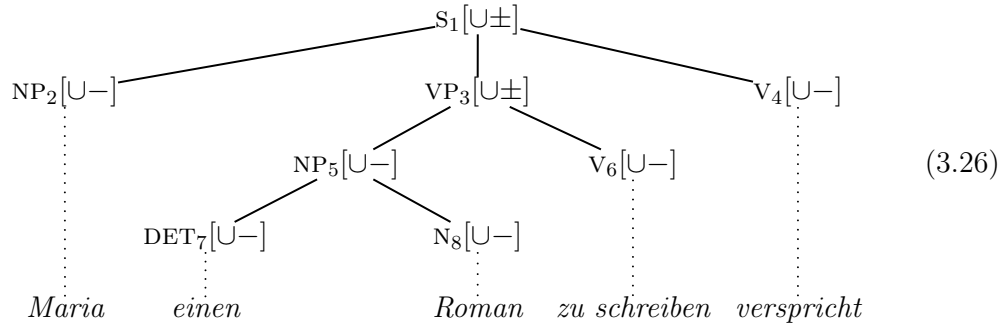
This LP tree licenses only the linearization *einen Roman zu schreiben*.

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):



LP rule (3.15) excludes sequence $\langle N_8, DET_7 \rangle$. Hence, the only licensed value for $\text{dom}(NP_5)$ is $\langle DET_7, N_8 \rangle$:

$$\text{dom}(NP_5) = \langle DET_7, N_8 \rangle \quad (3.27)$$

There is also only one possible word order domain at VP_3 :

$$\text{dom}(VP_3) = \langle NP_5, V_6 \rangle \quad (3.28)$$

since $\langle v_6, NP_5 \rangle$ 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:

$$\begin{aligned} \text{dom}(S_1) &\in \langle NP_2 \rangle \cup^* \langle VP_3 \rangle \cup^* \langle V_4 \rangle \\ &= \{ \langle NP_2, VP_3, V_4 \rangle, \langle VP_3, NP_2, V_4 \rangle, \langle NP_2, V_4, VP_3 \rangle \} \end{aligned} \quad (3.29)$$

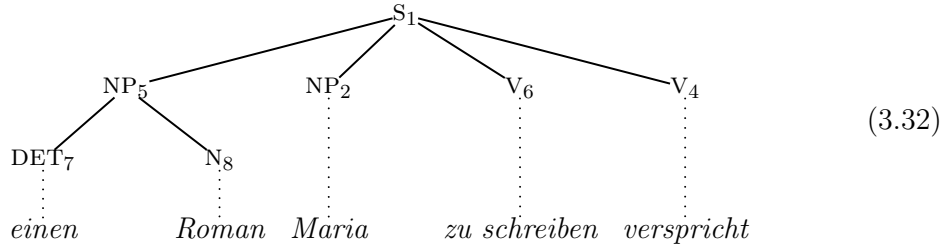
$$\begin{aligned} \text{dom}(S_1) &\in \langle NP_2 \rangle \cup^* \text{dom}(VP_3) \cup^* \langle V_4 \rangle \\ &= \langle NP_2 \rangle \cup^* \langle NP_5, V_6 \rangle \cup^* \langle V_4 \rangle \\ &= \{ \langle NP_2, NP_5, V_6, V_4 \rangle, \langle NP_5, NP_2, V_6, V_4 \rangle \} \end{aligned} \quad (3.30)$$

Only (3.30) generates a linearization where *Maria* (NP_2) occurs between *einen Roman* (NP_5) and *zu schreiben* (V_6), i.e.:

$$\langle NP_5, NP_2, V_6, V_4 \rangle \quad (3.31)$$

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 VPs, and (3.17) that *zu schreiben* (V_6) precedes *verspricht* (V_4), since $V_6 \in \text{dtrs}(V_4)$.

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



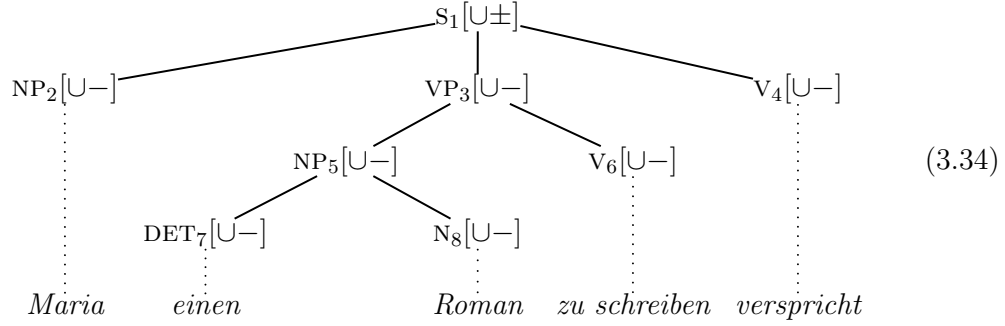
VP extraposition

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

$$\begin{aligned} &(\text{dass}) \quad \text{Maria} \quad \text{verspricht}, \quad \text{einen} \quad \text{Roman} \quad \text{zu} \quad \text{schreiben} \\ &(\text{that}) \quad \text{Maria} \quad \text{promises}, \quad \text{a} \quad \text{novel}(\text{acc}) \quad \text{to} \quad \text{write} \\ &“(that) \text{ Maria promises to write a novel}” \end{aligned} \quad (3.33)$$

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):



The word order domains at NP_5 and VP_3 are the same as in the scrambling example:

$$\text{dom}(NP_5) = \langle DET_7, N_8 \rangle \tag{3.35}$$

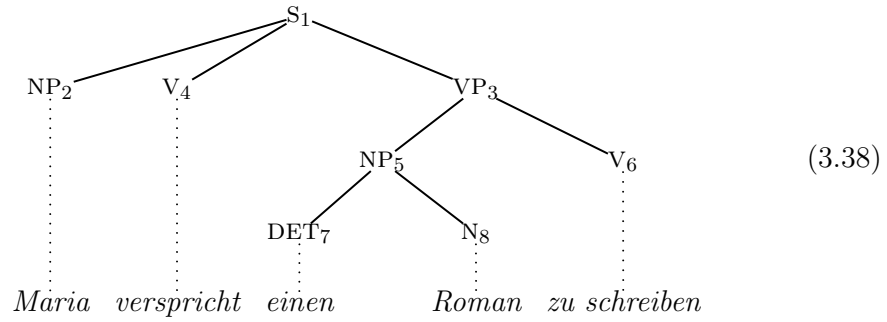
$$\text{dom}(VP_3) = \langle NP_5, V_6 \rangle \tag{3.36}$$

And VP_3 must be inserted into the domain of its mother S_1 because it is $[-]$:

$$\begin{aligned}
 \text{dom}(S_1) &\in \langle NP_2 \rangle \cup^* \langle VP_3 \rangle \cup^* \langle V_4 \rangle \\
 &= \{ \langle NP_2, VP_3, V_4 \rangle, \langle VP_3, NP_2, V_4 \rangle, \langle NP_2, V_4, VP_3 \rangle \}
 \end{aligned} \tag{3.37}$$

The other permutations $\langle V_4, NP_2, VP_3 \rangle$, $\langle V_4, VP_3, NP_2 \rangle$ and $\langle VP_3, V_4, NP_2 \rangle$ are excluded by LP rule (3.16).

Sentence (3.33) corresponds to linearization $\langle NP_2, V_4, VP_3 \rangle$, and its LP tree is shown in (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:

$$\begin{array}{llllllll}
 \text{(dass)} & \text{Maria} & \text{einen} & \text{Roman} & \text{verspricht,} & \text{zu} & \text{schreiben} \\
 \text{(that)} & \text{Maria} & \text{a} & \text{novel(acc)} & \text{promises,} & \text{to} & \text{write} \\
 \text{“(that) Maria promises to write a novel”}
 \end{array} \tag{3.39}$$

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:

$$\begin{aligned}
 \text{dom}(S_1) &\in \langle NP_2 \rangle \cup^* \langle VP_3 \rangle \cup^* \langle V_4 \rangle \\
 &= \{ \langle NP_2, VP_3, V_4 \rangle, \langle VP_3, NP_2, V_4 \rangle, \langle NP_2, V_4, VP_3 \rangle \}
 \end{aligned} \tag{3.40}$$

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:

$$vf \prec cf \prec mf \prec vc \prec nf \quad (3.42)$$

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 on 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 Heterogeneous 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.

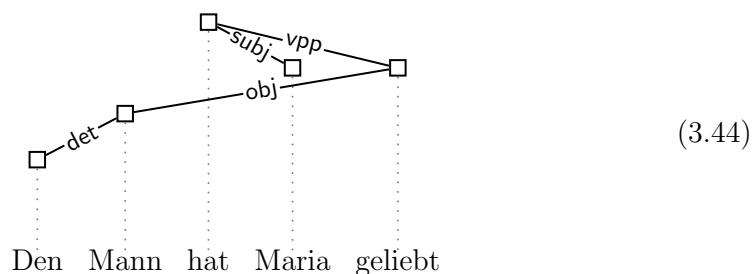
Order domain elements are assigned *features* which are later used in specifying precedence predicates that constrain surface word order. These features are used to model notions from topological fields theory: **initial** corresponds to the Vorfeld, **middle** to the Mittelfeld and **final** to the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

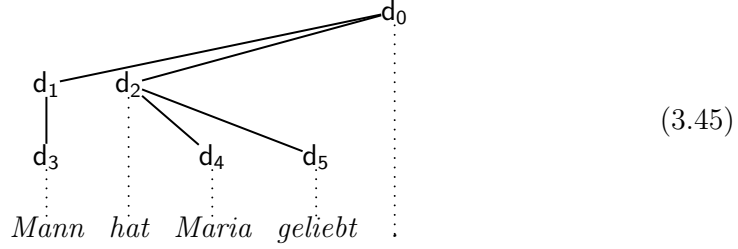
3.3.2 Example

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

Den Mann hat Maria geliebt.
 The man(*acc*) has Maria loved.
 “Maria has loved the man.”

(3.43)


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



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 $\langle d_1, d_2 \rangle$, 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:

$$\begin{aligned}
 d_0 &= \{Mann, hat, Maria, geliebt, .\} \\
 d_1 &= \{Mann\} \\
 d_2 &= \{hat, Maria, geliebt\} \\
 d_3 &= \{Mann\} \\
 d_4 &= \{Maria\} \\
 d_5 &= \{geliebt\}
 \end{aligned}
 \tag{3.46}$$

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

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

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

(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 (**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 d_2 of its direct head *hat*. *Mann* is not inserted into the domain d_5 of its direct head *geliebt*, but extracted into domain 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 $L = \{\mathbf{vpp}\}$ of dependency relations which may link its direct head with its positional head. The direct head of *Mann* is *geliebt*, and *geliebt* is a $\mathbf{vpp}$ -dependent of *hat*. Since $\mathbf{vpp} \in L$, 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:

$$\mathbf{vf}, [, \mathbf{mf},], \mathbf{nf} \quad (3.49)$$

where $\mathbf{vf}$ corresponds to the Vorfeld, $[$ to the left sentence bracket, $\mathbf{mf}$ to the Mittelfeld, $]$ to the right sentence bracket and $\mathbf{nf}$ 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

3.4.2 Example

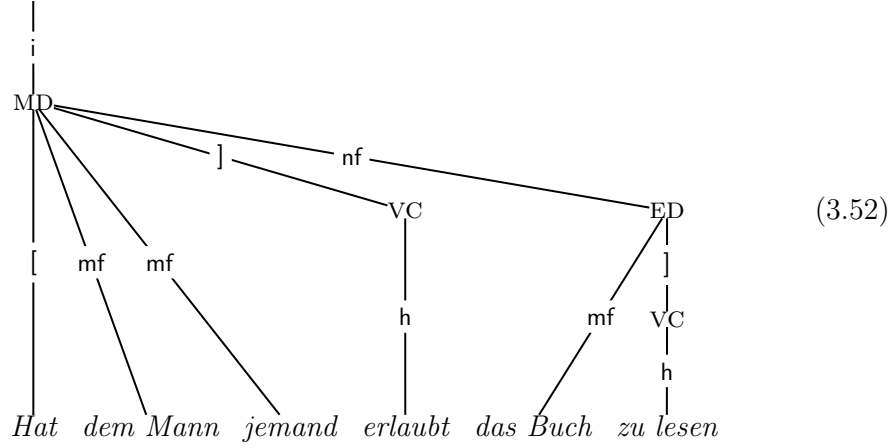
Hat dem Mann jemand erlaubt, das Buch zu lesen?
 Has the man(dat) anybody(nom) allowed, the book to read?
“Has anybody allowed the man to read the book?”

(3.50)

(3.51)

The syntactic dependency tree pictured in (3.51) above gives rise to the topological

structure shown in (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:

$$[MD Hat [dem Mann_{mf} jemand_{mf} [VC erlaubt_h]] [ED das Buch_{mf} [VC zu lesen_h]]]_{nf} i \quad (3.53)$$

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 (c, f, B, f') indicates that a word of category c placed in field f may create a domain 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 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:

$$(\mathbf{vfin}, i, \text{MD}, []) \quad (3.54)$$

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

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

$$(\mathbf{vnonfin}, \text{majf}, \text{ED}, []) \quad (3.55)$$

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 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 $B \rightarrow f_1, \dots, f_n$ indicates that domain B consists of the sequence of fields $f_1, \dots, f_n$.

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

$$\text{MD} \rightarrow \mathbf{vf}, [, \mathbf{mf},], \mathbf{nf} \quad (3.56)$$

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

$$\text{ED} \rightarrow \mathbf{mf},], \mathbf{nf} \quad (3.57)$$

Field description rules

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

Correspondence rules

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

$$(r, c_1, f_1, c_2, f_2, B) \quad (3.58)$$

Such a rule indicates that if there is a dependency edge $w_1 - r \rightarrow w_2$ 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 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 $\leq B$.

The part of the permeability order given in (Gerdes & Kahane 2001) which is relevant for our purposes (including the domains VC, ED and MD) is shown below:

$$VC < ED < MD \quad (3.59)$$

(3.59) indicates that extraction out of verb clusters (VC) is easier than extraction out of embedded domains (ED), and that in turn is easier than extraction out of main domains (MD).

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

$$(r, v, anyf, nv, majf, ED) \quad (3.60)$$

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

A Meta-Grammatical Framework for Dependency Grammar

Abstract

We propose a meta-grammatical framework for dependency grammar, accommodating any number of dimensions and restricting their licensed models through the application of parametric principles. We first instantiate this framework to obtain the version of Topological Dependency Grammar (TDG) proposed by (Duchier and Debusmann, 2001). We then describe an extended instantiation which adds support for semantic dependencies (thus also providing an account of control and raising constructions) and meaning assembly.

Topological Dependency Grammar (TDG) (Duchier and Debusmann, 2001) explains linearization phenomena through the interaction of two structures, similar to (Gerdes and Kahane, 2001): a non-ordered tree of syntactic dependencies and an ordered and projective tree of topological dependencies.

In this paper, we propose a generalized meta-grammatical framework, accommodating any number of dimensions (beyond syntax and topology) and restricting their licensed models through the application of parametric principles. First we show how this framework can be instantiated to obtain the earlier version of TDG. We then describe an extended instantiation which adds support for semantic dependencies (thus also providing an account of control and raising constructions) and meaning assembly. Figure 1 sketches briefly how the extended TDG is articulated around multiple-dimensions.

1 ID/LP TDG

In this section, we explain the basics of TDG. We regard TDG as a general grammar framework that needs to be instantiated to yield a particular grammar formalism. One instance is ID/LP TDG, as proposed in (Duchier and Debusmann, 2001).¹

ID/LP TDG is a lexicalized formulation of dependency grammar (Tesnière, 1959) that combines a non-projective account of syntax with an account of linear precedence inspired by the classical model of topological fields (Höhle, 1986). An analysis consists of two mutually constraining trees: a non-ordered tree of syntactic dependencies (ID tree) and an ordered and projective tree of topological dependencies (LP tree).

We introduce the ID/LP TDG framework using the following German example sentence:

- (1) *(dass) eine Frau jeder Mann*
(that) a woman+ACC every man+NOM
zu lieben scheint.
to love seems.
“(that) every man seems to love a woman.”

1.1 ID tree

The ID tree is a non-ordered tree of syntactic dependencies where edges are labeled with grammatical functions such as subj for subject or obj for object. Here is an ID tree analysis for example (1).²

¹“ID/LP TDG” used to be called “TDG” in (Duchier and Debusmann, 2001) and various other papers on the subject.

²For simplicity, we treat the NPs (*eine Frau* and *jeder Mann*) and the head of the VP (*zu lieben*) as independent words throughout this paper.

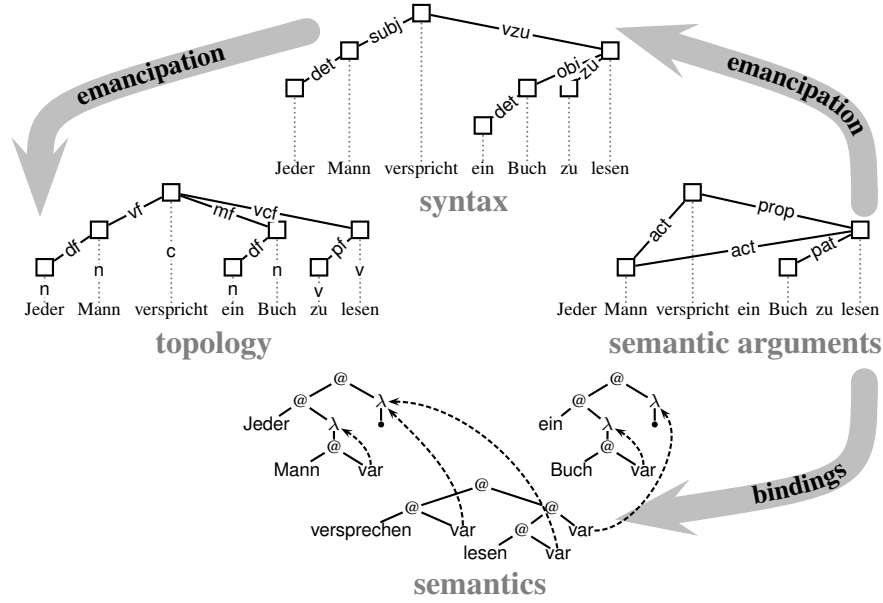
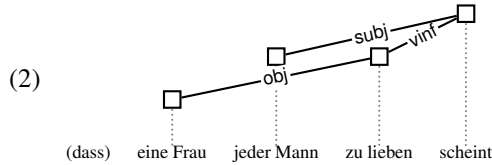


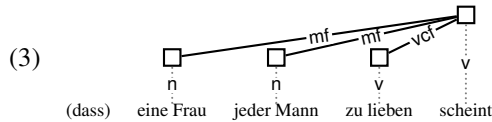
Figure 1: An architectural overview of extended TDG



jeder Mann is the subject of *scheint*, as signified by the edge labeled *subj*. Likewise, *zu lieben* is the infinitival complement (*vinf*) of *scheint*, and *eine Frau* is the object (*obj*) of *zu lieben*.

1.2 LP tree

The LP tree is an ordered and projective tree of topological dependencies. Edges of the LP tree are labeled with topological fields such as *mf* for “Mittelfeld” and *vcf* for verb field. In addition, to order nodes with respect to their dependents, we label the nodes of the LP tree by head fields such as *v* for verbs and *n* for nouns. Here is an LP tree analysis for example (1):



The two NPs *eine Frau* and *jeder Mann* both land in the Mittelfeld (*mf*) of the finite verb *scheint*, and the infinitival complement *zu lieben* lands in the verb field (*vcf*) of *scheint*.

LP trees must be well-ordered. To this end, the grammar stipulates a total order on LP edge and node labels. This imposes a linear order among the dependents of a node, and the node itself is positioned with respect to its dependents by its node label. The example grammar presented in this paper stipulates the following total order:

$$(4) \quad n \prec mf \prec vcf \prec v$$

The LP tree above is well-ordered with respect to this total order: the topological dependents of *scheint* labeled *mf* (i.e. *eine Frau* and *jeder Mann*) correctly precede the ones labeled *vcf* (*zu lieben*):

$$(5) \quad [eine\ Frau\ jeder\ Mann]_{mf} < [zu\ lieben]_{vcf}$$

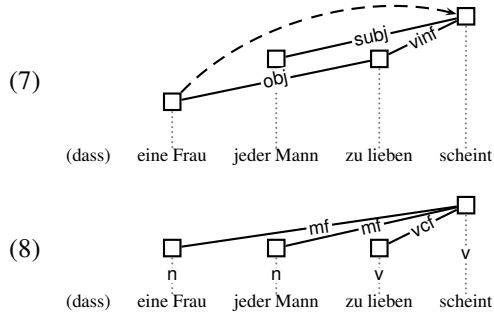
Moreover, the node *scheint* itself correctly follows all of its dependents:

$$(6) \quad [eine\ Frau\ jeder\ Mann]_{mf} < [zu\ lieben]_{vcf} < scheint_v$$

Note that *eine Frau* and *jeder Mann* have the same incoming edge label *mf*. That is, their respective order is underspecified, and either permutation (*eine Frau* preceding *jeder Mann* or the other way round) is licensed by the grammar. This reflects that in German, the order of the elements in the Mittelfeld is arbitrary.

1.3 Climbing

The ID and LP dimensions mutually constrain each other by the *climbing principle*. By the climbing principle, we allow nodes in the ID tree to migrate upwards to nodes higher in the LP tree. In our example for instance, the object *eine Frau* is the daughter of *zu lieben* in the ID tree, and of *scheint* in the LP tree. Hence we say that *eine Frau* has *climbed up* to *scheint*. For illustration, we repeat our ID and LP tree analyses, augmented with a dashed arrow which signifies the climbing up of the node corresponding to *eine Frau*:



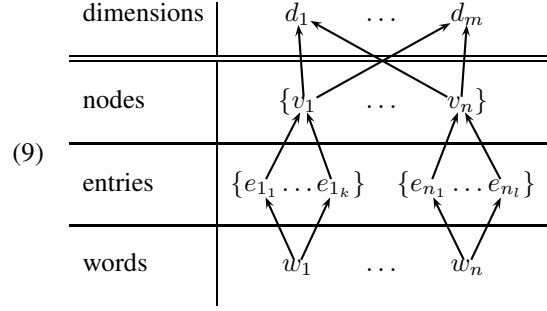
The effect of climbing is that the shape of the LP tree is a flattening of the ID tree's, similar to the effect achieved by sequence union in (Reape, 1994). This enables ID/LP TDG to handle a variety of difficult word order phenomena, especially in freer word order languages such as Dutch or German.

2 Generalized TDG

In this section, we generalize the TDG framework, starting from the ID/LP TDG grammar formalism outlined above.

2.1 Dimensions

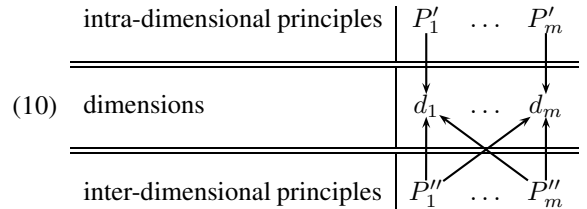
Each analysis in TDG consists of any number m of dimensions $D = \{d_1, \dots, d_m\}$ whose models are labeled directed graphs. These graphs are all made up of the same set of n nodes $V = \{v_1, \dots, v_n\}$, which correspond 1 : 1 to the words in the input sentence $W = w_1 \dots w_n$. To model lexical ambiguity, we map each word w_i to a set of i_k lexical entries $E_i = \{e_{i_1}, \dots, e_{i_k}\}$. Lexical entries are standard feature structures containing information about e.g. the subcategorization behavior. A selection function then selects one entry e_{i_j} from E_i that is eventually assigned to the node v_i corresponding to word w_i . Below, we give a sketch of this architecture:



Note that the lexical entries are *shared* across all dimensions. In fact they are one way to make the manifold dimensions mutually constrain each other.

2.2 Principles

Each dimension d_i is subject to a set P_i of *principles*, which are drawn from the same *principles pool* P . The models for dimension d_i are in the class of labeled directed graphs, and the principles further restrict their admissibility. Principles can operate only within one dimension or across several dimensions: We call principles that only operate within one dimension *intra-dimensional principles*, and those that operate across several dimensions *inter-dimensional principles*. Each set P_i of principles on dimension d_i is partitioned into subsets P'_i of intra-dimensional principles and P''_i of inter-dimensional principles: This is illustrated in the picture below.



2.3 Principles pool

ID/LP TDG draws all its principles from the following principles pool:

$$(11) P = \{\text{dag, tree, in, out, order, climbing, linking}\}$$

Each principle has arguments such as the dimensions $d_1, \dots, d_m$ on which the principle applies, or functions, taking a node in V as argument, that model lexical features.

Dag principle. *dag(d): Each analysis on dimension d is a directed acyclic graph.*

Tree principle. *tree(d): Each analysis on dimension d is a tree.*

Climbing principle. $\text{climbing}(d_1, d_2)$: A subtree on dimension d_2 may climb up and land higher up on dimension d_1 . Writing $v \rightarrow_d v'$ for an edge from v to v' on dimension d , and $v \rightarrow_d^* v'$ for its reflexive transitive closure, the climbing principle states the following constraint:

$$(12) \quad v \rightarrow_{d_1} v' \Rightarrow v \rightarrow_{d_2}^* v'$$

That is, an edge from v to v' on dimension d_1 is licensed only if v dominates v' on dimension d_2 .

Out principle. $\text{out}(d, \text{out}_d : V \rightarrow 2^{\Pi_{\mathcal{L}}})$: The outgoing edges of a node on dimension d must satisfy in label and number the stipulation of the corresponding out_d feature. This principle is parametrized by the lexical feature out_d , and $\mathcal{L}$ is the set of edge labels on dimension d . Π is a constructor for label patterns. Given a set $\mathcal{L}$ of labels, we write $\Pi_{\mathcal{L}}$ for the set of label patterns π that can be formed according to the following abstract syntax:

$$(13) \quad \pi ::= \ell \mid \ell? \mid \ell^*$$

where ℓ stands for “precisely one” outgoing edge, $\ell?$ “zero or one”, and ℓ^* “any number” of outgoing edges.

In principle. $\text{in}(d, \text{in}_d : V \rightarrow 2^{\Pi_{\mathcal{L}}})$: The incoming edges of a node on dimension d must satisfy in label and number the stipulation of the corresponding in_d feature. Symmetrical to the out principle, the in principle constrains the incoming edges of a node.

Order principle. $\text{order}(d, \prec_d, \text{on}_d : V \rightarrow 2^{\mathcal{F}_{\mathcal{N}}})$ For the order principle, we refer the reader to (Duchier, 2001). Suffice to say that it applies on dimension d with the total order $\prec_d$. Lexical feature on_d states the set of possible node labels.

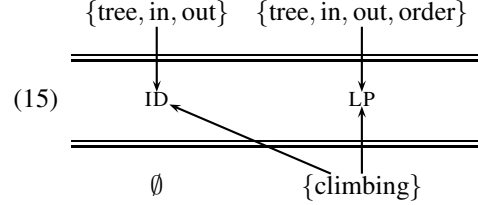
Linking principle. $\text{linking}(d_1, d_2, \text{link}_{d_1, d_2} : V \rightarrow \mathcal{L}_1 \rightarrow 2^{\mathcal{L}_2})$: The dependents of a node on dimension d_1 must have as their incoming edge label on dimension d_2 one of those stipulated by the link_{d_1, d_2} feature. $\mathcal{L}_1$ is the set of edge labels of dimension d_1 , and $\mathcal{L}_2$ of dimension d_2 . Essentially, the principle synchronizes two dimensions in a lexical fashion, by the following constraint, where $\ell_1 \in \mathcal{L}_1$ and $\ell_2 \in \mathcal{L}_2$:

$$(14) \quad v - \ell_1 \rightarrow_{d_1} v' \Rightarrow v'' - \ell_2 \rightarrow_{d_2} v' \wedge \ell_2 \in \text{link}_{d_1, d_2}(v)(\ell_1)$$

That is, an edge from v to v' on dimension d_1 is licensed only if, on dimension d_2 , there is a node v'' and an edge from v'' to v' labeled by one of $\text{link}_{d_1, d_2}(v)(\ell_1)$.

2.4 Obtaining ID/LP TDG

In the following, we instantiate TDG to get the ID/LP TDG grammar formalism:



On the ID dimension, ID/LP TDG makes use of the principles tree, in and out. On the LP dimension, it reuses the three, and adds the order principle. ID/LP TDG uses only one inter-dimensional principle, viz. the climbing principle on the LP dimension.

The instantiation of TDG in (15) gives rise to the following lexical entry signature:

$$(16) \quad \left[\begin{array}{l} \text{ID} : \left[\begin{array}{l} \text{in} : 2^{\Pi_{\mathcal{R}}} \\ \text{out} : 2^{\Pi_{\mathcal{R}}} \end{array} \right] \\ \text{LP} : \left[\begin{array}{l} \text{in} : 2^{\Pi_{\mathcal{F}_E}} \\ \text{on} : 2^{\mathcal{F}_N} \\ \text{out} : 2^{\Pi_{\mathcal{F}_E}} \end{array} \right] \end{array} \right]$$

where $\mathcal{R}$ is the set of edge labels on the ID dimension, $\mathcal{F}_E$ the set of edge labels, and $\mathcal{F}_N$ the set of node labels on the LP dimension.

2.5 ID/LP TDG example grammar

We present a simple example grammar for the ID/LP TDG grammar formalism. First, we define the edge labels on the ID and the LP dimension ($\mathcal{R}$ and $\mathcal{F}_E$), and the node labels on the LP dimension ($\mathcal{F}_N$):

$$(17) \quad \begin{aligned} \mathcal{R} &= \{\text{subj}, \text{obj}, \text{vinf}\} \\ \mathcal{F}_E &= \{\text{mf}, \text{vcf}\} \\ \mathcal{F}_N &= \{\text{n}, \text{v}\} \end{aligned}$$

For the order principle, we define the total order on $\mathcal{F}_E \cup \mathcal{F}_N$ as already posited in (4).

The rest of the grammar is made up of the lexical entries:

$$(18) \quad \begin{array}{l} \text{jeder Mann} \mapsto \\ \left[\begin{array}{l} \text{ID} : \left[\begin{array}{l} \text{in} : \{\text{subj?}\} \\ \text{out} : \emptyset \end{array} \right] \\ \text{LP} : \left[\begin{array}{l} \text{in} : \{\text{mf?}\} \\ \text{on} : \{\text{n}\} \\ \text{out} : \emptyset \end{array} \right] \end{array} \right] \end{array}$$

$$(19) \quad \begin{array}{c} \text{eine Frau} \mapsto \\ \left[\begin{array}{c} \text{ID} : \left[\begin{array}{c} \text{in} : \{\text{obj}^?\} \\ \text{out} : \emptyset \end{array} \right] \\ \text{LP} : \left[\begin{array}{c} \text{in} : \{\text{mf}^?\} \\ \text{on} : \{\text{n}\} \\ \text{out} : \emptyset \end{array} \right] \end{array} \right] \end{array}$$

$$(20) \quad \begin{array}{c} \text{zu lieben} \mapsto \\ \left[\begin{array}{c} \text{ID} : \left[\begin{array}{c} \text{in} : \{\text{vinf}^?\} \\ \text{out} : \{\text{obj}\} \end{array} \right] \\ \text{LP} : \left[\begin{array}{c} \text{in} : \{\text{vcf}^?\} \\ \text{on} : \{\text{v}\} \\ \text{out} : \emptyset \end{array} \right] \end{array} \right] \end{array}$$

$$(21) \quad \begin{array}{c} \text{scheint} \mapsto \\ \left[\begin{array}{c} \text{ID} : \left[\begin{array}{c} \text{in} : \emptyset \\ \text{out} : \{\text{subj}\} \end{array} \right] \\ \text{LP} : \left[\begin{array}{c} \text{in} : \emptyset \\ \text{on} : \{\text{v}\} \\ \text{out} : \{\text{mf}^*, \text{vcf}^?\} \end{array} \right] \end{array} \right] \end{array}$$

The lexical entries for the NPs *jeder Mann* and *eine Frau* are almost identical: on the ID dimension, their incoming edge must be labeled subj (*jeder Mann*) or obj (*eine Frau*), i.e. *jeder Mann* must be a subject, and *eine Frau* an object. On the LP dimension, their only possible incoming edge label is mf, which captures the generalization that nouns can only land in the topological field called Mittelfeld in German. Their node label must be n.

The lexical entry for the infinitive *zu lieben* states that its incoming edge must be vinf on the ID dimension and vcf on the LP dimension. The only possible node label is v. Moreover, *zu lieben* must have precisely one dependent labeled obj on the ID dimension. On the LP dimension, it must not have any outgoing edge.

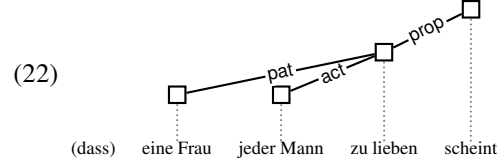
The lexical entry for *scheint* states the following. On the ID dimension, *scheint* must not have any incoming edges, and it requires precisely one outgoing edge labeled subj. On the LP dimension, *scheint* also must not have any incoming edge, and has node label v. In addition, the lexical entry for *scheint* licenses an arbitrary number of outgoing edges labeled mf, and zero or one outgoing edges labeled vcf.

3 ID/LP/TH TDG

In this section, we instantiate TDG to get the ID/LP/TH TDG grammar formalism, extending ID/LP TDG of the previous section by a dimension called TH dimension (for *thematic*).

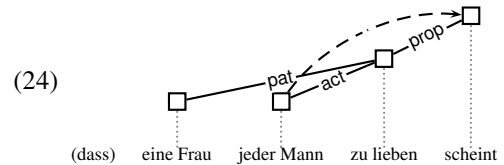
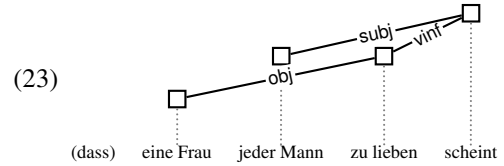
3.1 ID/LP/TH dags

An analysis on the TH dimension is called TH dag: a directed acyclic graph with edges labeled by semantic roles. Here is a TH dag for sentence (1):



Here, *zu lieben* is a prop-dependent (for *proposition*) of *scheint*. *jeder Mann* is the actor or deep subject (act), and *eine Frau* is the patient or deep object (pat) of *zu lieben*. These labels (except prop) are taken from the set of dependency relations on the tectogrammatical layer of Functional Generative Description (FGD) (Sgall et al., 1986).

TH dags represent the *semantic* argument structure of a sentence, contrary to ID trees, which represent the *syntactic* argument structure. The two do not always match. We contrast below the ID tree (23) with the TH dag analysis (24) of our example sentence:



In the ID tree, *jeder Mann* is the subject of *scheint*, and *eine Frau* is the object of *zu lieben*. In the TH dag, *eine Frau* also is a dependent of *zu lieben* (its patient). However, *jeder Mann* is not the dependent of *scheint* but of *zu lieben* (its actor). Again, as in ID/LP TDG, a node has climbed up: *jeder Mann* is the daughter of *zu lieben* on the TH dimension, and of *scheint* on the ID dimension, as indicated by the dashed arrow in (24). In standard linguistics terminology, this phenomenon is called *raising*.

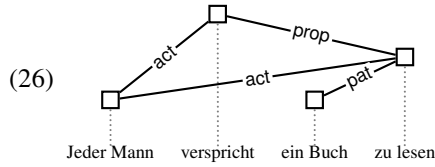
Here, the climbing direction is from the TH to the ID dimension. Before, in ID/LP TDG, we used climbing in the direction from the ID to the LP dimension. ID-LP climbing enabled us to model word

order freedom, whereas the TH-ID climbing allows us to model control and raising phenomena.

The attentive reader might pose the question why an analysis on the TH dimension is a directed acyclic graph, and not a tree. The answer is that for phenomena such as control, a node in the TH dag can have more than one incoming edge. As an example, consider the following sentence, including the control verb *verspricht*:

- (25) *Jeder Mann verspricht ein Buch zu lesen.*
 Every man+NOM tries a book+ACC
 to read.
 “Every man tries to read a book.”

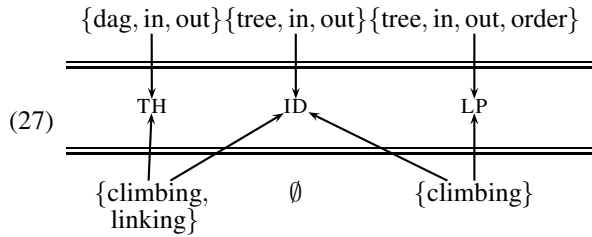
Below, we display the corresponding TH dag:



Here, *jeder Mann* has two incoming edges: it is not only the actor of *zu lieben*, but also of *verspricht*.

3.2 Obtaining ID/LP/TH TDG

ID/LP/TH TDG is an instantiation of TDG along the following lines:



That is, ID/LP/TH TDG recycles all principles already used in ID/LP TDG. Additionally, ID/LP/TH TDG utilizes the *dag principle* and the *linking principle*. We use the linking principle to specify by which grammatical function a semantic role can be realized. E.g. subjects typically realize actors. TDG’s linking principle has been elaborated in (Korthals and Debusmann, 2002), an approach that even surpasses the linking theory of (Davis, 1998) in some respects.

The instantiation of TDG in (27) gives rise to the following lexical entry signature:

$$(28) \left[\begin{array}{l} \text{ID} : \left[\begin{array}{l} \text{in} : 2^{\Pi_{\mathcal{R}}} \\ \text{out} : 2^{\Pi_{\mathcal{R}}} \end{array} \right] \\ \text{LP} : \left[\begin{array}{l} \text{in} : 2^{\Pi_{\mathcal{F}_E}} \\ \text{on} : 2^{\mathcal{F}_N} \\ \text{out} : 2^{\Pi_{\mathcal{F}_E}} \end{array} \right] \\ \text{TH} : \left[\begin{array}{l} \text{in} : 2^{\Pi_{\mathcal{S}}} \\ \text{out} : 2^{\Pi_{\mathcal{S}}} \\ \text{link} : \mathcal{S} \rightarrow 2^{\mathcal{R}} \end{array} \right] \end{array} \right]$$

where the ID and LP subsignatures are the same as in (16). In the TH subsignature, $\mathcal{S}$ is the set of edge labels on the TH dimension.

3.3 ID/LP/TH TDG example grammar

We present a simple example grammar for the ID/LP/TH TDG grammar formalism. We use the same sets of labels for the ID and LP dimensions as before in (17). The total order on the set $\mathcal{F}_E \cup \mathcal{F}_N$ of edge and node labels also stays the same. We define the new set $\mathcal{S}$ of edge labels on the TH dimension as follows:

$$(29) \mathcal{S} = \{\text{act}, \text{pat}, \text{prop}\}$$

The rest of the grammar again consists of the lexical entries:³

$$(30) \begin{array}{l} \text{jeder Mann} \mapsto \\ \left[\begin{array}{l} \text{TH} : \left[\begin{array}{l} \text{in} : \{\text{act}^?\} \\ \text{out} : \emptyset \\ \text{link} : \emptyset \end{array} \right] \end{array} \right] \end{array}$$

$$(31) \begin{array}{l} \text{eine Frau} \mapsto \\ \left[\begin{array}{l} \text{TH} : \left[\begin{array}{l} \text{in} : \{\text{pat}^?\} \\ \text{out} : \emptyset \\ \text{link} : \emptyset \end{array} \right] \end{array} \right] \end{array}$$

$$(32) \begin{array}{l} \text{zu lieben} \mapsto \\ \left[\begin{array}{l} \text{TH} : \left[\begin{array}{l} \text{in} : \{\text{prop}^?\} \\ \text{out} : \{\text{act}, \text{pat}\} \\ \text{link} : \{\text{act} \mapsto \{\text{subj}\}, \\ \text{pat} \mapsto \{\text{obj}\}\} \end{array} \right] \end{array} \right] \end{array}$$

$$(33) \begin{array}{l} \text{scheint} \mapsto \\ \left[\begin{array}{l} \text{TH} : \left[\begin{array}{l} \text{in} : \emptyset \\ \text{out} : \{\text{prop}\} \\ \text{link} : \{\text{prop} \mapsto \{\text{vinf}\}\} \end{array} \right] \end{array} \right] \end{array}$$

In this lexicon, *jeder Mann* can only be an actor, and *eine Frau* a patient by their in features, and *zu lieben* can only be a proposition (prop). In addition, *zu lieben* requires an actor and a patient by its out feature. The link feature states that the actor must

³Note that we do not repeat the ID and LP parts of the lexical entries already shown in (18–21).

⁴For want of space, we omit e.g. the q position for quantifiers, and the r position for restrictions that are both present in the full account.

5.2 Lambda bindings

To recover the lambda bindings for CLLS, we again identify distinguished positions in the CLLS fragments: we identify the n lambda binders in a fragment with features $lam_1, \dots, lam_n$, and the m variable occurrences by features $var_1, \dots, var_m$. N is the union of all these features.

We make use of the information contained in the TH dag to recover the lambda bindings. We introduce two new lexical features lam and var of type $\mathcal{S} \rightarrow 2^N$, mapping lam semantic roles to sets of lambda binder positions and to sets of variable positions respectively. lam and var map a semantic role θ either to the singleton set of the position corresponding to θ , or otherwise to the empty set. For an edge $v - \theta \rightarrow_{TH} v'$ in the TH dag, we obtain the corresponding lambda binding (dashed arrow) as follows:

$$(40) \quad v - \theta \rightarrow_{TH} v' \Rightarrow v.var.\theta \quad v'.lam.\theta$$

However note that we do not obtain lambda bindings for all edges: we obtain a lambda binding for edge $v - \theta \rightarrow_{TH} v'$ only if both $v.var.\theta$ and $v'.lam.\theta$ denote singletons. Thus we can exclude lambda bindings to positions in fragments which do not correspond to individuals (e.g. *scheint*).

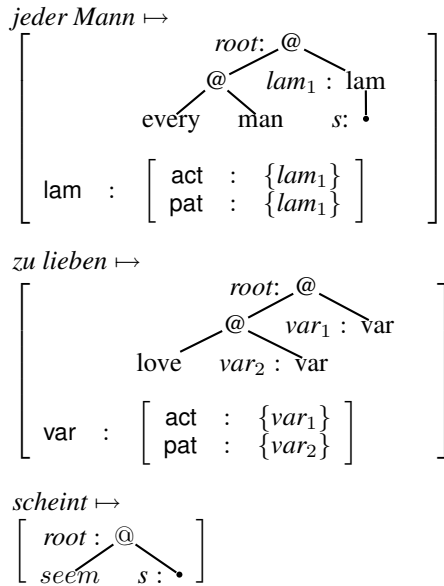


Figure 2: CLLS parts of the lexical entries

6 Conclusion

We presented a meta-grammatical framework, articulated around the notion of dependencies and lex-

icalized constraints, that generalizes TDG: instead of being limited to two interacting dimensions, one dedicated to syntax and the other to topology, this new framework can accommodate any number of dimensions. Furthermore, parametric principles are used to stipulate the models licensed for each dimension, as well the restrictions arising from inter-dimensional interactions. We have shown how it can be instantiated to obtain the earlier ID/LP TDG. We then described the ID/LP/TH extension which provides support for semantic dependencies and accounts for control and raising constructions. Finally the DE extension was presented to support meaning assembly in CLLS.

References

- Tony Davis. 1998. *Linking as constraints in the hierarchical lexicon*.
- Denys Duchier and Ralph Debusmann. 2001. Topological dependency trees: A constraint-based account of linear precedence. In *ACL 2001 Proceedings*.
- Denys Duchier. 2001. Lexicalized syntax and topology for non-projective dependency grammar. In *MOL 8 Proceedings*.
- Markus Egg, Alexander Koller, and Joachim Niehren. 2001. The constraint language for lambda structures. *Journal of Logic, Language, and Information*.
- Kim Gerdes and Sylvain Kahane. 2001. Word order in german: A formal dependency grammar using a topological hierarchy. In *ACL 2001 Proceedings*.
- Tilman Höhle. 1986. Der Begriff “Mittelfeld”, Anmerkungen über die Theorie der topologischen Felder.
- Aravind K. Joshi. 1987. An introduction to tree-adjoining grammars. In *Mathematics of Language*.
- Christian Korthals and Ralph Debusmann. 2002. Linking syntactic and semantic arguments in a dependency-based formalism. In *COLING 2002 Proceedings*.
- Mike Reape. 1994. Domain union and word order variation in german. In *German in Head-Driven Phrase Structure Grammar*.
- Petr Sgall, Eva Hajicova, and Jarmila Panevova. 1986. *The Meaning of the Sentence in its Semantic and Pragmatic Aspects*.
- Lucien Tesnière. 1959. *Éléments de Syntaxe Structurale*.

A Relational Syntax-Semantics Interface Based on Dependency Grammar

Ralph Debusmann Denys Duchier* Alexander Koller Marco Kuhlmann Gert Smolka Stefan Thater

Saarland University, Saarbrücken, Germany

*LORIA, Nancy, France

{rade|kuhlmann|smolka}@ps.uni-sb.de, duchier@loria.fr, {koller|stth}@coli.uni-sb.de

Abstract

We propose a syntax-semantics interface that realises the mapping between syntax and semantics as a relation and does not make functionality assumptions in either direction. This interface is stated in terms of Extensible Dependency Grammar (XDG), a grammar formalism we newly specify. XDG's constraint-based parser supports the concurrent flow of information between any two levels of linguistic representation, even when only partial analyses are available. This generalises the concept of underspecification.

1 Introduction

A key assumption of traditional syntax-semantics interfaces, starting with (Montague, 1974), is that the mapping from syntax to semantics is *functional*, i. e. that once we know the syntactic structure of a sentence, we can deterministically compute its semantics.

Unfortunately, this assumption is typically not justified. Ambiguities such as of quantifier scope or pronominal reference are genuine *semantic* ambiguities; that is, even a syntactically unambiguous sentence can have multiple semantic readings. Conversely, a common situation in natural language generation is that one semantic representation can be verbalised in multiple ways. This means that the relation between syntax and semantics is not functional at all, but rather a true *m-to-n* relation.

There is a variety of approaches in the literature on syntax-semantics interfaces for coping with this situation, but none of them is completely satisfactory. One way is to recast semantic ambiguity as syntactic ambiguity by compiling semantic distinctions into the syntax (Montague, 1974; Steedman, 1999; Moortgat, 2002). This restores functionality, but comes at the price of an artificial blow-up of syntactic ambiguity. A second approach is to assume a non-deterministic mapping from syntax to semantics as in generative grammar (Chomsky, 1965), but it is not always obvious how to reverse the relation, e. g. for generation. For LFG, the operation of functional uncertainty allows for a restricted form of relationality (Kaplan and Maxwell III, 1988). Finally, underspecification (Egg et al.,

2001; Gupta and Lamping, 1998; Copestake et al., 2004) introduces a new level of representation, which can be computed functionally from a syntactic analysis and encapsulates semantic ambiguity in a way that supports the enumeration of all semantic readings by need.

In this paper, we introduce a completely relational syntax-semantics interface, building upon the underspecification approach. We assume a set of linguistic *dimensions*, such as (syntactic) immediate dominance and predicate-argument structure; a grammatical analysis is a tuple with one component for each dimension, and a grammar describes a set of such tuples. While we make no *a priori* functionality assumptions about the relation of the linguistic dimensions, functional mappings can be obtained as a special case. We formalise our syntax-semantics interface using *Extensible Dependency Grammar* (XDG), a new grammar formalism which generalises earlier work on Topological Dependency Grammar (Duchier and Debusmann, 2001).

The relational syntax-semantics interface is supported by a parser for XDG based on constraint programming. The crucial feature of this parser is that it supports the concurrent flow of possibly partial information between any two dimensions: once additional information becomes available on one dimension, it can be propagated to any other dimension. Grammaticality conditions and preferences (e. g. selectional restrictions) can be specified on their natural level of representation, and inferences on each dimension can help reduce ambiguity on the others. This generalises the idea of underspecification, which aims to represent and reduce ambiguity through inferences on a *single* dimension only.

The structure of this paper is as follows: in Section 2, we give the general ideas behind XDG, its formal definition, and an overview of the constraint-based parser. In Section 3, we present the relational syntax-semantics interface, and go through examples that illustrate its operation. Section 4 shows how the semantics side of our syntax-semantics interface can be precisely related to mainstream semantics research. We summarise our results and point to further work in Section 5.

2 Extensible Dependency Grammar

This section presents Extensible Dependency Grammar (XDG), a description-based formalism for dependency grammar. XDG generalizes previous work on Topological Dependency Grammar (Duchier and Debusmann, 2001), which focussed on word order phenomena in German.

2.1 XDG in a Nutshell

XDG is a description language over finite labelled graphs. It is able to talk about two kinds of constraints on these structures: The *lexicon* of an XDG grammar describes properties local to individual nodes, such as valency. The grammar’s *principles* express constraints global to the graph as a whole, such as treeness. Well-formed analyses are graphs that satisfy all constraints.

An XDG grammar allows the characterisation of linguistic structure along several *dimensions* of description. Each dimension contains a separate graph, but all these graphs share the same set of nodes. Lexicon entries synchronise dimensions by specifying the properties of a node on all dimensions at once. Principles can either apply to a single dimension (*one-dimensional*), or constrain the relation of several dimensions (*multi-dimensional*).

Consider the example in Fig. 1, which shows an analysis for a sentence of English along two dimensions of description, immediate dominance (ID) and linear precedence (LP). The principles of the underlying grammar require both dimensions to be trees, and the LP tree to be a “flattened” version of the ID tree, in the sense that whenever a node v is a transitive successor of a node u in the LP tree, it must also be a transitive successor of u in the ID tree. The given lexicon specifies the potential incoming and required outgoing edges for each word on both dimensions. The word *does*, for example, accepts no incoming edges on either dimension and must therefore be at the root of both the ID and the LP tree. It is required to have outgoing edges to a subject (subj) and a verb base form (vbse) in the ID tree, needs fillers for a subject (sf) and a verb complement field (vcf) in the LP tree, and offers an optional field for topicalised material (tf). All these constraints are satisfied by the analysis, which is thus well-formed.

2.2 Formalisation

Formally, an XDG grammar is built up of dimensions, principles, and a lexicon, and characterises a set of well-formed analyses.

A *dimension* is a tuple $D = (Lab, Fea, Val, Pri)$ of a set Lab of edge labels, a set Fea of features, a set Val of feature values, and a set of one-dimensional

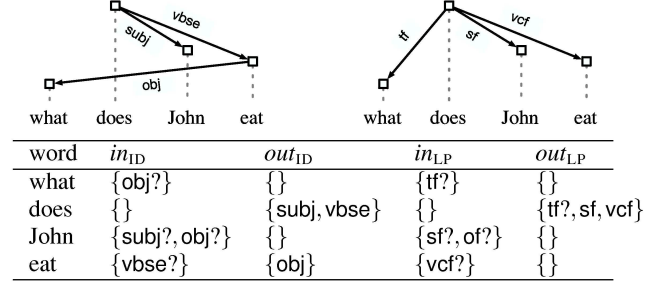


Figure 1: XDG analysis of “what does John eat”

principles Pri . A *lexicon* for the dimension D is a set $Lex \subseteq Fea \rightarrow Val$ of total feature assignments (or lexical entries). A D -*structure*, representing an analysis on dimension D , is a triple (V, E, F) of a set V of nodes, a set $E \subseteq V \times V \times Lab$ of directed labelled edges, and an assignment $F : V \rightarrow (Fea \rightarrow Val)$ of lexical entries to nodes. V and E form a graph. We write Str_D for the set of all possible D -structures. The principles characterise subsets of Str_D that have further dimension-specific properties, such as being a tree, satisfying assigned valencies, etc. We assume that the elements of Pri are finite representations of such subsets, but do not go into details here; some examples are shown in Section 3.2.

An XDG grammar $((Lab_i, Fea_i, Val_i, Pri_i)_{i=1}^n, Pri, Lex)$ consists of n dimensions, multi-dimensional principles Pri , and a lexicon Lex . An XDG *analysis* $(V, E_i, F_i)_{i=1}^n$ is an element of $Ana = Str_1 \times \dots \times Str_n$ where all dimensions share the same set of nodes V .

Multi-dimensional principles work just like one-dimensional principles, except that they specify subsets of Ana , i. e. couplings between dimensions (e. g. the flattening principle between ID and LP in Section 2.1). The lexicon $Lex \subseteq Lex_1 \times \dots \times Lex_n$ constrains all dimensions at once. An XDG analysis is licenced by Lex iff $(F_1(w), \dots, F_n(w)) \in Lex$ for every node $w \in V$.

In order to compute analyses for a given input, we model it as a set of *input constraints* (Inp), which again specify a subset of Ana . The parsing problem for XDG is then to find elements of Ana that are licenced by Lex and consistent with Inp and Pri . Note that the term “parsing problem” is traditionally used only for inputs that are sequences of words, but we can easily represent surface realisation as a “parsing” problem in which Inp specifies a semantic dimension; in this case, a “parser” would compute analyses that contain syntactic dimensions from which we can read off a surface sentence.

2.3 Constraint Solver

The parsing problem of XDG has a natural reading as a constraint satisfaction problem (CSP) (Apt,

2003) on finite sets of integers; well-formed analyses correspond to the solutions of this problem. The transformation, whose details we omit due to lack of space, closely follows previous work on axiomatising dependency parsing (Duchier, 2003) and includes the use of the *selection constraint* to efficiently handle lexical ambiguity.

We have implemented a constraint solver for this CSP using the Mozart/Oz programming system (Smolka, 1995; Mozart Consortium, 2004). This solver does a search for a satisfying variable assignment. After each case distinction (*distribution*), it performs simple inferences that restrict the ranges of the finite set variables and thus reduce the size of the search tree (*propagation*). The successful leaves of the search tree correspond to XDG analyses, whereas the inner nodes correspond to *partial analyses*. In these cases, the current constraints are too weak to specify a complete analysis, but they already express that some edges or feature values must be present, and that others are excluded. Partial analyses will play an important role in Section 3.3.

Because propagation operates on all dimensions concurrently, the constraint solver can frequently infer information about one dimension from information on another, if there is a multi-dimensional principle linking the two dimensions. These inferences take place while the constraint problem is being solved, and they can often be drawn before the solver commits to any single solution.

Because XDG allows us to write grammars with completely free word order, XDG solving is an NP-complete problem (Koller and Striegnitz, 2002). This means that the worst-case complexity of the solver is exponential, but the average-case complexity for the hand-crafted grammars we experimented with is often better than this result suggests. We hope there are useful fragments of XDG that would guarantee polynomial worst-case complexity.

3 A Relational Syntax-Semantics Interface

Now that we have the formal and processing frameworks in place, we can define a relational syntax-semantics interface for XDG. We will first show how we encode semantics within the XDG framework. Then we will present an example grammar (including some principle definitions), and finally go through an example that shows how the relationality of the interface, combined with the concurrency of the constraint solver, supports the flow of information between different dimensions.

3.1 Representing Meaning

We represent meaning within XDG on two dimensions: one for *predicate-argument structure* (PA),

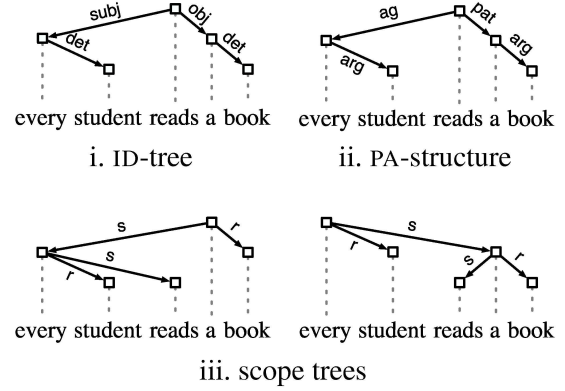


Figure 2: Two analyses for the sentence “every student reads a book.”

and one for *scope* (SC). The function of the PA dimension is to abstract over syntactic idiosyncrasies such as active-passive alternations or dative shifts, and to make certain semantic dependencies e. g. in control constructions explicit; it deals with concepts such as agent and patient, rather than subject and object. The purpose of the SC dimension is to reflect the structure of a logical formula that would represent the semantics, in terms of scope and restriction. We will make this connection explicit in Section 4. In addition, we assume an ID dimension as above. We do not include an LP dimension only for ease of presentation; it could be added completely orthogonally to the three dimensions we consider here.

While one ID structure will typically correspond to one PA structure, each PA structure will typically be consistent with multiple SC structures, because of scope ambiguities. For instance, Fig. 2 shows the unique ID and PA structures for the sentence “Every student reads a book.” These structures (and the input sentence) are consistent with the two possible SC-structures shown in (iii). Assuming a Davidsonian event semantics, the two SC trees (together with the PA-structure) represent the two readings of the sentence:

- $\lambda e. \forall x. \text{student}(x) \rightarrow \exists y. \text{book}(y) \wedge \text{read}(e, x, y)$
- $\lambda e. \exists y. \text{book}(y) \wedge \forall x. \text{student}(x) \rightarrow \text{read}(e, x, y)$

3.2 A Grammar for a Fragment of English

The lexicon for an XDG grammar for a small fragment of English using the ID, PA, and SC dimensions is shown in Fig. 3. Each row in the table specifies a (unique) lexical entry for each part of speech (determiner, common noun, proper noun, transitive verb and preposition); there is no lexical ambiguity in this grammar. Each column specifies a feature. The meaning of the features will be explained together

	in_{ID}	out_{ID}	in_{PA}	out_{PA}	in_{SC}	out_{SC}
DET	{subj?, obj?, pcomp?}	{det!}	{ag?, pat?, arg?}	{quant!}	{r?, s?, a?}	{r!, s!}
CN	{det?}	{prep*}	{quant?}	{mod?}	{r?, s?, a?}	{}
PN	{subj?, obj?, pcomp?}	{prep*}	{ag?, pat?, arg?}	{mod?}	{r?, s?, a?}	{r?, s!}
TV	{}	{subj!, obj!, prep*}	{}	{ag!, pat!, instr?}	{r?, s?, a?}	{}
PREP	{prep?}	{pcomp!}	{mod?, instr?}	{arg!}	{r?, s?, a?}	{a!}

	$link$	$codom$	$contradom$
DET	{quant $\mapsto$ {det}}	{quant $\mapsto$ {r}}	{}
CN,PN	{mod $\mapsto$ {prep}}	{}	{mod $\mapsto$ {a}}
TV	{ag $\mapsto$ {subj}, pat $\mapsto$ {obj}, instr $\mapsto$ {prep}}	{}	{ag $\mapsto$ {s}, pat $\mapsto$ {s}, instr $\mapsto$ {a}}
PREP	{arg $\mapsto$ {pcomp}}	{}	{arg $\mapsto$ {s}}

Figure 3: The example grammar fragment

with the principles that use them.

The ID dimension uses the edge labels $Lab_{ID} = \{\text{det}, \text{subj}, \text{obj}, \text{prep}, \text{pcomp}\}$ resp. for determined common noun,¹ subject, object, preposition, and complement of a preposition. The PA dimension uses $Lab_{PA} = \{\text{ag}, \text{pat}, \text{arg}, \text{quant}, \text{mod}, \text{instr}\}$, resp. for agent, patient, argument of a modifier, common noun pertaining to a quantifier, modifier, and instrument; and SC uses $Lab_{SC} = \{r, s, a\}$ resp. for restriction and scope of a quantifier, and for an argument.

The grammar also contains three one-dimensional principles (tree, dag, and valency), and three multi-dimensional principles (linking, co-dominance, and contra-dominance).

Tree and dag principles. The tree principle restricts ID and SC structures to be trees, and the dag principle restricts PA structures to be directed acyclic graphs.

Valency principle. The valency principle, which we use on all dimensions, states that the incoming and outgoing edges of each node must obey the specifications of the in and out features. The possible values for each feature in_d and out_d are subsets of $Lab_d \times \{!, ?, *\}$. $\ell!$ specifies a mandatory edge with label ℓ , $\ell?$ an optional one, and $\ell*$ zero or more.

Linking principle. The linking principle for dimensions d_1, d_2 constrains how dependents on d_1 may be realised on d_2 . It assumes a feature $link_{d_1, d_2}$ whose values are functions that map labels from Lab_{d_1} to sets of labels from Lab_{d_2} , and is specified by the following implication:

$$v \xrightarrow{!}_{d_1} v' \Rightarrow \exists \ell' \in link_{d_1, d_2}(v)(\ell) : v \xrightarrow{\ell'}_{d_2} v'$$

Our grammar uses this principle with the *link* feature to constrain the realisations of PA-dependents in the ID dimension. In Fig. 2, the agent (ag) of *reads* must be realised as the subject (subj), i. e.

$$reads \xrightarrow{ag}_{PA} every \Rightarrow reads \xrightarrow{subj}_{ID} every$$

Similarly for the patient and the object. There is no instrument dependent in the example, so this part of the *link* feature is not used. An ergative verb would use a *link* feature where the subject realises the patient; Control and raising phenomena can also be modelled, but we cannot present this here.

Co-dominance principle. The co-dominance principle for d_1, d_2 relates edges in d_1 to dominance relations in the same direction in d_2 . It assumes a feature $codom_{d_1, d_2}$ mapping labels in Lab_{d_1} to sets of labels in Lab_{d_2} and is specified as

$$v \xrightarrow{l}_{d_1} v' \Rightarrow \exists l' \in codom_{d_1, d_2}(v)(l) : v \xrightarrow{l'}_{d_2} v'$$

Our grammar uses the co-dominance principle on dimension PA and SC to express, e. g., that the propositional contribution of a noun must end up in the restriction of its determiner. For example, for the determiner *every* of Fig. 2 we have:

$$every \xrightarrow{quant}_{PA} student \Rightarrow every \xrightarrow{r}_{SC} \rightarrow_{SC}^* student$$

Contra-dominance principle. The contra-dominance principle is symmetric to the co-dominance principle, and relates edges in d_1 to dominance edges into the opposite direction in d_2 . It assumes a feature $contradom_{d_1, d_2}$ mapping labels of Lab_{d_1} to sets of labels from Lab_{d_2} and is specified as

$$v \xrightarrow{l}_{d_1} v' \Rightarrow \exists l' \in contradom_{d_1, d_2}(v)(l) : v' \xrightarrow{l'}_{d_2} v$$

Our grammar uses the contra-dominance principle on dimensions PA and SC to express, e. g., that predicates must end up in the scope of the quantifiers whose variables they refer to. Thus, for the transitive verb *reads* of Fig. 2, we have:

$$reads \xrightarrow{ag}_{PA} every \Rightarrow every \xrightarrow{s}_{SC} \rightarrow_{SC}^* reads$$

$$reads \xrightarrow{pat}_{PA} a \Rightarrow a \xrightarrow{s}_{SC} \rightarrow_{SC}^* reads$$

¹We assume on all dimensions that determiners are the heads of common nouns. This makes for a simpler relationship between the syntactic and semantic dimensions.

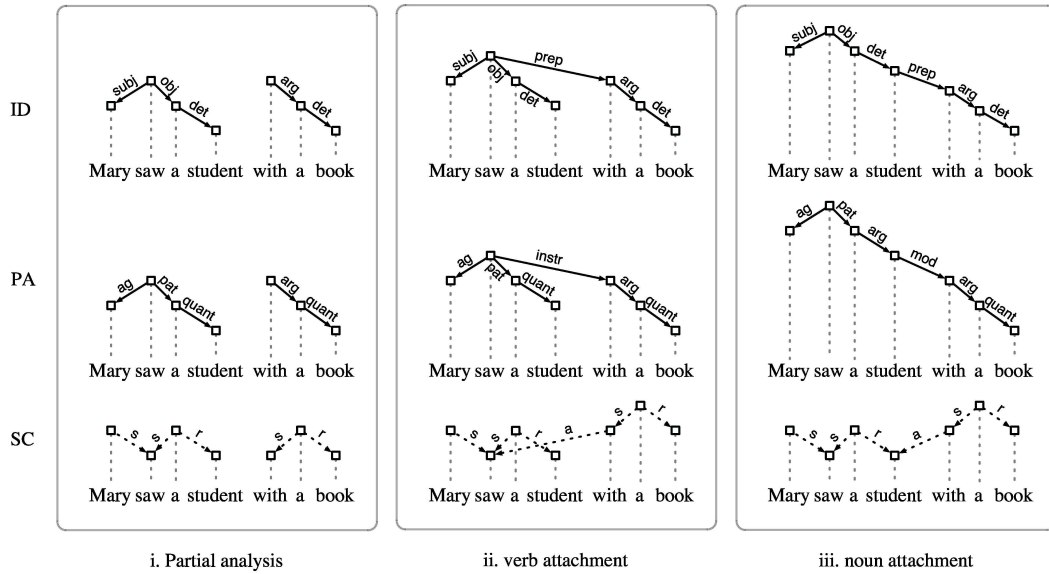


Figure 4: Partial description (left) and two solutions (right) for “Mary saw a student with a book.”

3.3 Syntax-Semantics Interaction

It is important to note at this point that the syntax-semantics interface we have defined is indeed relational. Each principle declaratively specifies a set of admissible analyses, i. e. a relation between the structures for the different dimensions, and the analyses that the complete grammar judges grammatical are simply those that satisfy all principles. The role of the lexicon is to provide the feature values which parameterise the principles defined above.

The constraint solver complements this relationality by supporting the use of the principles to move information between any two dimensions. If, say, the left-hand side of the linking principle is found to be satisfied for dimension d_1 , a propagator will infer the right-hand side and add it to dimension d_2 . Conversely, if the solver finds that the right-hand side must be false for d_2 , the negation of the left-hand side is inferred for d_1 . By letting principles interact concurrently, we can make some very powerful inferences, as we will demonstrate with the example sentence “Mary saw a student with a book,” some partial analyses for which are shown in Fig. 4.

Column (i) in the figure shows the state after the constraint solver finishes its initial propagation, at the root of the search tree. Even at this point, the valency and treeness principles have conspired to establish an almost complete ID-structure. By the linking principle, the PA-structure has been determined similarly closely. The SC-structure is still mostly undetermined, but by the co- and contra-dominance principles, the solver has already established that some nodes must *dominate* others: A dotted edge with label *s* in the picture means that the solver

knows there must be a path between these two nodes which starts with an *s*-edge. In other words, the solver has computed a large amount of semantic information from an incomplete syntactic analysis.

Now imagine some external source tells us that *with* is a *mod*-child of *student* on PA, i. e. the analysis in (iii). This information could come e. g. from a statistical model of selectional preferences, which will judge this edge much more probable than an *instr*-edge from the verb to the preposition (ii). Adding this edge will trigger additional inferences through the linking principle, which can now infer that *with* is a *prep*-child of *student* on ID. In the other direction, the solver will infer more dominances on SC. This means that semantic information can be used to disambiguate syntactic ambiguities, and semantic information such as selectional preferences can be stated on their natural level of representation, rather than be forced into the ID dimension directly.

Similarly, the introduction of new edges on SC could trigger a similar reasoning process which would infer new PA-edges, and thus indirectly also new ID-edges. Such new edges on SC could come from inferences with world or discourse knowledge (Koller and Niehren, 2000), scope preferences, or interactions with information structure (Duchier and Kruijff, 2003).

4 Traditional Semantics

Our syntax-semantics interface represents semantic information as graphs on the PA and SC dimensions. While this looks like a radical departure from traditional semantic formalisms, we consider these graphs simply an alternative way of presenting more

traditional representations. We devote the rest of the paper to demonstrating that a pair of a PA and a SC structure can be interpreted as a Montague-style formula, and that a partial analysis on these two dimensions can be seen as an underspecified semantic description.

4.1 Montague-style Interpretation

In order to extract a standard type-theoretic expression from an XDG analysis, we assign each node v two semantic values: a *lexical* value $\mathcal{L}(v)$ representing the semantics of v itself, and a *phrasal* value $\mathcal{P}(v)$ representing the semantics of the entire SC-subtree rooted at v . We use the SC-structure to determine functor-argument relationships, and the PA-structure to establish variable binding.

We assume that nodes for determiners and proper names introduce unique individual variables (“indices”). Below we will write $\langle\langle v \rangle\rangle$ to refer to the index of the node v , and we write $\downarrow \ell$ to refer to the node which is the ℓ -child of the current node in the appropriate dimension (PA or SC). The semantic lexicon is defined as follows; “ $\mathcal{L}(w)$ ” should be read as “ $\mathcal{L}(v)$, where v is a node for the word w ”.

$$\begin{aligned}\mathcal{L}(a) &= \lambda P \lambda Q \lambda e. \exists x (P(x) \wedge Q(x)(e)) \\ \mathcal{L}(\text{book}) &= \text{book}' \\ \mathcal{L}(\text{with}) &= \lambda P \lambda x. (\text{with}'(\langle\langle \downarrow \text{arg} \rangle\rangle)(x) \wedge P(x)) \\ \mathcal{L}(\text{reads}) &= \text{read}'(\langle\langle \downarrow \text{pat} \rangle\rangle)(\langle\langle \downarrow \text{ag} \rangle\rangle)\end{aligned}$$

Lexical values for other determiners, common nouns, and proper names are defined analogously. Note that we do not formally distinguish event variables from individual variables. In particular, $\mathcal{L}(\text{with})$ can be applied to either nouns or verbs, which both have type $\langle e, t \rangle$.

We assume that no node in the SC-tree has more than one child with the same edge label (which our grammar guarantees), and write $n(\ell_1, \dots, \ell_k)$ to indicate that the node n has SC-children over the edge labels $\ell_1, \dots, \ell_k$. The phrasal value for n is defined (in the most complex case) as follows:

$$\mathcal{P}(n(r, s)) = \mathcal{L}(n)(\mathcal{P}(\downarrow r))(\lambda \langle\langle n \rangle\rangle. \mathcal{P}(\downarrow s))$$

This rule implements Montague’s rule of quantification (Montague, 1974); note that $\lambda \langle\langle n \rangle\rangle$ is a binder for the variable $\langle\langle n \rangle\rangle$. Nodes that have no s-children are simply functionally applied to the phrasal semantics of their children (if any).

By way of example, consider the left-hand SC-structure in Fig. 2. If we identify each node by the word it stands for, we get the following phrasal

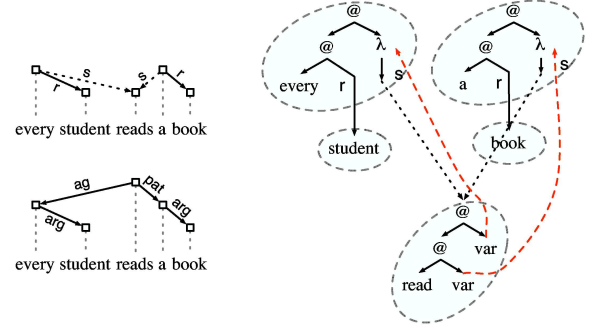


Figure 5: A partial SC-structure and its corresponding CLLS description.

value for the root of the tree:

$$\mathcal{L}(a)(\mathcal{L}(\text{book}))(\lambda x. \mathcal{L}(\text{every})(\mathcal{L}(\text{student}) (\lambda y. \text{read}'(y)(x))))$$

where we write x for $\langle\langle a \rangle\rangle$ and y for $\langle\langle \text{every} \rangle\rangle$. The arguments of read' are x and y because every and a are the arg and pat children of reads on the PA-structure. After replacing the lexical values by their definitions and beta-reduction, we obtain the familiar representation for this semantic reading, as shown in Section 3.1.

4.2 Underspecification

It is straightforward to extend this extraction of type-theoretic formulas from fully specified XDG analyses to an extraction of underspecified semantic descriptions from partial XDG analyses. We will briefly demonstrate this here for descriptions in the CLLS framework (Egg et al., 2001), which supports this most easily. Other underspecification formalisms could be used too.

Consider the partial SC-structure in Fig. 5, which could be derived by the constraint solver for the sentence from Fig. 2. We can obtain a CLLS constraint from it by first assigning to each node of the SC-structure a lexical value, which is now a part of the CLLS constraint (indicated by the dotted ellipses). Because student and book are known to be r-daughters of every and a on SC, we plug their CLLS constraints into the r-holes of their mothers’ constraints. Because we know that reads must be dominated by the s-children of the determiners, we add the two (dotted) dominance edges to the constraint. Finally, variable binding is represented by the binding constraints drawn as dashed arrows, and can be derived from PA exactly as above.

5 Conclusion

In this paper, we have shown how to build a fully relational syntax-semantics interface based on XDG.

This new grammar formalism offers the grammar developer the possibility to represent different kinds of linguistic information on separate dimensions that can be represented as graphs. Any two dimensions can be linked by multi-dimensional principles, which mutually constrain the graphs on the two dimensions. We have shown that a parser based on concurrent constraint programming is capable of inferences that restrict ambiguity on one dimension based on newly available information on another.

Because the interface we have presented makes no assumption that any dimension is more “basic” than another, there is no conceptual difference between parsing and generation. If the input is the surface sentence, the solver will use this information to compute the semantic dimensions; if the input is the semantics, the solver will compute the syntactic dimensions, and therefore a surface sentence. This means that we get bidirectional grammars for free.

While the solver is reasonably efficient for many (hand-crafted) grammars, it is an important goal for the future to ensure that it can handle large-scale grammars imported from e.g. XTAG (XTAG Research Group, 2001) or induced from treebanks. One way in which we hope to achieve this is to identify fragments of XDG with provably polynomial parsing algorithms, and which contain most useful grammars. Such grammars would probably have to specify word orders that are not completely free, and we would have to control the combinatorics of the different dimensions (Maxwell and Kaplan, 1993). One interesting question is also whether different dimensions can be compiled into a single dimension, which might improve efficiency in some cases, and also sidestep the monostratal vs. multi-stratal distinction.

The crucial ingredient of XDG that make relational syntax-semantics processing possible are the declaratively specified principles. So far, we have only given some examples for principle specifications; while they could all be written as Horn clauses, we have not committed to any particular representation formalism. The development of such a representation formalism will of course be extremely important once we have experimented with more powerful grammars and have a stable intuition about what principles are needed.

At that point, it would also be highly interesting to define a (logic) formalism that generalises both XDG and dominance constraints, a fragment of CLLS. Such a formalism would make it possible to take over the interface presented here, but use dominance constraints directly on the semantics dimensions, rather than via the encoding into PA and SC

dimensions. The extraction process of Section 4.2 could then be recast as a principle.

Acknowledgements

We thank Markus Egg for many fruitful discussions about this paper.

References

- K. Apt. 2003. *Principles of Constraint Programming*. Cambridge University Press.
- N. Chomsky. 1965. *Aspects of the Theory of Syntax*. MIT Press, Cambridge, MA.
- A. Copestake, D. Flickinger, C. Pollard, and I. Sag. 2004. Minimal recursion semantics. an introduction. *Journal of Language and Computation*. To appear.
- D. Duchier and R. Debusmann. 2001. Topological dependency trees: A constraint-based account of linear precedence. In *ACL 2001*, Toulouse.
- D. Duchier and G.-J. M. Kruijff. 2003. Information structure in topological dependency grammar. In *EACL 2003*.
- D. Duchier. 2003. Configuration of labeled trees under lexicalized constraints and principles. *Research on Language and Computation*, 1(3–4):307–336.
- M. Egg, A. Koller, and J. Niehren. 2001. The Constraint Language for Lambda Structures. *Logic, Language, and Information*, 10:457–485.
- V. Gupta and J. Lamping. 1998. Efficient linear logic meaning assembly. In *COLING/ACL 1998*.
- R. M. Kaplan and J. T. Maxwell III. 1988. An algorithm for functional uncertainty. In *COLING 1988*, pages 297–302, Budapest/HUN.
- A. Koller and J. Niehren. 2000. On underspecified processing of dynamic semantics. In *Proceedings of COLING-2000*, Saarbrücken.
- A. Koller and K. Striegnitz. 2002. Generation as dependency parsing. In *ACL 2002*, Philadelphia/USA.
- J. T. Maxwell and R. M. Kaplan. 1993. The interface between phrasal and functional constraints. *Computational Linguistics*, 19(4):571–590.
- R. Montague. 1974. The proper treatment of quantification in ordinary english. In Richard Thomason, editor, *Formal Philosophy. Selected Papers of Richard Montague*, pages 247–271. Yale University Press, New Haven and London.
- M. Moortgat. 2002. Categorical grammar and formal semantics. In *Encyclopedia of Cognitive Science*. Nature Publishing Group, MacMillan. To appear.
- Mozart Consortium. 2004. The Mozart-Oz website. <http://www.mozart-oz.org/>.
- G. Smolka. 1995. The Oz Programming Model. In *Computer Science Today*, Lecture Notes in Computer Science, vol. 1000, pages 324–343. Springer-Verlag.
- M. Steedman. 1999. Alternating quantifier scope in CCG. In *Proc. 37th ACL*, pages 301–308.
- XTAG Research Group. 2001. A lexicalized tree adjoining grammar for english. Technical Report IRCS-01-03, IRCS, University of Pennsylvania.

Extensible Dependency Grammar: A New Methodology

Ralph Debusmann
Programming Systems Lab
Saarland University
Postfach 15 11 50
66041 Saarbrücken
Germany
rade@ps.uni-sb.de

Denys Duchier
Équipe Calligramme
LORIA – UMR 7503
Campus Scientifique, B. P. 239
54506 Vandœuvre lès Nancy CEDEX
France
duchier@loria.fr

Geert-Jan M. Kruijff
Computational Linguistics
Saarland University
Postfach 15 11 50
66041 Saarbrücken
Germany
gj@coli.uni-sb.de

Abstract

This paper introduces the new grammar formalism of Extensible Dependency Grammar (XDG), and emphasizes the benefits of its methodology of explaining complex phenomena by interaction of simple principles on multiple dimensions of linguistic description. This has the potential to increase modularity with respect to linguistic description and grammar engineering, and to facilitate concurrent processing and the treatment of ambiguity.

1 Introduction

We introduce the new grammar formalism of Extensible Dependency Grammar (XDG). In XDG, complex phenomena arise out of the interaction of simple principles on multiple dimensions of linguistic description. In this paper, we point out how this novel methodology positions XDG in between multi-stratal approaches like LFG (Bresnan and Kaplan, 1982) and MTT (Mel’čuk, 1988), see also (Kahane, 2002), and mono-stratal ones like HPSG (Pollard and Sag, 1994), attempting to combine their benefits and avoid their problems.

It is the division of linguistic analyses into different dimensions which makes XDG multi-stratal. On the other, XDG is mono-stratal in that its principles interact to constrain all dimensions simultaneously. XDG combines the benefits of these two positions, and attempts to circumvent their problems. From multi-stratal approaches, XDG adopts a high degree of *modularity*, both with respect to linguistic description as well as for grammar engineering. This also facilitates the statement of cross-linguistic generalizations. XDG avoids the problem of placing too high a burden on the interfaces, and allows interactions between all and not only adjacent dimensions. From mono-stratal approaches, XDG adopts a high degree of *integration*, facilitating concurrent processing and the treatment of ambiguity. At the same time, XDG does not lose its modularity.

XDG is a descendant of Topological Dependency Grammar (TDG) (Duchier and Debusmann,

2001), pushing the underlying methodology further by generalizing it in two aspects:

- number of dimensions: two in TDG (ID and LP), arbitrary many in XDG
- set of principles: fixed in TDG, extensible principle library in XDG

The structure of this paper is as follows: In §2, we introduce XDG and the XDG solver used for parsing and generation. In §3, we introduce a number of XDG principles informally, before making use of them in an idealized example grammar in §4. In §5 we argue why XDG has the potential to be an improvement over multi-stratal and mono-stratal approaches, before we conclude in §6.

2 Extensible Dependency Grammar

In this section, we introduce XDG formally and mention briefly the constraint-based XDG solver for parsing and generation.

2.1 Formalization

Formally, an XDG grammar is built up of dimensions, a lexicon and principles, and characterizes a set of well-formed analyses.

A *dimension* is a tuple $D = (Lab, Fea, Val, Pri)$ of a set *Lab* of edge labels, a set *Fea* of features, a set *Val* of feature values, and a set of one-dimensional principles *Pri*. A *lexicon* for the dimension *D* is a set $Lex \subseteq Fea \rightarrow Val$ of total feature assignments called lexical entries. An analysis on dimension *D* is a triple (V, E, F) of a set *V* of nodes, a set $E \subseteq V \times V \times Lab$ of directed labeled edges, and an assignment $F : V \rightarrow (Fea \rightarrow Val)$ of lexical entries to nodes. *V* and *E* form a graph. We write Ana_D for the set of all possible analyses on dimension *D*. The principles characterize subsets of Ana_D . We assume that the elements of *Pri* are finite representations of such subsets.

An XDG grammar $((Lab_i, Fea_i, Val_i, Pri_i)_{i=1}^n, Pri, Lex)$ consists of *n* dimensions, multi-dimensional principles *Pri*, and a lexicon *Lex*. An XDG analysis

$(V, E_i, F_i)_{i=1}^n$ is an element of $Ana = Ana_1 \times \dots \times Ana_n$ where all dimensions share the same set of nodes V . We call a dimension of a grammar *grammar dimension*.

Multi-dimensional principles specify subsets of Ana , i.e. of tuples of analyses for the individual dimensions. The lexicon $Lex \subseteq Lex_1 \times \dots \times Lex_n$ constrains all dimensions at once, thereby synchronizing them. An XDG analysis is licensed by Lex iff $(F_1(v), \dots, F_n(v)) \in Lex$ for every node $v \in V$.

In order to compute analyses for a given input, we employ a set of *input constraints* (Inp), which again specify a subset of Ana . XDG solving then amounts to finding elements of Ana that are licensed by Lex , and consistent with Inp and Pri . The input constraints determine whether XDG solving is to be used for parsing or generation. For parsing, they specify a sequence of words, and for generation, a multiset of semantic literals.

2.2 Solver

XDG solving has a natural reading as a constraint satisfaction problem (CSP) on finite sets of integers, where well-formed analyses correspond to the solutions of the CSP (Duchier, 2003). We have implemented an XDG solver using the Mozart-Oz programming system.

XDG solving operates on all dimensions concurrently. This means that the solver can infer information about one dimension from information on another, if there is either a multi-dimensional principle linking the two dimensions, or by the synchronization induced by the lexical entries. For instance, not only can syntactic information trigger inferences in syntax, but also vice versa.

Because XDG allows us to write grammars with completely free word order, XDG solving is an NP-complete problem (Koller and Striegnitz, 2002). This means that the worst-case complexity of the solver is exponential. The average-case complexity of many smaller-scale grammars that we have experimented with seems polynomial, but it remains to be seen whether we can scale this up to large-scale grammars.

3 Principles

The well-formedness conditions of XDG analyses are stipulated by *principles*. Principles are parametrizable, e.g. by the dimensions on which they are applied, or by lexical features. They can be lexicalized or non-lexicalized, and can be one-dimensional or multi-dimensional. Principles are taken from an extensible *principle library*, and we introduce some of the most important principles in the following.

3.1 Tree principle

tree(i) The analysis on dimension i must be a tree.

The *tree principle* is non-lexicalized and parametrized by the dimension i .

3.2 Dag principle

dag(i) The analysis on dimension i must be a directed acyclic graph.

The *dag principle* is non-lexicalized and parametrized by the dimension i .

3.3 Valency principle

valency(i, in_i, out_i) All nodes on dimension i must satisfy their in and out specifications.

The *valency principle* is lexicalized and serves to lexically describe dependency graphs. It is parametrized by the dimension i , the *in specification* in_i and the *out specification* out_i . For each node, in_i stipulates the licensed incoming edges, and out_i the licensed outgoing edges.

In the example grammar lexicon part in Figure 1 below, the in specification is in_{ID} and out_{ID} is the out specification on the ID dimension. For the common noun *Roman*, the in specification licenses zero or one incoming edges labeled subj, and zero or one incoming edges labeled obj ($\{subj?, obj?\}$), i.e. it can be either a subject or an object. The out specification requires precisely one outgoing edge labeled det ($\{det!\}$), i.e. it requires a determiner.

3.4 Government principle

government($i, cases_i, govern_i$) All edges in dimension i must satisfy the government specification of the mother.

The *government principle* is lexicalized. Its purpose is to constrain the case feature of a dependent.¹ It is parametrized by the dimension i , the *cases specification* $cases_i$ and the *government specification* $govern_i$. $cases_i$ assigns to each word a set of possible cases, and $govern_i$ a mapping from labels to sets of cases.

In Figure 1, the cases specification for the determiner *den* is $\{acc\}$ (i.e. it can only be accusative). By its government specification, the finite verb *versucht* requires its subject to exhibit nominative case ($subj \mapsto \{nom\}$).

3.5 Agreement principle

agreement($i, cases_i, agree_i$) All edges in dimension i must satisfy the agreement specification of the mother.

¹We restrict ourselves to the case feature only for simplicity. In a fully-fledged grammar, the government principle would be used to constrain also other morphological aspects like number, person and gender.

The *agreement principle* is lexicalized. Its purpose is to enforce the case agreement of a daughter.² It is parametrized by dimension i , the lexical *cases specification* cases_i , assigning to each word a set of possible cases, and the *agreement specification* agree_i , assigning to each word a set of labels.

As an example, in Figure 1, the agreement specification for the common noun *Roman* is $\{\text{det}\}$, i.e. the case of the common noun must agree with its determiner.

3.6 Order principle.

$\text{order}(i, \text{on}_i, \prec_i)$ On dimension i , 1) each node must satisfy its node labels specification, 2) the order of the daughters of each node must be compatible with $\prec_i$, and 3) the node itself must be ordered correctly with respect to its daughters (using its node label).

The *order principle* is lexicalized. It is parametrized by the dimension i , the *node labels specification* on_i mapping each node to set of labels from Lab_i , and the total order $\prec_i$ on Lab_i .

Assuming the node labels specification given in Figure 2, and the total order in (5), the tree in (11) satisfies the order principle.³ For instance for the node *versucht*: 1) The node label of *versucht* is lbf, satisfying the node labels specification. 2) The order of the daughters *Roman* (under the edge labeled vf), *Peter* (mf) and *lesen* (rbf) is compatible with the total order prescribing $\text{vf} \prec \text{mf} \prec \text{rbf}$. 3) The node *versucht* itself is ordered correctly with respect to its daughters (the total order prescribes $\text{vf} \prec \text{lbf} \prec \text{mf}$).

3.7 Projectivity principle

$\text{projectivity}(i)$ The analysis on dimension i must be projective.

The *projectivity principle* is non-lexicalized. Its purpose is to exclude non-projective analyses.⁴ It is parametrized by dimension i .

3.8 Climbing principle

$\text{climbing}(i, j)$ The graph on dimension i must be flatter than the graph on dimension j .

The *climbing principle* is non-lexicalized and two-dimensional. It is parametrized by the two dimensions i and j .

For instance, the tree in (11) is flatter than the corresponding tree in (10). This concept was introduced as *lifting* in (Kahane et al., 1998).

²Again, we restrict ourselves to case for simplicity.

³The node labels are defined in (2) below.

⁴The projectivity principle of course only makes sense in combination with the order principle.

3.9 Linking principle

$\text{linking}(i, j, \text{link}_{i,j})$ All edges on dimension i must satisfy the linking specification of the mother.

The *linking principle* is lexicalized and two-dimensional. It is parametrized by the two dimensions i and j , and by the *linking specification* $\text{link}_{i,j}$, mapping labels from Lab_i to sets of labels from Lab_j . Its purpose is to specify how dependents on dimension i are realized by (or linked to) dependents on dimension j .

In the lexicon part in Figure 3, the linking specification for the transitive verb *lesen* requires that its agent on the PA dimension must be realized by a subject ($\text{ag} \mapsto \{\text{subj}\}$), and the patient by an object ($\text{pat} \mapsto \{\text{obj}\}$).

The linking principle is oriented. Symmetric linking could be gained simply by using the linking principle twice (in both directions).

4 Example grammar

In this section, we elucidate XDG with an example grammar fragment for German. With it, we demonstrate three aspects of the methodology of XDG:

- How complex phenomena such as topicalization and control arise by the interaction of simple principles on different dimensions of linguistic description.
- How the high degree of integration helps to reduce ambiguity.
- How the high degree of modularity facilitates the statement of cross-linguistic generalizations.

Note that this grammar fragment is an idealized example, and does not make any claims about XDG as a *grammar theory*. Its purpose is solely to substantiate our points about XDG as a *framework*. Moreover, the grammar is fully lexicalized for simplicity. However, XDG of course allows the grammar writer to formulate lexical abstractions using inheritance (like in HPSG) or crossings (Candito, 1996).

4.1 Dimensions

The grammar fragment make use of two dimensions: Immediate Dominance (ID) and Linear Precedence (LP). The models on the ID dimension are unordered, syntactic dependency trees whose edge labels correspond to syntactic functions like *subject* and *object*. On the LP dimension, the models are ordered, projective topological dependency trees whose edge labels are topological fields like *Vorfeld* and *Mittelfeld*.

4.2 Labels

The set Lab_{ID} of labels on the ID dimension is:

$$Lab_{ID} = \{\text{det}, \text{subj}, \text{obj}, \text{vinf}, \text{part}\} \quad (1)$$

These correspond resp. to determiner, subject, object, infinitive verbal complement, and particle.

The set Lab_{LP} of labels on the LP dimension is:

$$Lab_{LP} = \{\text{detcf}, \text{nounf}, \text{vf}, \text{lbf}, \text{mf}, \text{partf}, \text{rbf}\} \quad (2)$$

Corresponding resp. to determiner field, noun field, Vorfeld, left bracket field, Mittelfeld, particle field, and right bracket field.

4.3 Principles

On the ID dimension, we make use of the following one-dimensional principles:

$$\begin{aligned} & \text{tree}(\text{ID}) \\ & \text{valency}(\text{ID}, \text{in}_{\text{ID}}, \text{out}_{\text{ID}}) \\ & \text{government}(\text{ID}, \text{cases}_{\text{ID}}, \text{govern}_{\text{ID}}) \\ & \text{agreement}(\text{ID}, \text{cases}_{\text{ID}}, \text{agree}_{\text{ID}}) \end{aligned} \quad (3)$$

The LP dimension uses the following principles:

$$\begin{aligned} & \text{tree}(\text{LP}) \\ & \text{valency}(\text{LP}, \text{in}_{\text{LP}}, \text{out}_{\text{LP}}) \\ & \text{order}(\text{LP}, \text{on}_{\text{LP}}, \prec_{\text{LP}}) \\ & \text{projectivity}(\text{LP}) \end{aligned} \quad (4)$$

where the total order $\prec_{\text{LP}}$ is defined as:

$$\text{detcf} \prec \text{nounf} \prec \text{vf} \prec \text{lbf} \prec \text{mf} \prec \text{partf} \prec \text{rbf} \quad (5)$$

We make use of the following multi-dimensional principles:

$$\begin{aligned} & \text{climbing}(\text{LP}, \text{ID}) \\ & \text{linking}(\text{LP}, \text{ID}) \end{aligned} \quad (6)$$

4.4 Lexicon

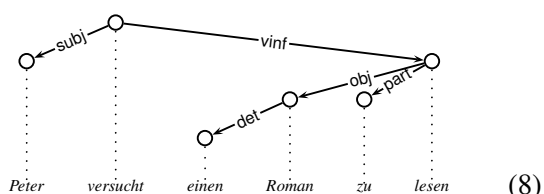
We split the lexicon into two parts. The ID and LP parts are displayed resp. in Figure 1⁵ and Figure 2. The LP part includes also the linking specification for the LP, ID-application of the linking principle.⁶

4.5 Government and agreement

Our first example is the following sentence:

$$\begin{aligned} & \text{Peter} \quad \text{versucht} \quad \text{einen} \quad \text{Roman} \quad \text{zu} \quad \text{lesen.} \\ & \text{Peter} \quad \text{tries} \quad \text{aacc} \quad \text{novel} \quad \text{to} \quad \text{read.} \\ & \text{Peter tries to read a novel.} \end{aligned} \quad (7)$$

We display the ID analysis of the sentence below:



⁵Here, $_$ stands for “don’t care”, this means e.g. for the verb *versucht* that it has unspecified case.

⁶We do not make use of the linking specification for the German grammar fragment (the mappings are all empty), but we will do so as we switch to Dutch in §4.8 below.

Here, *Peter* is the subject of *versucht*. *lesen* is the infinitival verbal complement of *versucht*, *zu* the particle of *lesen*, and *Roman* the object of *lesen*. Finally, *einen* is the determiner of *Roman*.

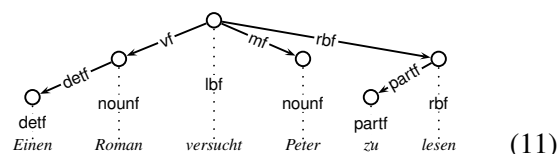
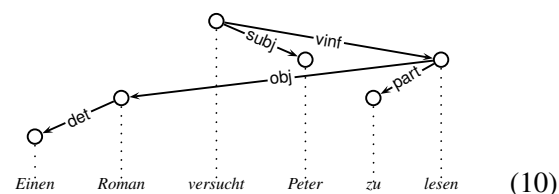
Under our example grammar, the sentence is unambiguous, i.e. the given ID tree is the only possible one. Other ID trees are ruled out by the interaction of the principles on the ID dimension. For instance, the government and agreement principles conspire to rule out the reading where *Roman* is the subject of *versucht* (and *Peter* the object). How? By the agreement principle, *Roman* must be accusative, since it agrees with its accusative determiner *einen*. By the government principle, the subject of *versucht* must be nominative, and the object of *lesen* accusative. Thus *Roman*, by virtue of being accusative, cannot become the subject of *versucht*. The only other option for it is to become the object of *lesen*. Consequently, *Peter*, which is unspecified for case, must become the subject of *versuchen* (*versuchen* must have a subject by the valency principle).

4.6 Topicalization

Our second example is a case of topicalization, where the object has moved into the Vorfeld, to the left of the finite verb:

$$\text{Einen} \quad \text{Roman} \quad \text{versucht} \quad \text{Peter} \quad \text{zu} \quad \text{lesen.} \quad (9)$$

Here is the ID tree and the LP tree analysis:



The ID tree analysis is the same as before, except that the words are shown in different positions. In the LP tree, *Roman* is in the Vorfeld of *versucht*, *Peter* in the Mittelfeld, and *lesen* in the right bracket field. *versucht* itself is (by its node label) in the left bracket field. Moreover, *Einen* is in the determiner field of *Roman*, and *zu* in the particle field of *lesen*.

Again, this is an example demonstrating how complex phenomena (here: topicalization) are explained by the interaction of simple principles. Topicalization does not have to explicitly taken care of, it is rather a consequence of the interacting principles. Here, the valency, projectivity and climbing

	in _{ID}	out _{ID}	cases _{ID}	govern _{ID}	agree _{ID}
<i>den</i>	{det?}	{}	{acc}	{}	{}
<i>Roman</i>	{subj?, obj?}	{det!}	{nom, dat, acc}	{}	{det}
<i>Peter</i>	{subj?, obj?}	{}	{nom, dat, acc}	{}	{}
<i>versucht</i>	{}	{subj!, vinf!}	-	{subj $\mapsto$ {nom}}	{}
<i>zu</i>	{part?}	{}	-	{}	{}
<i>lesen</i>	{vinf?}	{obj!}	-	{obj $\mapsto$ {acc}}	{}

Figure 1: Lexicon for the example grammar fragment, ID part

	in _{LP}	out _{LP}	on _{LP}	link _{LP, ID}
<i>den</i>	{detf?}	{}	{detf}	{}
<i>Roman</i>	{vf?, mf?}	{detf!}	{nounf}	{}
<i>Peter</i>	{vf?, mf?}	{}	{nounf}	{}
<i>versucht</i>	{}	{vf?, mf*, rbf?}	{lbf}	{}
<i>zu</i>	{partf?}	{}	{partf}	{}
<i>lesen</i>	{rbf?}	{}	{rbf}	{}

Figure 2: Lexicon for the example grammar fragment, LP part

principles conspire to bring about the “climbing up” of the NP *Einen Roman* from being the daughter of *lesen* in the ID tree to being the daughter of *versucht* in the LP tree: The out specification of *lesen* does not license any outgoing edge. Hence, *Roman* must become the daughter of another node. The only possibility is *versucht*. The determiner *Einen* must then also “climb up” because *Roman* is its only possible mother. The result is an LP tree which is flatter with respect to the ID tree. The LP tree is also projective. If it were not be flatter, then it would be non-projective, and ruled out by the projectivity principle.

4.7 Negative example

Our third example is a negative example, i.e. an ungrammatical sentence:

**Peter einen Roman versucht zu lesen.* (12)

This example is perfectly legal on the unordered ID dimension, but has no model on the LP dimension. Why? Because by its LP out specification, the finite verb *versucht* allows only one dependent to the left of it (in its Vorfeld), and here we have two. The interesting aspect of this example is that although we can find a well-formed ID tree for it, this ID tree is never actually generated. The interactions of the principles, viz. here of the principles on the LP dimension, rule out the sentence *before* any full ID analysis has been found.

4.8 From German to Dutch

For the fourth example, we switch from German to Dutch. We will show how to use the lexicon to concisely capture an important cross-linguistic generalization. We keep the same grammar as before, but with two changes, arising from the lesser degree of

inflection and the higher reliance on word order in Dutch:

- The determiner *een* is not case-marked but can be either nominative, dative or accusative: $\text{cases}_{ID} = \{\text{nom}, \text{dat}, \text{acc}\}$.
- The Vorfeld of the finite verb *probeert* cannot be occupied by an object (but only by an object): $\text{link}_{LP, ID} = \{\text{vf} \mapsto \{\text{subj}\}\}$.⁷

Now to the example, a Dutch translation of (7):

Peter probeert een roman te lezen.
Peter tries a novel to read. (13)
Peter tries to read a novel.

We get only one analysis on the ID dimension, where *Peter* is the subject and *roman* the object. An analysis where *Peter* is the object of *lezen* and *roman* the subject of *probeert* is impossible, as in the German example. The difference is, however, how this analysis is excluded. In German, the accusative inflection of the determiner *einen* triggered the agreement and the government principle to rule it out. In Dutch, the determiner is not inflected. The unwanted analysis is excluded on the grounds of word order instead: By the linking principle, the Vorfeld of *probeert* must be filled by a subject, and not by an object. That means that *Peter* in the Vorfeld (to the left of *probeert*) must be a subject, and consequently, the only other choice for *roman* is that it becomes the object of *lezen*.

4.9 Predicate-Argument Structure

Going towards semantics, we extend the grammar with another dimension, Predicate-Argument Structure (PA), where the models are not trees but directed acyclic graphs (dags), to model re-entrancies

⁷Of course, this is an idealized assumption. In fact, given the right stress, the Dutch Vorfeld *can* be filled by objects.

e.g. caused by control constructions. Thanks to the modularity of XDG, the PA part of the grammar is the same for German and Dutch.

The set Lab_{PA} of labels on the PA dimension is:

$$Lab_{PA} = \{ag, pat, prop\} \quad (14)$$

Corresponding resp. to agent, patient and proposition.

The PA dimension uses the following one-dimensional principles:

$$\begin{aligned} dag(PA) \\ valency(PA, in_{PA}, out_{PA}) \end{aligned} \quad (15)$$

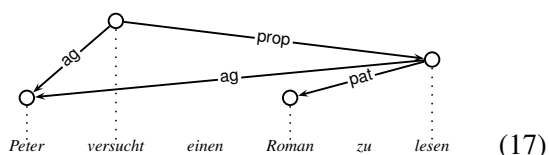
Note that we re-use the valency principle again, as we did on the ID and LP dimensions.

And also the following multi-dimensional principles:

$$\begin{aligned} climbing(ID, PA) \\ linking(PA, ID) \end{aligned} \quad (16)$$

Here, we re-use the climbing and linking principles. That is, we state that the ID tree is flatter than the corresponding PA dag. This captures raising and control, where arguments of embedded infinite verbs can “climb up” and become arguments of a raising or control verb, in the same way as syntactic arguments can “climb up” from ID to LP. We use the linking principle to specify how semantic arguments are to be realized syntactically (e.g. the agent as a subject etc.). We display the PA part of the lexicon in Figure 3.⁸

Here is an example PA dag analysis of example sentence (7):



Here, *Peter* is the agent of *versucht*, and also the agent of *lesen*. Furthermore, *lesen* is a proposition dependent of *versucht*, and *Roman* is the patient of *lesen*.

Notice that the PA dag is indeed a dag and not a tree since *Peter* has two incoming edges: It is simultaneously the agent of *versucht* and of *lesen*. This is enforced by the valency principle: Both *versucht* and *lesen* require an agent. *Peter* is the only word which can be the agent of both, because it is a subject and the agents of *versucht* and *lesen* must be subjects by the linking principle. The climbing principle ensures that predicate arguments can

⁸Notice that we specify linking lexically, allowing us to capture deviations from the typical linking patterns. Still, we can also accommodate linking generalizations using lexical abstractions.

be “raised” on the ID structure with respect to the PA structure. Again, this example demonstrates that XDG is able to reduce a complex phenomenon such as control to the interaction of per se fairly simple principles such as valency, climbing and linking.

5 Comparison

This section includes a more in-depth comparison of XDG with purely multi- and mono-stratal approaches.

Contrary to multi-stratal approaches like LFG or MTT, XDG is more integrated. For one, it places a lighter burden the interfaces between the dimensions. In LFG for instance, the ϕ -mapping from c-structure to f-structure is rather specific, and has to be specifically adapted to new c-structures, e.g. in order to handle a new construction with a different word order. That is, not only the grammar rules for the c-structure need to be adapted, but also the interface between c- and f-structure. In XDG, complex phenomena arise out of the interaction of simple, maximally general principles. To accommodate the new construction, the grammar would ideally only need to be adapted on the word order dimension.

Furthermore, XDG allows interactions of relational constraints between all dimensions, not only between adjacent ones (like c- and f-structure), and in all directions. For one, this gets us bi-directionality for free. Secondly, the interactions of XDG have the potential to help greatly in reducing ambiguity. In multi-stratal approaches, ambiguity must be duplicated throughout the system. E.g. suppose there are two candidate c-structures in LFG parsing, but one is ill-formed semantically. Then they can only be ruled out after duplicating the ambiguity on the f-structure, and then filtering out the ill-formed structure on the semantic σ -structure. In XDG on the other hand, the semantic principles can rule out the ill-formed analysis much earlier, typically on the basis of a partial syntactic analysis. Thus, ill-formed analyses are never duplicated.

Contrary to mono-stratal ones, XDG is more modular. For one, as (Oliva et al., 1999) note, mono-stratal approaches like HPSG usually give precedence to the syntactic tree structure, while putting the description of other aspects of the analysis on the secondary level only, by means of features spread over the nodes of the tree. As a result, it becomes a hard task to modularize grammars. Because syntax is privileged, the phenomena ascribing to semantics cannot be described independently, and whenever the syntax part of the grammar changes, the semantics part needs to be adapted. In XDG, no dimension is privileged to another. Semantic phe-

	in _{PA}	out _{PA}	link _{PA,ID}
<i>den</i>	{}	{}	{}
<i>Roman</i>	{ag?, pat?}	{}	{}
<i>Peter</i>	{ag?, pat?}	{}	{}
<i>versucht</i>	{}	{ag!, prop!}	{ag $\mapsto$ {subj}, prop $\mapsto$ {vinf}}
<i>zu</i>	{}	{}	{}
<i>lesen</i>	{prop?}	{ag!, pat!}	{ag $\mapsto$ {subj}, pat $\mapsto$ {obj}}

Figure 3: Lexicon of the example grammar fragment, PA part

nomena can be described much more independently from syntax. This facilitates grammar engineering, and also the statement of cross-linguistic generalizations. Assuming that the semantics part of a grammar stay invariant for most natural languages, in order to accommodate a new language, ideally only the syntactic parts would need to be changed.

6 Conclusion

In this paper, we introduced the XDG grammar framework, and emphasized that its new methodology places it in between the extremes of multi- and mono-stratal approaches. By means of an idealized example grammar, we demonstrated how complex phenomena are explained as arising from the interaction of simple principles on numerous dimensions of linguistic description. On the one hand, this methodology has the potential to modularize linguistic description and grammar engineering, and to facilitate the statement of linguistic generalizations. On the other hand, as XDG is a inherently concurrent architecture, inferences from any dimension can help reduce the ambiguity on others.

XDG is a new grammar formalism, and still has many open issues. Firstly, we need to continue work on XDG as a framework. Here, one important goal is to find out what criteria we can give to restrict the principles. Secondly, we need to evolve the XDG grammar theory, and in particular the XDG syntax-semantics interface. Thirdly, for practical use, we need to improve our knowledge about XDG solving (i.e. parsing and generation). So far, our only good results are for smaller-scale handwritten grammars, and we have not good results yet for larger-scale grammars induced from treebanks (NEGRA, PDT) or converted from other grammar formalisms (XTAG). Finally, we need to incorporate statistics into the picture, e.g. to guide the search for solutions, in the vein of (Dienes et al., 2003).

References

- Joan Bresnan and Ronald Kaplan. 1982. Lexical-functional grammar: A formal system for grammatical representation. In Joan Bresnan, editor, *The Mental Representation of Grammatical Re-*
- lations*, pages 173–281. The MIT Press, Cambridge/USA.
- Marie-Hélène Candito. 1996. A principle-based hierarchical representation of LTAG. In *Proceedings of COLING 1996*, Copenhagen/DEN.
- Peter Dienes, Alexander Koller, and Marco Kuhlmann. 2003. Statistical A* Dependency Parsing. In *Prospects and Advances in the Syntax/Semantics Interface*, Nancy/FRA.
- Denys Duchier and Ralph Debusmann. 2001. Topological dependency trees: A constraint-based account of linear precedence. In *Proceedings of ACL 2001*, Toulouse/FRA.
- Denys Duchier. 2003. Configuration of labeled trees under lexicalized constraints and principles. *Research on Language and Computation*, 1(3–4):307–336.
- Sylvain Kahane, Alexis Nasr, and Owen Rambow. 1998. Pseudo-projectivity: a polynomially parsable non-projective dependency grammar. In *36th Annual Meeting of the Association for Computational Linguistics (ACL 1998)*, Montréal/CAN.
- Sylvain Kahane. 2002. *Grammaire d’Unification Sens-Texte: Vers un modèle mathématique articulé de la langue*. Université Paris 7. Document de synthèse de l’habilitation à diriger les recherches.
- Alexander Koller and Kristina Striegnitz. 2002. Generation as dependency parsing. In *Proceedings of ACL 2002*, Philadelphia/USA.
- Igor Mel’čuk. 1988. *Dependency Syntax: Theory and Practice*. State Univ. Press of New York, Albany/USA.
- Karel Oliva, M. Andrew Moshier, and Sabine Lehmann. 1999. Grammar engineering for the next millennium. In *Proceedings of the 5th Natural Language Processing Pacific Rim Symposium 1999 “Closing the Millennium”*, Beijing/CHI. Tsinghua University Press.
- Carl Pollard and Ivan A. Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Chicago/USA.

A Parser System for Extensible Dependency Grammar

Ralph Debusmann

Programming Systems Lab
Universität des Saarlandes, Geb. 45
Postfach 15 11 50
66041 Saarbrücken, Germany
`rade@ps.uni-sb.de`

Abstract

This paper introduces a parser system for the meta grammar formalism of Extensible Dependency Grammar (XDG). XDG is a generalisation of Topological Dependency Grammar (TDG) (Duchier and Debusmann, 2001). The XDG parser system comprises a constraint-based parser for all possible instances of XDG, a statically typed grammar input language, and a flexible backend for handling parser output. A powerful graphical user interface provides for easy accessibility of all the functionality of the system. In the future, we will use the XDG parser system to accommodate new dependency grammar formalisms such as Semantic Topological Dependency Grammar (STDG), and to experiment with other interesting XDG instances.

1 Introduction

Extensible Dependency Grammar (XDG) is a new meta grammar formalism for dependency grammar (Tesnière, 1959). XDG is based on the grammar formalism of Topological Dependency Grammar (TDG) (Duchier and Debusmann, 2001). XDG can be likened to Head-driven Phrase Structure Grammar (HPSG) (Pollard and Sag, 1994): both XDG and HPSG are meta grammar formalisms which need to be instantiated with particular principles and parameters to obtain specific grammar formalisms. The fundamental difference is that whereas HPSG is based on typed feature structures, XDG is based on graph descriptions.

A parser system for TDG can be found in (Duchier and Debusmann, 2002). The parser uses constraint-based techniques developed in (Duchier, 1999) and (Duchier, 2002) to achieve polynomial parse time in the average-case, although the TDG parsing problem has proven to be NP-complete (Koller and Striegnitz, 2002) in the worst-case. The TDG parser system is written using the Mozart-Oz programming language (Moz, 1998).

In this paper, we present a parser system for XDG. The system includes a constraint-based parser for all possible instances of XDG, a statically typed grammar input language, and a flexible backend for handling parser output. The XDG parser is a generalisation of the TDG parser, and achieves the same polynomial parse time in the average case for the TDG instance. The XDG parser system is also written in Mozart-Oz.

The XDG parser system has at least two purposes. On the one hand, we use it for parsing grammars written in Semantic Topological Dependency Grammar (STDG), an extension of TDG with a syntax-semantics interface to underspecified semantics in the Constraint Language for Lambda Structures (CLLS) (Egg et al., 1998). On the other, we provide the parser system as a tool for research on various interesting dependency grammar formalisms which are instantiations of XDG.

The structure of the paper is as follows: we briefly introduce XDG in section 2. In section 3, we present the XDG parser, and proceed to present the frontend and backend of the system in sections 4 and 5. We introduce the graphical user interface (GUI) of the system in section 6, before we conclude in section 7.

2 Extensible Dependency Grammar (XDG)

XDG is a graph description language for a set of *graph dimensions*, where each graph dimension corresponds to a graph. All graphs share the same set of nodes, but have different edges and different edge labels. Simple feature structures can be attached to each node, and by lexicalisation, XDG can be turned into a powerful dependency-based meta grammar formalism.

The well-formedness conditions of an XDG analysis are stipulated by parametrised *principles*. Each dimension uses a set of principles, stipulating restrictions on the licensed graphs on this dimension. Principles can also pose restrictions on any number of dimensions simultaneously. An XDG analysis is well-formed only if all used principles are satisfied on all dimensions. Principles are taken from a shared and extensible *principle library*.

The recipe to get an instance of XDG is easy: 1) define the set of used graph dimensions, and 2) define the set of used principles (and their parameters). The TDG instance uses two dimensions, ID (Immediate Dominance) and LP (Linear Precedence). On the ID dimension, we use the tree, in (accepted edges) and out (valency) principles. On the LP dimension, we use the tree, in, out, projectivity, climbing and barriers principles. A detailed description of these principles can be found in (Duchier, 2002).

3 XDG parser

The XDG parser is at the heart of the XDG parser system. It can be instantiated to yield parsers for any instance of XDG, including TDG. The parser is a generalisation of the TDG parser (Duchier and Debusmann, 2002), and makes use of the same programming techniques, originally developed in (Duchier, 1999) and (Duchier, 2002).

The XDG parser can handle any number of dimensions, and any number of principles used on them. It provides a predefined *principle library*, including implementations of all the principles required for the TDG instance of XDG. The principle library can be freely extended with new principles written in Mozart-Oz.

Each principle in the principle library consists of a number of Oz functors which implement the principle. Therefore, it is not only possible to replace whole principles with new ones, but also to only partially substitute parts of principles with others.¹

The parser retains the good average-case performance of the TDG parser for the TDG instance and small test grammars. As with the TDG parser, the lack of bigger grammars has as of yet prohibited proper evaluation. However, we are currently working on the creation of bigger grammars for Czech, English and German.

4 Frontend

The XDG parser system frontend provides a input language for grammars called IL (Intermediate Language). A grammar defines 1) an instance of XDG (i.e. a set of used dimensions and a set of used principles and parameters), and 2) a lexicon. The IL is statically typed to ease the discovery of errors.

Grammars written in the IL are hard to read for humans. This is why the parser system arranges for the definition of input languages on top of the IL. At the moment, the system provides a language called UL (User Language) which is much better suited to write grammars in. We also provide an emacs mode for the UL. In the near future, we will provide an XML-based input language to improve interoperability.

Grammars written in any language are first compiled into the IL, and then into the PL (Parser Language), which is the language used in the XDG parser itself.²

Also new to the XDG parser system is the inclusion of features originating from Metagrammar (Candito, 1996). Most notably, it is possible to use disjunction almost everywhere in the lexicon to specify a non-deterministic choice. Together with conjunction (which amounts to lexical inheritance

¹This is particularly useful for non-deterministic search: the existing functor implementing naive search can easily be replaced by a functor implementing oracle-guided best-first search.

²At this stage, one big advantage of the new XDG parser system over the old TDG parser system is the ability to save precompiled grammars to disk, such that there is no need to compile the grammars again and again before use.

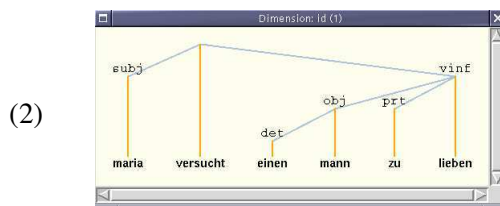
of lexical types), the frontend provides for the succinct description of a number of lexical entries by just one metagrammatical expression. Lexical economy is further improved by parametrised lexical types. Below is an example metagrammatical expression written in the User Language UL:

(1) `finite($ third sg) &
(canonical | noncanonical)`

The expression uses three lexical types (`finite`, `canonical` and `noncanonical`). `finite` is parametrised by an agreement expression (here: `$ third sg` for *third person singular*), and stands for a finite verb. `canonical` and `noncanonical` stand for *canonical position* and *non-canonical position* respectively. The expression defines a set of lexical entries which are finite verbs in third person singular inflection, and which are in canonical position or in non-canonical position. After compilation, this yields at least two distinct lexical entries, one for the finite verb in canonical position, and one in non-canonical position. Both entries will have third person singular inflection.

5 Backend

After parsing, a parser system needs to provide a means to conveniently display the parsing result. This is the role of the backend of the XDG parser system. We provide an *output library* of output functors to display the result of a parse either graphically or textually. It is also possible to get output which is directly usable in $\text{\LaTeX}$ documents. Below in (2), we show an example graph output:

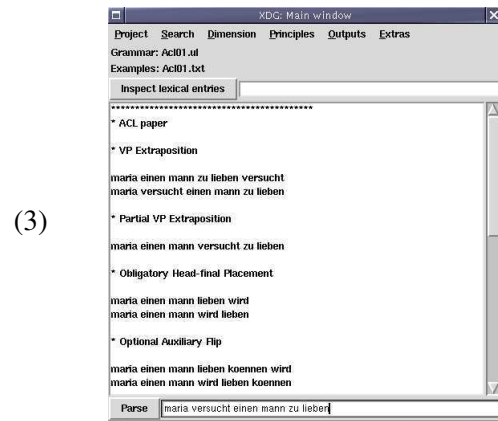


The output library is also extensible. We provide means to easily write own output functors in Mozart-Oz, to display both partial and full parses.

6 GUI

The GUI of the XDG parser system makes all functionality easily accessible and simplifies

grammar debugging. It allows to precompile grammars, load them and create parsing log files. Grammars can be loaded in any of the available input languages, and all available output functors of the backend can be used. We display an example screenshot in (3) below:



7 Conclusions

We introduced a new parser system for Extensible Dependency Grammar (XDG), as a generalisation of the existing TDG parser system (Duchier and Debusmann, 2002). The XDG parser system can be found in (Debusmann and Duchier, 2003). It improves in almost all respects on the TDG parser system: the XDG parser per se is much more flexible and extensible, and can cater for all instances of the XDG meta grammar formalism. The frontend is also much more flexible and includes features from Metagrammar to significantly improve lexical economy. The backend is much more flexible too, and also easily extensible.

We hope that the XDG parser system encourages many people to join working on XDG and its various instances.

References

- M.H. Candito. 1996. A principle-based hierarchical representation of LTAG. In *COLING 1996 Proceedings*, Kopenhagen/DEN.
- Ralph Debusmann and Denys Duchier. 2003. eXtensible dependency grammar. <http://www.mozart-oz.org/mogul/info/debusmann/xdg.html>.

- Denys Duchier and Ralph Debusmann. 2001. Topological dependency trees: A constraint-based account of linear precedence. In *ACL 2001 Proceedings*.
- Denys Duchier and Ralph Debusmann. 2002. Topological Dependency Grammar 1.2. <http://www.mozart-oz.org/mogul/info/duchier/coli/dg.html>.
- Denys Duchier. 1999. Axiomatizing dependency parsing using set constraints. In *Sixth Meeting on the Mathematics of Language*, Orlando/FL.
- Denys Duchier. 2002. Configuration of labeled trees under lexicalized constraints and principles. To appear in the *Journal of Language and Computation*.
- Markus Egg, Joachim Niehren, Peter Ruhrberg, and Feiyu Xu. 1998. Constraints over lambda-structures in semantic underspecification. In *Proceedings of COLING/ACL 1998*, pages 353–359, Montreal/CAN.
- Alexander Koller and Kristina Striegnitz. 2002. Generation as dependency parsing. In *Proceedings of ACL 2002*.
1998. Mozart. <http://www.mozart-oz.org/>.
- Carl Pollard and Ivan Sag. 1994. *Head-Driven Phrase Structure Grammar*. University of Chicago Press, Chicago.
- Lucien Tesnière. 1959. *Éléments de Syntaxe Structurale*.

Topological Dependency Trees: A Constraint-Based Account of Linear Precedence

Denys Duchier

Programming Systems Lab
Universität des Saarlandes, Geb. 45
Postfach 15 11 50
66041 Saarbrücken, Germany
duchier@ps.uni-sb.de

Ralph Debusmann

Computational Linguistics
Universität des Saarlandes, Geb. 17
Postfach 15 11 50
66041 Saarbrücken, Germany
rade@coli.uni-sb.de

Abstract

We describe a new framework for dependency grammar, with a modular decomposition of immediate dependency and linear precedence. Our approach distinguishes two orthogonal yet mutually constraining structures: a *syntactic dependency tree* and a *topological dependency tree*. The syntax tree is non-projective and even non-ordered, while the topological tree is projective and partially ordered.

1 Introduction

Linear precedence in so-called free word order languages remains challenging for modern grammar formalisms. To address this issue, we propose a new framework for dependency grammar which supports the modular decomposition of immediate dependency and linear precedence. Duchier (1999) formulated a constraint-based axiomatization of dependency parsing which characterized well-formed syntax trees but ignored issues of word order. In this article, we develop a complementary approach dedicated to the treatment of linear precedence.

Our framework distinguishes two orthogonal, yet mutually constraining structures: a *syntactic dependency tree* (ID tree) and a *topological dependency tree* (LP tree). While edges of the ID tree are labeled by syntactic roles, those of the LP tree are labeled by topological fields (Bech, 1955). The shape of the LP tree is a flattening of the ID tree's obtained by allowing nodes to 'climb up' to land in an appropriate field at a host node where that field is available. Our theory of ID/LP

trees is formulated in terms of (a) lexicalized constraints and (b) principles governing e.g. climbing conditions.

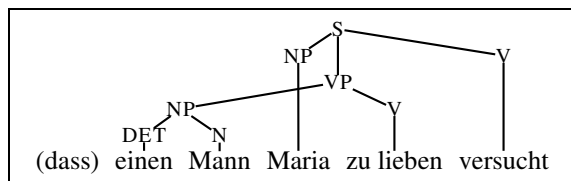
In Section 2 we discuss the difficulties presented by discontinuous constructions in free word order languages, and briefly touch on the limitations of Reape's (1994) popular theory of 'word order domains'. In Section 3 we introduce the concept of topological dependency tree. In Section 4 we outline the formal framework for our theory of ID/LP trees. Finally, in Section 5 we illustrate our approach with an account of the word-order phenomena in the verbal complex of German verb final sentences.

2 Discontinuous Constructions

In free word order languages, discontinuous constructions occur frequently. German, for example, is subject to *scrambling* and *partial extraposition*. In typical phrase structure based analyses, such phenomena lead to e.g. discontinuous VPs:

- (1) (dass) einen Mann Maria zu lieben versucht
(that) a man_{acc} Maria_{nom} to love tries

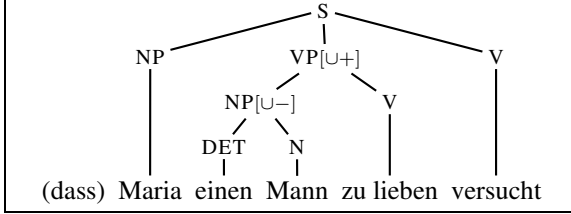
whose natural syntax tree exhibits crossing edges:



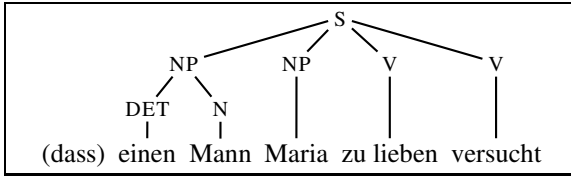
Since this is classically disallowed, discontinuous constituents must often be handled indirectly through grammar extensions such as traces.

Reape (1994) proposed the theory of *word order domains* which became quite popular in the HPSG community and inspired others such as Müller (1999) and Kathol (2000). Reape distinguished two orthogonal tree structures: (a) the unordered syntax tree, (b) the totally ordered tree of

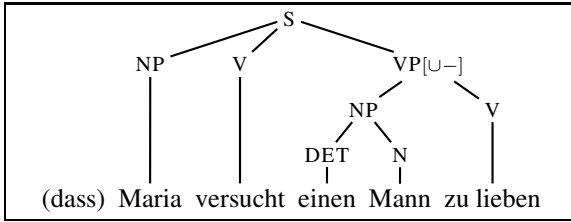
word order domains. The latter is obtained from the syntax tree by *flattening* using the operation of *domain union* to produce arbitrary interleavings. The boolean feature $[\cup\pm]$ of each node controls whether it must be flattened out or not. Infinitives in canonical position are assigned $[\cup+]$:



Thus, the above licenses the following tree of word order domains:



Extrapolated infinitives are assigned $[\cup-]$:

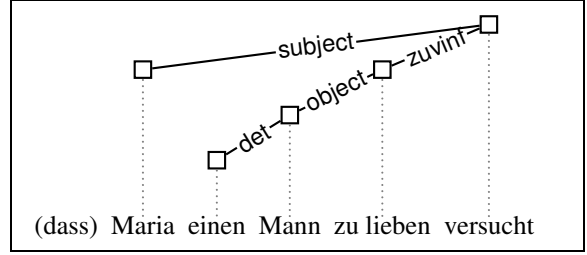


As a consequence, Reape's theory correctly predicts scrambling (2,3) and full extraposition (4), but cannot handle the partial extraposition in (5):

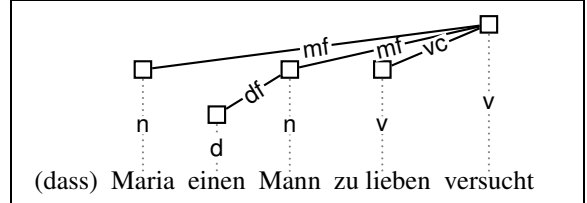
- (2) (dass) Maria einen Mann zu lieben versucht
- (3) (dass) einen Mann Maria zu lieben versucht
- (4) (dass) Maria versucht, einen Mann zu lieben
- (5) (dass) Maria einen Mann versucht, zu lieben

3 Topological Dependency Trees

Our approach is based on dependency grammar. We also propose to distinguish two structures: (a) a tree of syntactic dependencies, (b) a tree of topological dependencies. The syntax tree (ID tree) is unordered and non-projective (i.e. it admits crossing edges). For display purposes, we pick an arbitrary linear arrangement:



The topological tree (LP tree) is partially ordered and projective:



Its edge labels are called (*external*) *fields* and are totally ordered: $df \prec mf \prec vc$. This induces a linear precedence among the daughters of a node in the LP tree. This precedence is partial because daughters with the same label may be freely permuted.

In order to obtain a linearization of a LP tree, it is also necessary to position each node with respect to its daughters. For this reason, each node is also assigned an *internal* field (d, n, or v) shown above on the vertical pseudo-edges. The set of internal and external fields is totally ordered: $d \prec df \prec n \prec mf \prec vc \prec v$

Like Reape, our LP tree is a flattened version of the ID tree (Reape, 1994; Uszkoreit, 1987), but the flattening doesn't happen by 'unioning up'; rather, we allow each individual daughter to *climb up* to find an appropriate landing place. This idea is reminiscent of GB, but, as we shall see, proceeds rather differently.

4 Formal Framework

The framework underlying both ID and LP trees is the configuration of labeled trees under valency (and other) constraints. Consider a finite set $\mathcal{L}$ of edge labels, a finite set V of nodes, and $E \subseteq V \times V \times \mathcal{L}$ a finite set of directed labeled edges, such that (V, E) forms a tree. We write $w-\ell \rightarrow w'$ for an edge labeled ℓ from w to w' . We define the ℓ -daughters $\ell(w)$ of $w \in V$ as follows:

$$\ell(w) = \{w' \in V \mid w-\ell \rightarrow w' \in E\}$$

We write $\widehat{\mathcal{L}}$ for the set of valency specifications $\widehat{\ell}$ defined by the following abstract syntax:

$$\widehat{\ell} ::= \ell \mid \ell? \mid \ell* \quad (\ell \in \mathcal{L})$$

A valency is a subset of $\widehat{\mathcal{L}}$. The tree (V, E) satisfies the valency assignment $\text{valency} : V \rightarrow 2^{\widehat{\mathcal{L}}}$ if for all $w \in V$ and all $\ell \in \mathcal{L}$:

$$\begin{aligned} \ell \in \text{valency}(w) &\Rightarrow |\ell(w)| = 1 \\ \ell? \in \text{valency}(w) &\Rightarrow |\ell(w)| \leq 1 \\ \ell* \in \text{valency}(w) &\Rightarrow |\ell(w)| \geq 0 \\ \text{otherwise} &\Rightarrow |\ell(w)| = 0 \end{aligned}$$

4.1 ID Trees

An ID tree $(V, E_{\text{ID}}, \text{lex}, \text{cat}, \text{valency}_{\text{ID}})$ consists of a tree (V, E_{ID}) with $E_{\text{ID}} \subseteq V \times V \times \mathcal{R}$, where the set $\mathcal{R}$ of edge labels (Figure 1) represents syntactic roles such as **subject** or **vinf** (bare infinitive argument). $\text{lex} : V \rightarrow \text{Lexicon}$ assigns a lexical entry to each node. An illustrative *Lexicon* is displayed in Figure 1 where the 2 features **cats** and $\text{valency}_{\text{ID}}$ of concern to ID trees are grouped under table heading “Syntax”. Finally, **cat** and $\text{valency}_{\text{ID}}$ assign a category and an $\widehat{\mathcal{R}}$ valency to each node $w \in V$ and must satisfy:

$$\begin{aligned} \text{cat}(w) &\in \text{lex}(w).\text{cats} \\ \text{valency}_{\text{ID}}(w) &= \text{lex}(w).\text{valency}_{\text{ID}} \end{aligned}$$

(V, E_{ID}) must satisfy the $\text{valency}_{\text{ID}}$ assignment as described earlier. For example the lexical entry for *versucht* specifies (Figure 1):

$$\text{valency}_{\text{ID}}(\text{versucht}) = \{\text{subject}, \text{zuvinf}\}$$

Furthermore, (V, E_{ID}) must also satisfy the edge constraints stipulated by the grammar (see Figure 1). For example, for an edge $w\text{-det}\rightarrow w'$ to be licensed, w' must be assigned category *det* and both w and w' must be assigned the same agreement.¹

4.2 LP Trees

An LP tree $(V, E_{\text{LP}}, \text{lex}, \text{valency}_{\text{LP}}, \text{field}_{\text{ext}}, \text{field}_{\text{int}})$ consists of a tree (V, E_{LP}) with $E_{\text{LP}} \subseteq V \times V \times \mathcal{F}_{\text{ext}}$, where the set $\mathcal{F}_{\text{ext}}$ of edge labels represents topological fields (Bech, 1955): **df** the determiner field, **mf** the ‘Mittelfeld’, **vc**

the verbal complement field, **xf** the extraposition field. Features of lexical entries relevant to LP trees are grouped under table heading “Topology” in Figure 1. $\text{valency}_{\text{LP}}$ assigns a $\widehat{\mathcal{F}}_{\text{ext}}$ valency to each node and is subject to the lexicalized constraint:

$$\text{valency}_{\text{LP}}(w) = \text{lex}(w).\text{valency}_{\text{LP}}$$

(V, E_{LP}) must satisfy the $\text{valency}_{\text{LP}}$ assignment as described earlier. For example, the lexical entry for *zu lieben₂* specifies:

$$\text{valency}_{\text{LP}}(\text{zu lieben}_2) = \{\text{mf*}, \text{xf?}\}$$

which permits 0 or more **mf** edges and at most one **xf** edge; we say that it offers fields **mf** and **xf**. Unlike the ID tree, the LP tree must be projective.

The grammar stipulates a total order on $\mathcal{F}_{\text{ext}}$, thus inducing a partial linear precedence on each node’s daughters. This order is partial because all daughters in the same field may be freely permuted: our account of *scrambling* rests on free permutations within the **mf** field. In order to obtain a linearization of the LP tree, it is necessary to specify the position of a node with respect to its daughters. For this reason each node is assigned an internal field in $\mathcal{F}_{\text{int}}$. The set $\mathcal{F}_{\text{ext}} \cup \mathcal{F}_{\text{int}}$ is totally ordered:

$$\text{d} \prec \text{df} \prec \text{n} \prec \text{mf} \prec \text{vc} \prec \text{v} \prec \text{xf}$$

In what (external) field a node may land and what internal field it may be assigned is determined by assignments $\text{field}_{\text{ext}} : V \rightarrow \mathcal{F}_{\text{ext}}$ and $\text{field}_{\text{int}} : V \rightarrow \mathcal{F}_{\text{int}}$ which are subject to the lexicalized constraints:

$$\begin{aligned} \text{field}_{\text{ext}}(w) &\in \text{lex}(w).\text{field}_{\text{ext}} \\ \text{field}_{\text{int}}(w) &\in \text{lex}(w).\text{field}_{\text{int}} \end{aligned}$$

For example, *zu lieben₁* may only land in field **vc** (canonical position), and *zu lieben₂* only in **xf** (extraposed position). The LP tree must satisfy:

$$w\text{-}\ell\rightarrow w' \in E_{\text{LP}} \Rightarrow \ell = \text{field}_{\text{ext}}(w')$$

Thus, whether an edge $w\text{-}\ell\rightarrow w'$ is licensed depends both on $\text{valency}_{\text{LP}}(w)$ and on $\text{field}_{\text{ext}}(w')$. In other words: w must offer field ℓ and w' must accept it.

For an edge $w\text{-}\ell\rightarrow w'$ in the ID tree, we say that w is the head of w' . For a similar edge in the LP

¹Issues of agreement will not be further considered in this paper.

Grammar Symbols					
$\mathcal{C} = \{det, n, vfin, vinf, vpast, zuvinf\}$			(Categories)		
$\mathcal{R} = \{det, subject, object, vinf, vpast, zuvinf\}$			(Syntactic Roles)		
$\mathcal{F}_{ext} = \{df, mf, vc, xf\}$			(External Topological Fields)		
$\mathcal{F}_{int} = \{d, n, v\}$			(Internal Topological Fields)		
$d \prec df \prec n \prec mf \prec vc \prec v \prec xf$			(Topological Ordering)		
Edge Constraints					
$w \text{---} det \text{---} w'$	$\Rightarrow$	$cat(w') = det$	$\wedge agr(w) = agr(w')$		
$w \text{---} subject \text{---} w'$	$\Rightarrow$	$cat(w') = n$	$\wedge agr(w) = agr(w') \in NOM$		
$w \text{---} object \text{---} w'$	$\Rightarrow$	$cat(w') = n$	$\wedge agr(w') \in ACC$		
$w \text{---} vinf \text{---} w'$	$\Rightarrow$	$cat(w') = vinf$			
$w \text{---} vpast \text{---} w'$	$\Rightarrow$	$cat(w') = vpast$			
$w \text{---} zuvinf \text{---} w'$	$\Rightarrow$	$cat(w') = zuvinf$			
Lexicon					
Word	Syntax		Topology		
	cats	valency _{ID}	field _{int}	field _{ext}	valency _{LP}
<i>einen</i>	{ <i>det</i> }	{ }	{ <i>d</i> }	{ <i>df</i> }	{ }
<i>Mann</i>	{ <i>n</i> }	{ <i>det</i> }	{ <i>n</i> }	{ <i>mf</i> }	{ <i>df</i> ?}
<i>Maria</i>	{ <i>n</i> }	{ }	{ <i>n</i> }	{ <i>mf</i> }	{ }
<i>lieben</i>	{ <i>vinf</i> }	{ <i>object</i> ?}	{ <i>v</i> }	{ <i>vc</i> }	{ }
<i>geliebt</i>	{ <i>vpast</i> }	{ <i>object</i> ?}	{ <i>v</i> }	{ <i>vc</i> }	{ }
<i>können</i> ₁	{ <i>vinf</i> }	{ <i>vinf</i> }	{ <i>v</i> }	{ <i>vc</i> }	{ <i>vc</i> ?}
<i>können</i> ₂	{ <i>vinf</i> , <i>vpast</i> }	{ <i>vinf</i> }	{ <i>v</i> }	{ <i>xf</i> }	{ <i>mf</i> *, <i>vc</i> ?, <i>xf</i> ?}
<i>wird</i>	{ <i>vfin</i> }	{ <i>subject</i> , <i>vinf</i> }	{ <i>v</i> }	{ <i>vc</i> }	{ <i>mf</i> *, <i>vc</i> ?, <i>xf</i> ?}
<i>haben</i>	{ <i>vinf</i> }	{ <i>vpast</i> }	{ <i>v</i> }	{ <i>xf</i> }	{ <i>mf</i> *, <i>vc</i> ?, <i>xf</i> ?}
<i>hat</i>	{ <i>vinf</i> }	{ <i>subject</i> , <i>vpast</i> }	{ <i>v</i> }	{ <i>vc</i> }	{ <i>mf</i> *, <i>vc</i> ?, <i>xf</i> ?}
<i>zu lieben</i> ₁	{ <i>zuvinf</i> }	{ <i>object</i> ?}	{ <i>v</i> }	{ <i>vc</i> }	{ }
<i>zu lieben</i> ₂	{ <i>zuvinf</i> }	{ <i>object</i> ?}	{ <i>v</i> }	{ <i>xf</i> }	{ <i>mf</i> *, <i>xf</i> ?}
<i>versucht</i>	{ <i>vfin</i> }	{ <i>subject</i> , <i>zuvinf</i> }	{ <i>v</i> }	{ <i>vc</i> }	{ <i>mf</i> *, <i>vc</i> ?, <i>xf</i> ?}

Figure 1: Grammar Fragment

tree, we say that w is the host of w' or that w' lands on w . The shape of the LP tree is a flat-tened version of the ID tree which is obtained by allowing nodes to *climb up* subject to the following principles:

Principle 1 *a node must land on a transitive head²*

Principle 2 *it may not climb through a barrier*

We will not elaborate the notion of barrier which is beyond the scope of this article, but, for example, a noun will prevent a determiner from climbing through it, and finite verbs are typically general barriers.

²This is Bröcker's terminology and means a node in the transitive closure of the head relation.

Principle 3 *a node must land on, or climb higher than, its head*

Subject to these principles, a node w' may climb up to any host w which offers a field licensed by $field_{ext}(w')$.

Definition. An ID/LP analysis is a tuple $(V, E_{ID}, E_{LP}, lex, cat, valency_{ID}, valency_{LP}, field_{ext}, field_{int})$ such that $(V, E_{ID}, lex, cat, valency_{ID})$ is an ID tree and $(V, E_{LP}, lex, valency_{LP}, field_{ext}, field_{int})$ is an LP tree and all principles are satisfied.

Our approach has points of similarity with (Bröcker, 1999) but eschews modal logic in favor of a simpler and arguably more perspicuous constraint-based formulation. It is also related

to the lifting rules of (Kahane et al., 1998), but where they choose to stipulate rules that license liftings, we opt instead for placing constraints on otherwise unrestricted climbing.

5 German Verbal Phenomena

We now illustrate our theory by applying it to the treatment of word order phenomena in the verbal complex of German verb final sentences. We assume the grammar and lexicon shown in Figure 1. These are intended purely for didactic purposes and we extend for them no claim of linguistic adequacy.

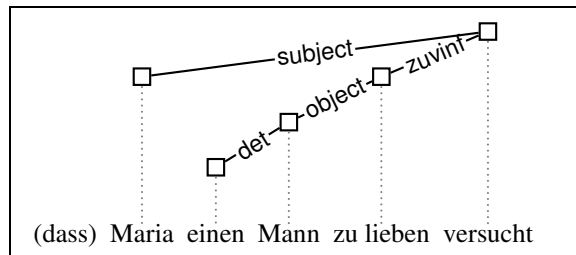
5.1 VP Extraposition

Control verbs like *versuchen* or *versprechen* allow their *zu*-infinitival complement to be optionally extraposed. This phenomenon is also known as optional coherence.

(6) (dass) Maria einen Mann zu lieben versucht

(7) (dass) Maria versucht, einen Mann zu lieben

Both examples share the following ID tree:



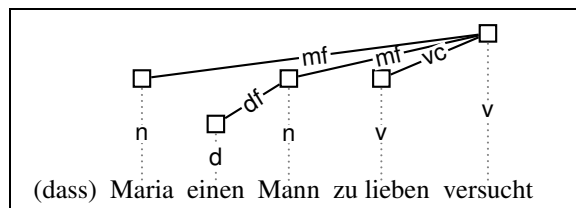
Optional extraposition is handled by having two lexical entries for *zu lieben*. One requires it to land in canonical position:

$$\text{field}_{\text{ext}}(\text{zu lieben}_1) = \{\text{vc}\}$$

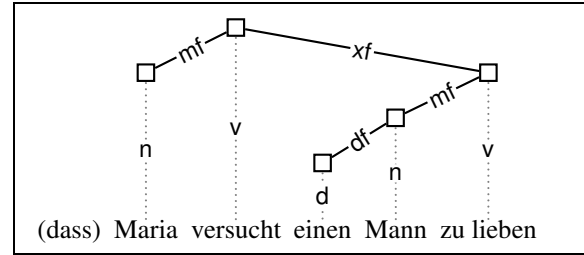
the other requires it to be extraposed:

$$\text{field}_{\text{ext}}(\text{zu lieben}_2) = \{\text{xf}\}$$

In the canonical case, *zu lieben*₁ does not offer field *mf* and *einen Mann* must climb to the finite verb:



In the extraposed case, *zu lieben*₂ itself offers field *mf*:

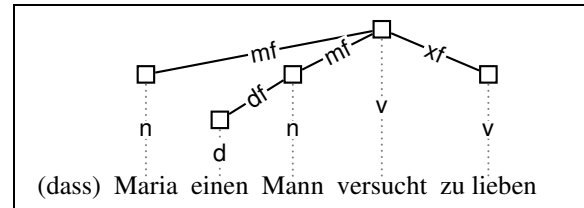


5.2 Partial VP Extraposition

In example (8), the *zu*-infinitive *zu lieben* is extraposed to the right of its governing verb *versucht*, but its nominal complement *einen Mann* remains in the Mittelfeld:

(8) (dass) Maria einen Mann versucht, zu lieben

In our account, *Mann* is restricted to land in an *mf* field which both extraposed *zu lieben*₂ and finite verb *versucht* offer. In example (8) the nominal complement simply climbed up to the finite verb:



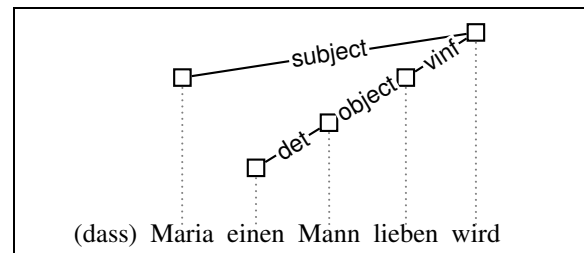
5.3 Obligatory Head-final Placement

Verb clusters are typically head-final in German: non-finite verbs precede their verbal heads.

(9) (dass) Maria einen Mann lieben wird
(that) Maria_{nom} a man_{acc} love will

(10)* (dass) Maria einen Mann wird lieben

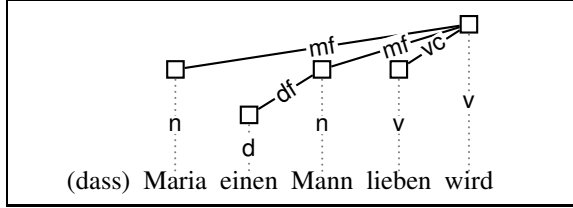
The ID tree for (9) is:



The lexical entry for the bare infinitive *lieben* requires it to land in a *vc* field:

$$\text{field}_{\text{ext}}(\text{lieben}) = \{\text{vc}\}$$

therefore only the following LP tree is licensed:³



where $mf \prec vc \prec v$, and subject and object, both in field mf, remain mutually unordered. Thus we correctly license (9) and reject (10).

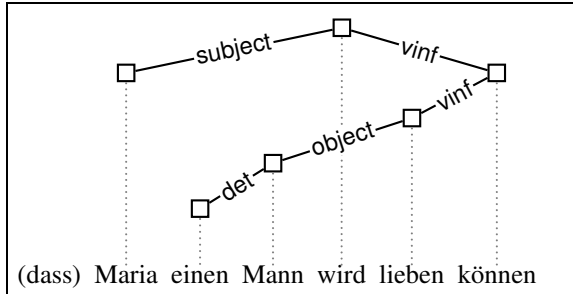
5.4 Optional Auxiliary Flip

In an auxiliary flip construction (Hinrichs and Nakazawa, 1994), the verbal complement of an auxiliary verb, such as *haben* or *werden*, follows rather than precedes its head. Only a certain class of bare infinitive verbs can land in extraposed position. As we illustrated above, main verbs do not belong to this class; however, modals such as *können* do, and may land in either canonical (11) or in extraposed (12) position. This behavior is called ‘optional auxiliary flip’.

(11) (dass) Maria einen Mann lieben können wird
(that) Maria a man love can will
(that) Maria will be able to love a man

(12) (dass) Maria einen Mann wird lieben können

Both examples share the following ID tree:



Our grammar fragment describes optional auxiliary flip constructions in two steps:

- *wird* offers both vc and xf fields:

$$\text{valency}_{\text{ID}}(\text{wird}) = \{mf*, vc?, xf?\}$$

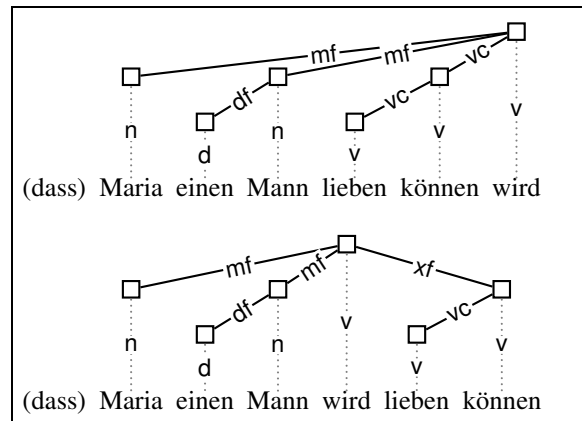
- *können* has two lexical entries, one canonical and one extraposed:

$$\text{field}_{\text{ext}}(\text{können}_1) = \{vc\}$$

$$\text{field}_{\text{ext}}(\text{können}_2) = \{xf\}$$

³It is important to notice that there is no spurious ambiguity concerning the topological placement of *Mann*: *lieben* in canonical position does not offer field mf; therefore *Mann* must climb to the finite verb.

Thus we correctly account for examples (11) and (12) with the following LP trees:



The astute reader will have noticed that other LP trees are licensed for the earlier ID tree: they are considered in the section below.

5.5 V-Projection Raising

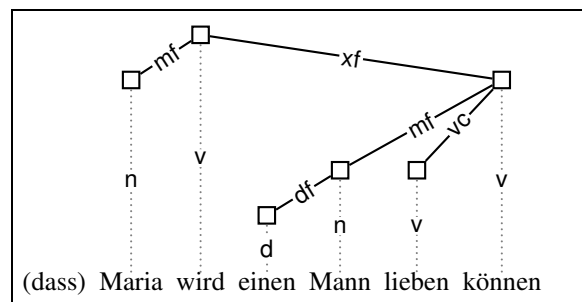
This phenomenon related to auxiliary flip describes the case where non-verbal material is interspersed in the verb cluster:

(13) (dass) Maria wird einen Mann lieben können

(14)* (dass) Maria lieben einen Mann können wird

(15)* (dass) Maria lieben können einen Mann wird

The ID tree remains as before. The NP *einen Mann* must land in a mf field. *lieben* is in canonical position and thus does not offer mf, but both extraposed *können*₂ and finite verb *wird* do. Whereas in (12), the NP climbed up to *wird*, in (13) it climbs only up to *können*.



(14) is ruled out because *können* must be in the vc of *wird*, therefore *lieben* must be in the vc of *können*, and *einen Mann* must be in the mf of *wird*. Therefore, *einen Mann* must precede both *lieben* and *können*. Similarly for (15).

5.6 Intermediate Placement

The *Zwischenstellung* construction describes cases where the auxiliary has been flipped but its verbal argument remains in the Mittelfeld. These are the remaining linearizations predicted by our theory for the running example started above:

(16) (dass) Maria einen Mann lieben wird können

(17) (dass) einen Mann Maria lieben wird können

where *lieben* has climbed up to the finite verb.

5.7 Obligatory Auxiliary Flip

Substitute infinitives (Ersatzinfinitiv) are further examples of extraposed verbal forms. A substitute infinitive exhibits bare infinitival inflection, yet acts as a complement of the perfectizer *haben*, which syntactically requires a past participle. Only modals, Acl-verbs such as *sehen* and *lassen*, and the verb *helfen* can appear in substitute infinitival inflection.

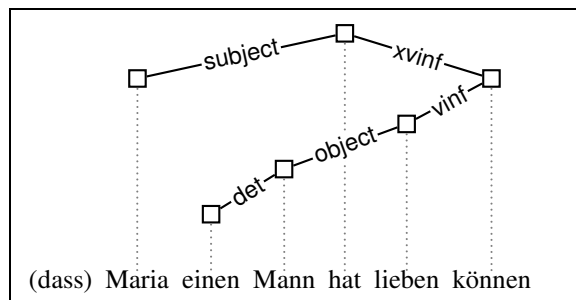
A substitute infinitive cannot land in canonical position; it must be extraposed: an auxiliary flip involving a substitute infinitive is called an ‘obligatory auxiliary flip’.

(18) (dass) Maria einen Mann hat lieben können
(that) Maria a man has love can
(that) Maria was able to love a man

(19) (dass) Maria hat einen Mann lieben können

(20)*(dass) Maria einen Mann lieben können hat

These examples share the ID tree:



hat subcategorizes for a verb in past participle inflection because:

$$\text{valency}_{\text{ID}}(\text{hat}) = \{\text{subject}, \text{vpast}\}$$

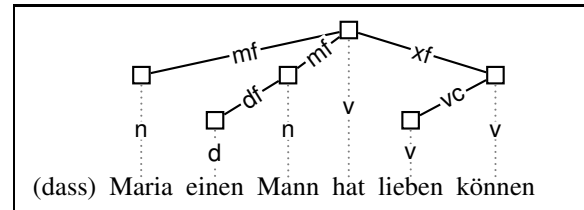
and the edge constraint for $w\text{-vpast} \rightarrow w'$ requires:

$$\text{cat}(w') = \text{vpast}$$

This is satisfied by *können*₂ which insists on being extraposed, thus ruling (20) out:

$$\text{field}_{\text{ext}}(\text{können}_2) = \{\text{xf}\}$$

Example (18) has LP tree:



In (18) *einen Mann* climbs up to *hat*, while in (19) it only climbs up to *können*.

5.8 Double Auxiliary Flip

Double auxiliary flip constructions occur when an auxiliary is an argument of another auxiliary. Each extraposed verb form offers both *vc* and *mf*: thus there are more opportunities for verbal and nominal arguments to climb to.

(21) (dass) Maria wird haben einen Mann lieben können
(that) Maria will have been able to love a man

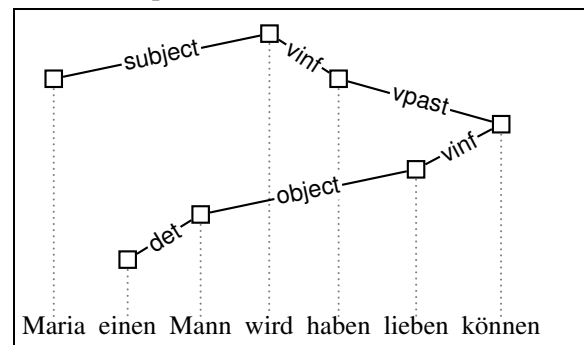
(22) (dass) Maria einen Mann wird haben lieben können

(23) (dass) Maria wird einen Mann lieben haben können

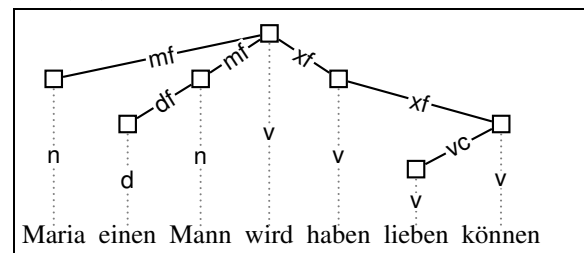
(24) (dass) Maria einen Mann wird lieben haben können

(25) (dass) Maria einen Mann lieben wird haben können

These examples have ID tree:



and (22) obtains LP tree:



5.9 Obligatory Coherence

Certain verbs like *scheint* require their argument to appear in canonical (or coherent) position.

(26) (dass) Maria einen Mann zu lieben scheint
(that) Maria a man to love seems
(that) Maria seems to love a man

(27)*(dass) Maria einen Mann scheint, zu lieben

Obligatory coherence may be enforced with the following constraint principle: if w is an obligatory coherence verb and w' is its verbal argument, then w' must land in w 's VC field. Like barriers, the expression of this principle in our grammatical formalism falls outside the scope of the present article and remains the subject of active research.⁴

6 Conclusions

In this article, we described a treatment of linear precedence that extends the constraint-based framework for dependency grammar proposed by Duchier (1999). We distinguished two orthogonal, yet mutually constraining tree structures: unordered, non-projective ID trees which capture purely syntactic dependencies, and ordered, projective LP trees which capture topological dependencies. Our theory is formulated in terms of (a) lexicalized constraints and (b) principles which govern 'climbing' conditions.

We illustrated this theory with an application to the treatment of word order phenomena in the verbal complex of German verb final sentences, and demonstrated that these traditionally challenging phenomena emerge naturally from our simple and elegant account.

Although we provided here an account specific to German, our framework intentionally permits the definition of arbitrary language-specific topologies. Whether this proves linguistically adequate in practice needs to be substantiated in future research.

Characteristic of our approach is that the formal presentation defines valid analyses as the solutions of a constraint satisfaction problem which is amenable to efficient processing through constraint propagation. A prototype was implemented in Mozart/Oz and supports a parsing

mode as well as a mode generating all licensed linearizations for a given input. It was used to prepare all examples in this article.

While the preliminary results presented here are encouraging and demonstrate the potential of our approach to linear precedence, much work remains to be done to extend its coverage and to arrive at a cohesive and comprehensive grammar formalism.

References

- Gunnar Bech. 1955. *Studien über das deutsche Verbum infinitum*. 2nd unrevised edition published 1983 by Max Niemeyer Verlag, Tübingen (Linguistische Arbeiten 139).
- Norbert Bröker. 1999. *Eine Dependenzgrammatik zur Kopplung heterogener Wissensquellen*. Linguistische Arbeiten 405. Max Niemeyer Verlag, Tübingen/FRG.
- Denys Duchier. 1999. Axiomatizing dependency parsing using set constraints. In *Sixth Meeting on the Mathematics of Language*, Orlando/FL, July.
- Erhard Hinrichs and Tsuneko Nakazawa. 1994. Linearizing AUXs in German verbal complexes. In Nerbonne et al. (Nerbonne et al., 1994), pages 11–37.
- Sylvain Kahane, Alexis Nasr, and Owen Rambow. 1998. Pseudo-projectivity: a polynomially parsable non-projective dependency grammar. In *Proc. ACL/COLING'98*, pages 646–52, Montréal.
- Andreas Kathol. 2000. *Linear Syntax*. Oxford University Press.
- Igor Mel'čuk. 1988. *Dependency Syntax: Theory and Practice*. The SUNY Press, Albany, N.Y.
- Stefan Müller. 1999. *Deutsche Syntax deklarativ. Head-Driven Phrase Structure Grammar für das Deutsche*. Linguistische Arbeiten 394. Max Niemeyer Verlag, Tübingen/FRG.
- John Nerbonne, Klaus Netter, and Carl Pollard, editors. 1994. *German in Head-Driven Phrase Structure Grammar*. CSLI, Stanford/CA.
- Mike Reape. 1994. Domain union and word order variation in German. In Nerbonne et al. (Nerbonne et al., 1994), pages 151–197.
- Hans Uszkoreit. 1987. *Word Order and Constituent Structure in German*. CSLI, Stanford/CA.

⁴we also thank an anonymous reviewer for pointing out that our grammar fragment does not permit intraposition

Configuration Of Labeled Trees Under Lexicalized Constraints And Principles

Denys Duchier

*Programming Systems Lab, Universität des Saarlandes, Saarbrücken,
Email: `duchier@ps.uni-sb.de`*

ABSTRACT: *Trees with labeled edges have widespread applicability, for example for the representation of dependency syntax trees. Given a fixed number of nodes and constraints on how edges may be drawn between them, the task of finding solution trees is known as a configuration problem. In this paper, we formalize the configuration problem of labeled trees and argue that it can be regarded as a constraint satisfaction problem which can be solved directly and efficiently by constraint propagation. In particular, we derive and prove correct a formulation of dependency parsing as a constraint satisfaction problem.*

Our approach, based on constraints on finite sets and a new family of ‘selection’ constraints, is especially well-suited for the compact representation and efficient processing of ambiguity. We address various issues of interest to the computational linguist such as lexical ambiguity, structural ambiguity, valency constraints, grammatical principles, and linear precedence. Finally we turn to the challenge of efficient processing and characterize the services expected of a constraint programming system: we define a formal constraint language and specify its operational semantics with inference rules of propagation and distribution.

This framework generalizes our presentation of immediate syntactic dependence for dependency parsing [4] and extends naturally to our corresponding treatment of linear precedence [6] based on a notion of topological rather than syntactic dependencies.

KEYWORDS: *labeled trees, configuration, constraint satisfaction, constraint propagation, set constraints, parsing, dependency grammar*

1 Introduction

This article presents a formalization of finite trees with labeled edges: well-formedness is characterized by a small number of equations and trees correspond precisely to the solutions of these equations. The advantage of our approach is that all our equations can be interpreted as constraints and can be solved directly using constraint programming technology. The constraint-based approach is especially well-suited for the compact representation and efficient processing of ambiguity, and constraint propagation is very effective in pruning the search space.

We begin with a characterization of the legal trees which can be assembled from a finite set V of nodes and a finite set $\mathcal{L}$ of edge labels. Arranging the nodes V into a tree requires choosing, from the set of all possible labeled edges, a subset such that the resulting graph is a tree. This task may be regarded as a configuration problem and can be formulated as constraint satisfaction problem well-suited to a constraint-based approach. We then entertain various refinements, especially relevant to linguistics, where admissibility is further restricted either by general principles or through lexicalized constraints. Finally, we address the issue of efficient processing: it is achieved by effective model elimination through constraint propagation. For this reason, we give precise operational semantics to all our constraints in the form of inference rules. The search for solutions of a constraint satisfaction problem (CSP) is defined formally as the derivation of consistent saturations under these inference rules.

The objects described by modern linguistic theories such as HPSG [21, 22] or LFG [15] are typed features structures (TFS) and the theories themselves consist primarily in the formulation of general structural principles of well-formedness that determine which of these objects are licensed. These theories are declarative and constraint-based and remain uncommitted to any particular processing method. While TFS are appealing and easily integrated in unification-based computational frameworks, they result in grammatical formalisms which are hard to process efficiently. The processing challenge is further exacerbated when attempting to account for languages, such as German, where free word order and discontinuous constituents violate the assumptions of linearity and adjacency underlying many parsing techniques.

In [4], we described an alternative approach based on Dependency Grammar (DG). An advantage of DG is that it allows syntax trees with crossing branches, and thus does not fall prey to the difficulties we just mentioned plaguing grammatical formalisms that have traditionally assumed and required projective analyses. We showed how parsing could be formulated succinctly as a constraint satisfaction problem (CSP) solvable efficiently by constraint programming. One novelty of our approach was the central importance given to sets and constraints on set variables.

Our purpose in the present article is twofold. Firstly, we abstract the approach of [4] away from the details specific to dependency grammar. We tackle the more general problem of configuring trees with labeled edges and demonstrate how it can be formulated as a CSP. In this manner, our techniques gain wider scope, for example extending naturally to the treatment of linear precedence described in [6]. Secondly,

we give a formal account of constraint programming sufficient to solve the aforementioned CSPs: we define an abstract constraint language, completely specify its operational semantics by means of inference rules for propagation and distribution, and prove that solutions precisely correspond to the consistent saturations. It is our hope that, by providing formally precise blueprints of the constraint propagation mechanisms required, we may both promote an understanding and facilitate the adoption of our constraints in other systems, not necessarily based, as ours, on the concurrent constraint programming language Oz [19].

We begin the article, in Section 2, with an example of dependency analysis for German taken from [4]: it provides a concrete illustration that helps understand the abstractions which follow. Section 3 lays down the foundation in the form of a formalization of labeled trees. To a great extent, the conciseness of our formulation and the effectiveness of our treatment of ambiguity rely on ‘selection constraints’: first introduced in [4], they are presented intuitively in Section 4 and further extended to an entire family of aggregative selection constraints.

We then consider various refinements of interest to the computational linguist: in Section 5 we tackle the problem of lexical ambiguity; in Section 6 we further extend it to the treatment of lexicalized valency constraints; in Section 7 we describe how families of well-formedness principles can be expressed and integrated into our framework to further restrict admissibility of edges, e.g. for reasons of ‘agreement’; in Section 8 we introduce ‘disjunctive attributes’ for a treatment of ambiguity that helps improve lexical economy; then in Section 9 we propose an extension for partially ordered trees which permits the characterization of projective analyses and the formulation of linear precedence constraints.

Finally in Section 11 we address the issue of efficient processing and describe what we expect from the constraint programming services. We do not commit to any particular implementation technique, rather we describe constraint programming at a very abstract level as a formal system with deterministic inference rules of ‘propagation’ and non-deterministic rules of ‘distribution’. In this framework, the search for solutions is precisely the derivation of consistent inferential saturations. We believe that we offer the right level of abstraction to permit instantiation to a variety of constraint programming and constraint logic programming systems, and the right level of detail for the practical implementation of our constraints should they not already exist in the target system.

2 Informal Introduction to Dependency Parsing

In [4] we described a constraint-based approach for constructing dependency tree analyses of German sentences: it serves as the starting point for the more general and abstract formulation developed in the remainder of the present article. In this section, we begin with an example taken from that earlier work to provide the reader with a concrete illustration of what we propose to generalize.

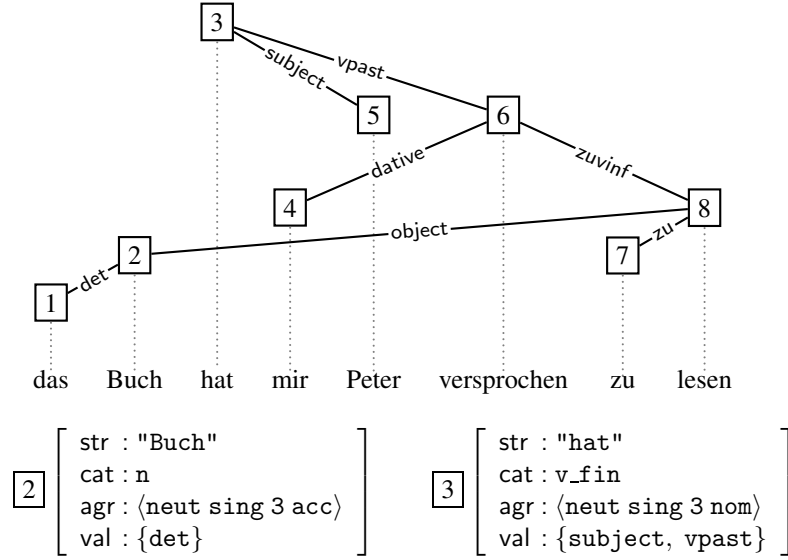


FIG. 1. Example of Dependency Tree

Consider the German sentence “*das Buch hat mir Peter versprochen zu lesen*”.¹ Figure 1 shows a corresponding dependency analysis and illustrates the fact that the relative freeness of word order often requires analyses with crossing branches. Characteristic of the dependency approach, there is exactly one node per word, displayed here as a box labeled with the linear position of the word in the sentence, and edges are labeled with syntactic roles such as ‘subject’ or ‘zuinf’. Furthermore each node is assigned a lexical entry from a lexicon: Figure 1 displays the lexical entries assigned by the parser to ‘Buch’ and ‘hat’ identified respectively by 2 and 3. Each entry stipulates phonology, category, agreement and valency. In Section 6, we propose an alternative approach to lexicalized valency constraints.

Dependency Grammar. We now briefly review the formal framework for dependency grammar proposed in [4]. For simplicity, we omit the treatment of modifiers² and of linear precedence.³ A dependency grammar is given by:

$$(Strs, Cats, Agrs, Roles, Lexicon, Rules)$$

where *Strs* is a finite set of strings, such as “Buch” or “hat”, notating the fully inflected forms of words, *Cats* is a finite set of categories such as n for noun, d for

¹Stripped of intonation and of modifiers, for the sake of simplicity, the example does not sound convincing to the German ear, but the following sentence due to Joachim Niehren exhibits the same structure and sounds perfectly natural: “*Genau diese Flasche Wein hat mir mein Kommissionär versprochen auf der Auktion zu ersteigern*”.

²We shall see in Section 6 that modifiers do not actually require separate treatment in our approach.

³We consider it again in Section 9 and more extensively in [6].

determiner, or *v_fin* for finite verb. Assuming the following sets for gender, number, person and case:

$$\begin{aligned} \text{Gender} &= \{\text{masc}, \text{fem}, \text{neut}\} & \text{Number} &= \{\text{sing}, \text{plur}\} \\ \text{Person} &= \{1, 2, 3\} & \text{Case} &= \{\text{nom}, \text{acc}, \text{dat}, \text{gen}\} \end{aligned}$$

we pose $\text{Agrs} = \text{Gender} \times \text{Number} \times \text{Person} \times \text{Case}$ for the set of agreement tuples such as $\langle \text{masc sing 3 nom} \rangle$. *Roles* is a finite set of grammatical functions such as *subject* or *zuvinf* for an infinitive with ‘zu’ which serve as edge labels in the dependency tree. A lexical entry is an attribute value matrix with signature:

$$\begin{bmatrix} \text{str} : \text{Strs} \\ \text{cat} : \text{Cats} \\ \text{agr} : \text{Agrs} \\ \text{val} : 2^{\text{Roles}} \end{bmatrix}$$

and specifies phonology, category, agreement and valency. The *Lexicon* is a finite set of lexical entries. We use functional notation and write $\text{cat}(w)$ for the category of the lexical entry assigned to node w . Finally, *Rules* is a family (Γ_ρ) of binary predicates, indexed by grammatical functions $\rho \in \text{Roles}$, expressing local grammatical principles: for an edge labeled ρ from mother w_1 to daughter w_2 to be admissible, the condition $\Gamma_\rho(w_1, w_2)$ must be satisfied; for example, for w_2 to serve as the determiner of noun w_1 it must (1) be a determiner, (2) agree with the noun, i.e.:

$$\Gamma_{\text{det}}(w_1, w_2) \quad \equiv \quad \text{cat}(w_2) = \text{d} \quad \wedge \quad \text{agr}(w_1) = \text{agr}(w_2)$$

Dependency Trees. We assume an infinite set *Nodes* of nodes and define a labeled directed edge as an element of $\text{Nodes} \times \text{Nodes} \times \text{Roles}$. A dependency tree (V, E, entry) consists of a finite set $V \subseteq \text{Nodes}$ of nodes, a finite set $E \subseteq V \times V \times \text{Roles}$ of labeled edges between these nodes, and a function $\text{entry} : V \rightarrow \text{Lexicon}$ assigning a lexical entry to each node. A dependency tree is admissible iff (1) it forms a tree in the classical graph theoretical sense, (2) every node has precisely the outgoing edges required by its valency, (3) for every edge $(w_1, w_2, \rho) \in E$, the condition $\Gamma_\rho(w_1, w_2)$ is satisfied.

Of course, the lexicon typically contains several lexical entries for each word which results in considerable lexical ambiguity. Our approach is very effective in handling both lexical and structural ambiguity and achieves this largely through the use of ‘selection constraints’ (Section 4). Figure 2 illustrates the processing achieved for our example by the parser of [4]. On the left is displayed the preferred reading, while on the right we see the complete search tree where a circle represents a choice point, a diamond leaf a solution, and a square leaf a failure. What is interesting is that there are no failures: constraint propagation is very effective; we need exactly 1 choice in order to enumerate the two possible analyses. Why are there 2 analyses? Simply because both ‘Buch’ and ‘Peter’ can indifferently be assigned either nominative or accusative

case, therefore either one can be subject while the other is object. Constraint propagation is sufficient to resolve all other ambiguities, both lexical (what lexical entry to choose for each word), and structural (what edges to draw between nodes).



FIG. 2. Parser Demo

3 Formalization of Valid Labeled Trees

We begin with a precise characterization of all well-formed trees which can be assembled from a fixed finite set of nodes V and edges with labels drawn from a finite set $\mathcal{L}$. These trees correspond precisely to the solutions of the constraint satisfaction problem on set variables articulated below.

Finite Labeled Graphs. We assume given an infinite set $\mathcal{V}$ of nodes and a finite set $\mathcal{L}$ of labels. A directed labeled edge is an element of $\mathcal{V} \times \mathcal{V} \times \mathcal{L}$. We write $\mathbf{G}(\mathcal{V}, \mathcal{L})$ for the set of finite graphs $G = (V, E)$ formed from a finite set of nodes $V \subseteq \mathcal{V}$ and a finite set of labeled edges $E \subseteq V \times V \times \mathcal{L}$. Note that, since we assume E to be a set, we only consider graphs without duplicate edges. We write $\mathbf{G}(V, \mathcal{L})$ for the graphs in $\mathbf{G}(\mathcal{V}, \mathcal{L})$ whose node set is V .

Finite Labeled Trees. A finite graph is a tree if and only if it satisfies the following “treeness conditions”:

- (a) Each node has at most one incoming edge
- (b) There is precisely one node with no incoming edge (one root)
- (c) There are no cycles

We write $\mathbf{T}(V, \mathcal{L})$ for the subset of $\mathbf{G}(V, \mathcal{L})$ satisfying conditions (a), (b) and (c). In the following, we are going to formulate a constraint $\Phi_{\text{ID}}(V, \mathcal{L})$ which a finite graph $G = (V, E) \in \mathbf{G}(V, \mathcal{L})$ must satisfy in order to be in $\mathbf{T}(V, \mathcal{L})$.

We write $w-\ell \rightarrow w'$ for a labeled edge (w, w', ℓ) and $w-\ell \rightarrow_G w'$ for $w-\ell \rightarrow w' \in E$. We define the successor relation $\rightarrow_G = \cup\{-\ell \rightarrow_G \mid \ell \in \mathcal{L}\}$ and write $\rightarrow_G^+$ and $\rightarrow_G^*$ for

its transitive and reflexive transitive closures. Given a relation $R \subseteq V \times V$, we define $R : V \rightarrow 2^V$ and the overloading $R : 2^V \rightarrow 2^V$ as follows:

$$R(x) = \{y \mid (x, y) \in R\} \quad R(S) = \cup\{R(x) \mid x \in S\}$$

In this manner, the edges of a graph G induce the following functions:

$$\begin{aligned} \ell_G &= -\ell \rightarrow_G & \text{down}_G &= \rightarrow_G^+ & \text{roots}_G &= V \setminus \rightarrow_G(V) \\ \text{daughters}_G &= \rightarrow_G & \text{eqdown}_G &= \rightarrow_G^* \end{aligned}$$

Given these definitions, we can reformulate the treeness conditions more formally. Condition (a) states that each node has at most one incoming edge, i.e. that 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, that:

$$\forall \ell, \ell' \in \mathcal{L} \quad \forall w, w' \in V \quad (\ell \neq \ell' \vee w \neq w') \Rightarrow \ell_G(w) \parallel \ell'_G(w') \quad (3.1)$$

where $\parallel$ represents disjointness. Condition (b) requires that there be one unique root:

$$|\text{roots}_G| = 1 \quad (3.2)$$

Finally, condition (c) forbids cycles, i.e. it must never be the case that $w \rightarrow_G^+ w$:

$$\forall w \in V \quad w \notin \text{down}_G(w) \quad (3.3)$$

Our formalization so far assumes that the edges are given. As such, it is appropriate for deciding whether a graph is a tree. On the other hand, it is poorly suited for parsing where the edges are unknown and the task is precisely to find possible edges licensed by the grammar. To overcome this difficulty, instead of using the edges as a starting point, we are going to use the functions defined above which they induce.

In a tree, these functions satisfy additional properties which we state below. $\ell_G(w)$ are the ℓ -daughters of w . By definition of $\rightarrow_G$ and condition (a) restated as (3.1), the daughters of w satisfy the equation:

$$\text{daughters}_G(w) = \uplus\{\ell_G(w) \mid \ell \in \mathcal{L}\} \quad (3.4)$$

where $\uplus$ denotes disjoint union. By definition of roots_G and condition (a), a node is either a root or is the daughter of precisely one node:

$$V = \text{roots}_G \uplus \uplus\{\text{daughters}_G(w) \mid w \in V\} \quad (3.5)$$

By definition of transitive closure: $\rightarrow_G^+ = \rightarrow_G \circ \rightarrow_G^*$. In other words, the nodes strictly below w are those equal to or strictly below its daughters:

$$\text{down}_G(w) = \cup\{\text{eqdown}_G(w') \mid w' \in \text{daughters}_G(w)\} \quad (3.6)$$

$$\begin{aligned}
\Phi_{\text{ID}}(V, \mathcal{L}) \equiv & \\
& V = \text{roots} \uplus \uplus \{\text{daughters}(w) \mid w \in V\} \\
\wedge & \quad |\text{roots}| = 1 \\
\wedge & \quad \forall w \in V \\
& \quad \text{eqdown}(w) = \{w\} \uplus \text{down}(w) \\
\wedge & \quad \text{down}(w) = \cup \{\text{eqdown}(w') \mid w' \in \text{daughters}(w)\} \\
\wedge & \quad \text{daughters}(w) = \uplus \{\ell(w) \mid \ell \in \mathcal{L}\}
\end{aligned}$$

FIG. 3. well-formedness condition of labeled trees

By definition of reflexive transitive closure: $\rightarrow_G^*(w) = \{w\} \cup \rightarrow_G^+(w)$. Additionally the acyclicity condition (c) requires that w does not occur in $\rightarrow_G^+(w)$. Therefore:

$$\text{eqdown}_G(w) = \{w\} \uplus \text{down}_G(w) \quad (3.7)$$

These properties lead us to formulate a constraint $\Phi_{\text{ID}}(V, \mathcal{L})$ (see Figure 3) in terms of variable roots of type 2^V and functional variables daughters , down , eqdown and ℓ (for all $\ell \in \mathcal{L}$) of type $V \rightarrow 2^V$. A solution of $\Phi_{\text{ID}}(V, \mathcal{L})$ is an assignment to these variables such that $\Phi_{\text{ID}}(V, \mathcal{L})$ is satisfied; in other words, posing $\mathcal{L} = \{\ell_1, \dots, \ell_n\}$ and writing F for the type $V \rightarrow 2^V$, a solution of $\Phi_{\text{ID}}(V, \mathcal{L})$ is a tuple:

$$(\text{roots}, \text{daughters}, \text{down}, \text{eqdown}, \ell_1, \dots, \ell_n) : 2^V \times F \times F \times F \times \dots \times F$$

that satisfies $\Phi_{\text{ID}}(V, \mathcal{L})$. We write $\text{Sols}(\Phi_{\text{ID}}(V, \mathcal{L}))$ for the set of solutions of $\Phi_{\text{ID}}(V, \mathcal{L})$. Every solution σ of $\Phi_{\text{ID}}(V, \mathcal{L})$ defines a graph $\llbracket \sigma \rrbracket_{\text{ID}} = (V, E)$ where:

$$E = \{w - \ell \rightarrow w' \mid w \in V, \ell \in \mathcal{L}, w' \in \ell(w)\}$$

Overloading the notation, for each $G \in \mathbf{G}(V, \mathcal{L})$ we also define:

$$\llbracket G \rrbracket_{\text{ID}} = (\text{roots}_G, \text{daughters}_G, \text{down}_G, \text{eqdown}_G, \ell_{1G}, \dots, \ell_{nG})$$

THEOREM 3.1

$\mathbf{T}(V, \mathcal{L})$ is in bijection with the solutions of $\Phi_{\text{ID}}(V, \mathcal{L})$. More precisely a graph G is a tree iff $\llbracket G \rrbracket_{\text{ID}}$ satisfies $\Phi_{\text{ID}}(V, \mathcal{L})$:

$$\forall G \in \mathbf{G}(V, \mathcal{L}) \quad G \in \mathbf{T}(V, \mathcal{L}) \equiv \llbracket G \rrbracket_{\text{ID}} \in \text{Sols}(\Phi_{\text{ID}}(V, \mathcal{L}))$$

and every solution σ of $\Phi_{\text{ID}}(V, \mathcal{L})$ defines a tree:

$$\forall \sigma \in \text{Sols}(\Phi_{\text{ID}}(V, \mathcal{L})) \quad \llbracket \sigma \rrbracket_{\text{ID}} \in \mathbf{T}(V, \mathcal{L})$$

The first claim follows from properties (3.4–3.7), while for the second one it is straightforward to establish $\sigma = \llbracket \llbracket \sigma \rrbracket_{\text{ID}} \rrbracket_{\text{ID}}$. $\Phi_{\text{ID}}(V, \mathcal{L})$ can be interpreted as a constraint satisfaction problem (CSP) and Theorem 3.1 establishes the correspondence between the solutions of this CSP and the formal objects of interest, namely the trees $\mathbf{T}(V, \mathcal{L})$. Throughout this article, we state similar theorems to validate the constraint-based approach.

4 Set Constraints And Selection Constraints

Equation (3.6) while mathematically elegant poses a processing challenge: when solving the CSP, (1) we don't know the elements of $\text{daughters}(w)$, (2) we don't know what set values eqdown takes at these elements, and yet we must compute their combined union. In this section, we introduce the constraint-based concepts that will allow us to achieve it simply and efficiently.

4.1 Set Constraints

Finite domain (FD) constraints have become a reasonably standard tool of the trade and are routinely used in computational linguistics applications. Set constraints, on the other hand, have remained largely unexploited even though they are available and well supported by modern constraint technology [12, 20, 18].

In our work, constraints on finite sets (FS) of integers have emerged as an especially elegant and computationally effective tool for such linguistics applications as parsing with a dependency grammar [4, 6] or solving dominance constraints [5, 8], for the treatment of discourse [7], parsing with tree descriptions [9], and underspecified representations of semantics [10].

We will elaborate on constraint programming at greater length in Section 11. For the moment, we shall simply say that the partial information about a FD variable I can be represented in the form $I \in D$ where D is a set of integers, and the partial information about a FS variable S can be expressed by a lower bound D_1 and an upper bound D_2 in the form $D_1 \subseteq S \subseteq D_2$. The role of constraint propagation is to improve this partial information. When $I \in \{k\}$ we say that I is 'determined' and write $I = k$. When $D \subseteq S \subseteq D$, we say that S is 'determined' and write $S = D$.

4.2 Selection Constraints

An essential contribution of [4] was the 'selection constraint' which permits the compact representation and effective processing of many forms of selectional ambiguity such as lexical ambiguity (i.e. the selection of a lexical entry from those available for a particular word in the lexicon). Consider a variable X which may be equated with one of n variables (V_i). We can explicitly represent this choice using an integer variable I , also called a finite domain (FD) variable, taking values in $\{1 \dots n\}$ and the selection constraint below:

$$X = \langle V_1, \dots, V_n \rangle [I]$$

where $\langle V_1, \dots, V_n \rangle$ represents the sequence of variables V_1 through V_n and the notation $\langle V_1, \dots, V_n \rangle [I]$ was chosen for its similarity to the subscripting notation of 'array lookup' in many programming languages, and indicates selection of the I th element out of the sequence. Thus the declarative semantics of the above constraint is simply $X = V_I$.

The origins of this powerful idea are to be found in CHIP's 'element' constraint [2] which related two finite domain variables I and K and a sequence $\langle j_1, \dots, j_n \rangle$ of integer values:

$$I = \langle j_1, \dots, j_n \rangle[K]$$

In [4], we extended it in two directions: first we allowed the sequence to consist of variables rather than constants; second, we supported both selection out of sequences of finite domain variables:

$$I = \langle J_1, \dots, J_n \rangle[K]$$

as well as out of sequences of finite set (FS) variables:

$$S = \langle S_1, \dots, S_n \rangle[K]$$

where S, S_i are FS variables denoting finite sets of integers.

4.3 Propagation And Constructive Disjunction

One advantage of the selection constraint is that it is able to implement simply and efficiently a form of constructive disjunction (lifting of information common to all alternatives not yet ruled out). Here is an example that illustrates the propagation which may be expected from the selection constraint:

$$S = \langle S_1, S_2, S_3 \rangle[K] \quad \{1, 3\} \subseteq S_1 \subseteq \{1, 2, 3\} \quad S_2 = \{2, 4\} \quad \{1\} \subseteq S_3 \subseteq \{1, 4\}$$

From the above, constraint propagation infers:

$$K \in \{1, 2, 3\} \quad S \subseteq \{1, 2, 3, 4\}$$

If we further assert $4 \notin S$, then S_2 becomes incompatible since it contains 4:

$$K \in \{1, 3\} \quad \{1\} \subseteq S \subseteq \{1, 2, 3\}$$

Note that 1 was inferred to be a necessary element of S since it is a known element of both alternatives S_1 and S_3 , one of which must eventually be chosen. If we now assert $2 \in S$, then S_3 becomes incompatible since it cannot contain 2:

$$K = 1 \quad S = S_1 = \{1, 2, 3\}$$

4.4 Dependent Disjunction

The fact that the selection constraints makes the choice explicit through a selector variable K permits dependent selections. Consider for example:

$$I = \langle J_1, \dots, J_n \rangle[K] \tag{4.1}$$

$$I' = \langle J'_1, \dots, J'_n \rangle[K] \tag{4.2}$$

the choice of which J_i to equate with I and which J'_i to equate with I' are mutually dependent since they must be effected by the same selector K . (4.1) and (4.2) can be viewed as contextual constraints sharing the same context variable K , or equivalently as realizing the following dependent (or named) disjunctions both labeled with name I [17, 3, 11, 13]:

$$\begin{aligned} (I = J_1 \vee \dots \vee I = J_n)_I \\ (I' = J'_1 \vee \dots \vee I' = J'_n)_I \end{aligned}$$

Notational variations on dependent disjunctions have been used to concisely express covariant assignment of values to different features in feature structures. The selection constraint provides the same notational convenience and declarative semantics, but additionally enjoys a computational reading with all the benefits that accrue from state-of-the-art constraint technology.

4.5 Selection Union Constraint

A novel contribution of the present article is the ‘selection union’ constraint:

$$S = \cup \langle S_1, \dots, S_n \rangle [S']$$

where the selector S' is now a set. Its declarative semantics are given by the following equation:

$$S = \cup \{S_k \mid k \in S'\}$$

Thus we empower an essential mathematical instrument with a computational reading based on very effective constraint propagation. Posing $V = \{w_1, \dots, w_n\}$ and identifying w_i with the integer i representing its position in the input sentence,⁴ equation (3.6) can now be rewritten:

$$\text{down}(w) = \cup \langle \text{eqdown}(w_1), \dots, \text{eqdown}(w_n) \rangle [\text{daughters}(w)]$$

A family of aggregative selection constraints. The selection union constraint selects a subset specified by S' of the elements of sequence $\langle S_1, \dots, S_n \rangle$ and combines them using union. Clearly, other modes of combination are possible which opens up a whole family of aggregative selection constraints. For example:

$$\begin{aligned} S &= \cap \langle S_1, \dots, S_n \rangle [S'] \\ S &= \oplus \langle S_1, \dots, S_n \rangle [S'] \\ I &= + \langle J_1, \dots, J_n \rangle [S'] \end{aligned}$$

In Section 11 we take the ‘selection union’ and ‘selection intersection’ constraints as primitives and fully specify their operational semantics by inference rules of propagation. The simpler selection constraints of Section 4.2 are defined in terms of the selection union constraint.

⁴This identification of a node with the linear position of the corresponding word in the input sentence will remain in effect for the remainder of the article.

5 Lexical Ambiguity

For the application to dependency parsing, each node is assigned a lexical entry from a lexicon. A lexical entry supplies a number of attributes in terms of which additional constraints may be formulated. In this section, we formalize these notions and demonstrate how the selection constraint elegantly addresses the issue of lexical ambiguity.

A lexicon $(\mathcal{E}, \mathcal{A})$ consists of a finite set $\mathcal{E}$ of objects called ‘lexical entries’ and a finite set $\mathcal{A}$ of functions called ‘attributes’, where each $\alpha \in \mathcal{A}$ is of type $\alpha : \mathcal{E} \rightarrow \mathbb{N}$ or $\alpha : \mathcal{E} \rightarrow 2^{\mathbb{N}}$. For each $e \in \mathcal{E}$ and $\alpha \in \mathcal{A}$, $\alpha(e)$ is the value of attribute α in lexical entry e . An attribute might specify such things as category, agreement, or valency.

Given a lexicon $(\mathcal{E}, \mathcal{A})$, a graph $(V, E) \in \mathbf{G}(V, \mathcal{L})$, and an assignment $\varepsilon : V \rightarrow \mathcal{E}$ of lexical entries to nodes, we call (V, E, ε) an *attributed graph*. We write $\mathbf{G}(V, \mathcal{L}, \mathcal{E}, \mathcal{A})$ for the set of attributed graphs and $\mathbf{T}(V, \mathcal{L}, \mathcal{E}, \mathcal{A})$ for the subset which are trees. For each $\alpha \in \mathcal{A}$ with type $\alpha : \mathcal{E} \rightarrow T$, where T is $\mathbb{N}$ or $2^{\mathbb{N}}$, we introduce the overloaded function $\alpha : V \rightarrow T$ called a *node attribute* and defined by $\alpha(w) = \alpha(\varepsilon(w))$ for all $w \in V$.

For parsing, the choice of ε is not free: in particular, only a subset of $\mathcal{E}$ is applicable to each word. This we model by means of a restriction function $\text{lex} : V \rightarrow 2^{\mathcal{E}}$. For example, if w corresponds to the word ‘versprochen’, then $\text{lex}(w)$ should be the set of lexical entries for ‘versprochen’. We say that (V, E, ε) is *lex-attributed* if:

$$\forall w \in V \ \varepsilon(w) \in \text{lex}(w) \quad (5.1)$$

and write $\mathbf{T}(V, \mathcal{L}, \mathcal{E}, \mathcal{A}, \text{lex})$ for the set of lex attributed trees (V, E, ε) over $(\mathcal{E}, \mathcal{A})$. During parsing, ε is not given but must be chosen. The degree of freedom in this choice (see 5.1) is called lexical ambiguity. In order to support efficient parsing, it should be possible to constrain a node attribute $\alpha(w)$ while leaving the choice $\varepsilon(w)$ of lexical entry underspecified.

Thus we are confronted with the problem of computing a function at a point which is only partially known. Supposing $\text{lex}(w) = \{e_1, \dots, e_n\}$, the idea is to introduce an FD variable $\text{entry}(w) \in \{1, \dots, n\}$ to represent the index of the selected entry and to obtain $\alpha(w)$ with the following selection constraint:

$$\alpha(w) = \langle \alpha(e_1), \dots, \alpha(e_n) \rangle [\text{entry}(w)] \quad (5.2)$$

In this equation, the sequence consists of elements homogeneously of type $\mathbb{N}$ or $2^{\mathbb{N}}$ and thus is in the domain of applicability of the selection constraint. For $w \in V$, we have one equation (5.2) for each $\alpha \in \mathcal{A}$, but they all share the same selector $\text{entry}(w)$. In this fashion, as explained in Section 4.4, all attribute selections for the same node are forced to be covariant. This is an additional source of effective propagation: if any constraint affects one selection, it affects them all.

It is this intuition which we now proceed to formalize. Without loss of generality, we revise the type of $\text{lex} : V \rightarrow \mathcal{E}^*$ to map each node to a sequence rather than a set

of lexical entries, and assume that each sequence contains no duplicates. We define:

$$e = \langle e_1, \dots, e_n \rangle[k] \equiv 1 \leq k \leq n \wedge e = e_k$$

We write $e \in \text{lex}(w)$ when e occurs in sequence $\text{lex}(w)$ and denote by $e@_{\text{lex}(w)}$ its position in this sequence, or more generally the smallest i such that $e = \text{lex}(w)[i]$. An assignment $\varepsilon : V \rightarrow \mathcal{E}$ is *lex-restricted* iff:

$$\forall w \in V \quad \varepsilon(w) \in \text{lex}(w) \quad (5.3)$$

ε induces functions $\text{entry}_\varepsilon : V \rightarrow \mathbb{N}$ and $\alpha_{i\varepsilon} : V \rightarrow T_i$ for each $\alpha_i : \mathcal{E} \rightarrow T_i \in \mathcal{A}$:

$$\begin{aligned} \text{entry}_\varepsilon(w) &= \varepsilon(w)@_{\text{lex}(w)} \\ \alpha_{i\varepsilon}(w) &= \alpha_i(\varepsilon(w)) \end{aligned}$$

Given a sequence $\langle e_1 \dots e_n \rangle \in \mathcal{E}^*$, we define:

$$\langle \alpha \mid \langle e_1 \dots e_n \rangle \rangle = \langle \alpha(e_1) \dots \alpha(e_n) \rangle$$

The functions induced by ε satisfy the following property:

$$\forall w \in V, \forall \alpha_i \in \mathcal{A} \quad \alpha_{i\varepsilon} = \langle \alpha_i \mid \text{lex}(w) \rangle[\text{entry}_\varepsilon(w)] \quad (5.4)$$

Posing $\mathcal{A} = \{\alpha_1 : \mathcal{E} \rightarrow T_1, \dots, \alpha_n : \mathcal{E} \rightarrow T_n\}$ (where T_i is $\mathbb{N}$ or $2^{\mathbb{N}}$), we now formulate a constraint $\Phi_{\text{LEX}}(V, \mathcal{E}, \mathcal{A}, \text{lex})$ (Figure 4) in terms of functional variables $\text{entry} : V \rightarrow \mathbb{N}$ and $\alpha_i : V \rightarrow T_i$ for $\alpha_i : \mathcal{E} \rightarrow T_i \in \mathcal{A}$. $\Phi_{\text{LEX}}(V, \mathcal{E}, \mathcal{A}, \text{lex})$ characterizes all lex-restricted assignments $\varepsilon : V \rightarrow \mathcal{E}$ over $(\mathcal{E}, \mathcal{A})$.

$$\begin{aligned} \Phi_{\text{LEX}}(V, \mathcal{E}, \mathcal{A}, \text{lex}) &\equiv \\ \forall w \in V \quad \bigwedge_{\alpha \in \mathcal{A}} \alpha(w) &= \langle \alpha \mid \text{lex}(w) \rangle[\text{entry}(w)] \end{aligned}$$

FIG. 4. Well-formedness condition for lex-restricted assignments

A solution of $\Phi_{\text{LEX}}(V, \mathcal{E}, \mathcal{A}, \text{lex})$ is a tuple:

$$(\text{entry}, \alpha_1, \dots, \alpha_n) : (V \rightarrow \mathbb{N}) \times (V \rightarrow T_1) \times \dots \times (V \rightarrow T_n)$$

that satisfies $\Phi_{\text{LEX}}(V, \mathcal{E}, \mathcal{A}, \text{lex})$. Every solution σ of $\Phi_{\text{LEX}}(V, \mathcal{E}, \mathcal{A}, \text{lex})$ defines a lex-restricted assignment $\llbracket \sigma \rrbracket_{\text{LEX}}$:

$$\llbracket \sigma \rrbracket_{\text{LEX}}(w) = \text{lex}(w)[\text{entry}(w)]$$

Furthermore, for each lex-restricted assignment ε , we also define:

$$\llbracket \varepsilon \rrbracket_{\text{LEX}} = (\text{entry}_\varepsilon, \alpha_{1\varepsilon}, \dots, \alpha_{n\varepsilon})$$

Writing $\mathbf{A}(V, \mathcal{E}, \mathcal{A}, \text{lex})$ for the set of lex-restricted assignments $\varepsilon : V \rightarrow \mathcal{E}$ over $(\mathcal{E}, \mathcal{A})$, we have:

THEOREM 5.1

$\mathbf{A}(V, \mathcal{E}, \mathcal{A}, \text{lex})$ is in bijection with $\text{Sols}(\Phi_{\text{LEX}}(V, \mathcal{E}, \mathcal{A}, \text{lex}))$.

$$\begin{aligned} \forall \sigma \in \text{Sols}(\Phi_{\text{LEX}}(V, \mathcal{E}, \mathcal{A}, \text{lex})) & \quad \llbracket \sigma \rrbracket_{\text{LEX}} \in \mathbf{A}(V, \mathcal{E}, \mathcal{A}, \text{lex}) \\ \forall \varepsilon \in \mathbf{A}(V, \mathcal{E}, \mathcal{A}, \text{lex}) & \quad \llbracket \varepsilon \rrbracket_{\text{LEX}} \in \text{Sols}(\Phi_{\text{LEX}}(V, \mathcal{E}, \mathcal{A}, \text{lex})) \end{aligned}$$

6 Structural Ambiguity and Valency Constraints

In the constraint formulation $\Phi_{\text{ID}}(V, \mathcal{L})$ of Figure 3, for each $w \in V$ and $\ell \in \mathcal{L}$, $\ell(w)$ is a set variable. During parsing, this variable is typically partially known. For example it might be constrained by the following bounds:

$$\{w_1, w_2\} \subseteq \ell(w) \subseteq \{w_1, w_2, w_3\}$$

indicating that w 's only possible outgoing edges labeled with ℓ are $w-\ell \rightarrow w_1$, $w-\ell \rightarrow w_2$ and $w-\ell \rightarrow w_3$, that the first two have been accepted, but that the case of the last one hasn't been decided yet. Thus, with set variables, we are able to represent an ambiguous tree structure. All possible edges are simultaneously represented, initially $\emptyset \subseteq \ell(w) \subseteq V$, and parsing is a process of disambiguation: candidate edges are either accepted or rejected. When every $\ell(w)$ is determined, the tree is fully disambiguated.

Disambiguation is to a large extent driven by grammatical constraints for subcategorization embodied in lexicalized valency constraints.

Lexicalized Valency Constraints. When parsing with a dependency grammar [4], we are not free to draw arbitrary edges between nodes. The outgoing edges of a node represent the complements and modifiers of the corresponding word. The nature and number of these edges are restricted by grammatical valencies specified in the lexicon. We formalize this as follows:

For every $\ell \in \mathcal{L}$, there is a corresponding attribute $|\cdot|_{\ell} \in \mathcal{A}$ with type $|\cdot|_{\ell} : \mathcal{E} \rightarrow 2^{\mathbb{N}}$. Thus each lexical entry e stipulates a set $|e|_{\ell}$ of licensed cardinalities for the ℓ -daughter set: the number of outgoing edges of w labeled with ℓ must be one in $|w|_{\ell}$:

$$|\ell(w)| \in |w|_{\ell} \tag{6.1}$$

$|w|_{\ell}$ depends on the choice of lexical entry and, following equation (5.2), is given by the selection constraint:

$$|w|_{\ell} = \langle |\cdot|_{\ell} \mid \text{lex}(w) \rangle [\text{entry}(w)] \tag{6.2}$$

In order to specify a required ℓ -argument, a lexical entry e need only fix $|e|_{\ell} = \{1\}$. For an optional argument: $|e|_{\ell} = \{0, 1\}$. For a modifier which may appear 0 or any number of times: $|e|_{\ell} = \{0, 1, \dots, \mu\}$ where μ is some arbitrarily large integer—for any particular sentence of length n , it is sufficient to choose $\mu = n - 1$ since for any one word, there are at most $n - 1$ arguments to be had. For an illegal argument: $|e|_{\ell} = \{0\}$.

We say that an attributed graph $G = (V, E, \varepsilon)$ fulfills its valencies iff:

$$\forall \ell \cdot |\ell| \in \mathcal{A}, \forall w \in V \quad |\ell_G(w)| \in |\varepsilon(w)|_\ell \quad (6.3)$$

and formulate a corresponding constraint $\Phi_{\text{VAL}}(V, \mathcal{L}, \mathcal{E}, \mathcal{A})$ in Figure 5.

$$\Phi_{\text{VAL}}(V, \mathcal{L}, \mathcal{E}, \mathcal{A}) \quad \equiv \quad \forall w \in V \quad \bigwedge_{\ell \in \mathcal{L}} |\ell(w)| \in |w|_\ell$$

FIG. 5. Well-formedness constraint of valency-fulfilling graphs

Writing $\mathbf{T}_{\text{VAL}}(V, \mathcal{L}, \mathcal{E}, \mathcal{A}, \text{lex})$ for the set of lex-attributed trees that fulfill their valencies, we have:

THEOREM 6.1

$\mathbf{T}_{\text{VAL}}(V, \mathcal{L}, \mathcal{E}, \mathcal{A}, \text{lex})$ is in bijection with the solutions of:

$$\Phi_{\text{ID}}(V, \mathcal{L}) \wedge \Phi_{\text{LEX}}(V, \mathcal{E}, \mathcal{A}, \text{lex}) \wedge \Phi_{\text{VAL}}(V, \mathcal{L}, \mathcal{E}, \mathcal{A})$$

Partial Functions. In passing, it may be useful to mention a related and generally useful transformation for the constraint-based treatment of partial functions. A partial function $f : A \rightarrow B$ is especially vexing for a constraint-based approach because a constraint about f is not meaningful everywhere, but only at points where f is defined. Frequently, the problem can be solved by replacing the partial function f by the total function $f' : A \rightarrow 2^B$ such that $f'(x) = \{f(x)\}$ if f is defined at x , and $f'(x) = \emptyset$ otherwise. For example, we might in this fashion conveniently model the notion of ‘mother’: every node has a unique mother except the root which has none.

7 Grammatical Principles Licensing Edges

While valency constraints restrict the number of edges for each grammatical function $\ell \in \mathcal{L}$, grammatical principles express additional local conditions for the admissibility of edges. For example, an object complement is required to be an accusative NP. In [4] (see Section 2) we proposed that a dependency grammar stipulate a family (Γ_ℓ) of binary predicates indexed by edge labels and such that $\Gamma_\ell(w, w')$ characterizes the grammatical admissibility of an edge $w-\ell \rightarrow w'$. In this section, we develop the framework for expressing these predicates and formalize the corresponding restrictions.

For our purposes, it suffices to consider the language whose abstract syntax is given in Figure 6 where x, y are variables ranging over nodes, i denotes an arbitrary integer, D an arbitrary finite set of integers, $\alpha(x)$ an attribute of node x , and $E \parallel E'$ expresses the disjointness of the sets denoted by E and E' .

For each $\ell \in \mathcal{L}$ there is a binary predicate Γ_ℓ of the form $\lambda x, y \cdot C$ (i.e. a P) which must be a closed abstraction of our language. We say that an attributed graph

$$\begin{aligned}
E &::= i \mid D \mid \alpha(x) \\
C &::= C \wedge C' \\
&\quad \mid E < E' \mid E \leq E' \\
&\quad \mid E = E' \mid E \neq E' \\
&\quad \mid E \in E' \mid E \notin E' \\
&\quad \mid E \subseteq E' \\
&\quad \mid E \parallel E' \\
P &::= \lambda x, y. C
\end{aligned}$$

FIG. 6. Constraint Language For Principles

$G = (V, E, \varepsilon)$ satisfies the grammatical principles (Γ_ℓ) if for all $w, w' \in V$ and $\ell \in \mathcal{L}$:

$$w' \in \ell(w) \Rightarrow G \models \Gamma_\ell(w, w') \quad (7.1)$$

where $G \models \Gamma_\ell(w, w')$ means that G satisfies $\Gamma_\ell(w, w')$ and is defined in the usual Tarskian way. For example, the grammatical principles validating an object complement or an adjective edge might be expressed as follows:

$$\begin{aligned}
\Gamma_{\text{object}} &\equiv \lambda x, y. \text{cat}(y) \in \text{NP} \wedge \text{agr}(y) \in \text{ACC} \\
\Gamma_{\text{adj}} &\equiv \lambda x, y. \text{cat}(y) = \mathbf{a} \wedge \text{agr}(x) = \text{agr}(y)
\end{aligned}$$

where NP represents the set of noun phrase categories (e.g. $\{\mathbf{n}, \mathbf{pro}\}$), ACC the set of all agreements with accusative case, and $\mathbf{a}$ the category of adjectives. (7.1) gives rise to a quadratic number of implicational constraints which are expected to work in both direction, i.e. when $\Gamma_\ell(w, w')$ becomes inconsistent, $w' \notin \ell(w)$ should be inferred.

We formulate in Figure 7 the constraint $\Phi_P(V, \mathcal{L}, \mathcal{E}, \mathcal{A}, (\Gamma_\ell))$ which characterizes the attributed graphs that satisfy the grammatical principles (Γ_ℓ) .

$$\Phi_P(V, \mathcal{L}, \mathcal{E}, \mathcal{A}, (\Gamma_\ell)) \equiv \forall w, w' \in V, \forall \ell \in \mathcal{L} \quad w' \in \ell(w) \Rightarrow \Gamma_\ell(w, w')$$

FIG. 7. well-formedness constraint for the satisfaction of grammatical principles

8 Disjunctive Attributes

Often it is convenient to use a set valued attribute $\alpha(w)$ to indicate a disjunction: any value in $\alpha(w)$ is licensed. We call α a disjunctive attribute. For example, distinct agreement values are frequently not morphologically distinguishable: while we could

nonetheless create many otherwise identical lexical entries that differed only in agreement, in the interest of lexical economy it is convenient to use an attribute $\text{agrs}(e)$ to represent the set of possible agreement values for lexical entry e .

For example, the German article “den” is either masculine singular accusative or any gender plural dative. However, only one value from this set may be assigned to the corresponding node. We formalize this notion as follows:

Let $\mathcal{A}_d \subseteq \mathcal{A}$ be a distinguished subset of so called ‘disjunctive’ attributes. They must be set-valued, i.e. for each $\alpha \in \mathcal{A}_d$, the lexical attribute has type $\alpha : \mathcal{E} \rightarrow 2^{\mathbb{N}}$. For each $\alpha \in \mathcal{A}_d$, there must exist a corresponding node attribute $\text{the}_\alpha : V \rightarrow \mathbb{N}$ defined as follows:

$$\text{the}_\alpha(w) \in \alpha(w) \quad (8.1)$$

For example, writing agr instead of the_{agrs} , we have:

$$\text{agr}(w) \in \text{agrs}(w)$$

which ensures that node w is assigned precisely one agreement value from the set of agreements licensed by its selected lexical entry. In practice, it is frequently legal for a parser to leave an attribute the_α underspecified. It is sound to so when it can be guaranteed that every partial solution that leaves the_α underspecified can be consistently extended to a complete solution that determines the_α . The conditions under which propagation is complete in this sense are outside the scope of this article.

9 Partially Ordered Projective Labeled Trees

The labeled trees considered so far are unordered and therefore cannot naturally express word order. In this section, we consider an extension to our formalization of labeled trees which overcomes this limitation.

Unordered labeled trees are adequate for the representation of ‘syntactic’ dependencies and sufficed for the treatment of immediate dependence presented in [4]. For a corresponding treatment of linear precedence, we proposed in [6] to introduce a second tree to represent ‘topological’ dependencies.

In the topological tree, both edges and nodes are labeled and the set of labels is totally ordered. As we shall see, thanks to this total order, it is possible to define the linearizations licensed by the tree. For concreteness, we present now an example:

dass Maria einen Mann wird lieben können⁵
that Maria a man will love can

Figure 8 displays its unordered non-projective syntax tree, while Figure 9 presents one possible topological tree. In the topological tree, edge labels are intended to correspond to the notion of *fields* in the classical topological sentence model [1], e.g.

⁵ that Maria will be able to love a man

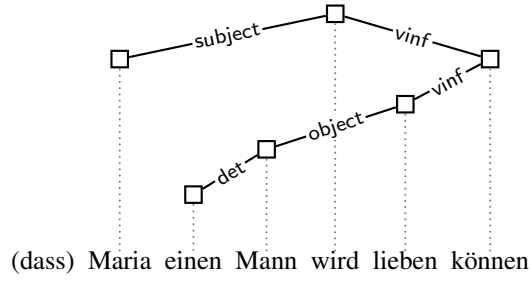


FIG. 8. Syntax tree

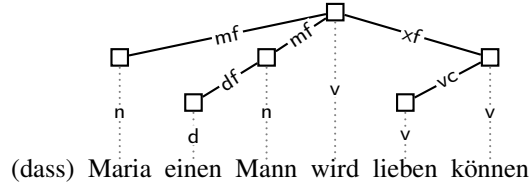


FIG. 9. Topological tree

mf represents the Mittelfeld and xf the extraposition field. Nodes are also labeled: in Figure 9 these labels are displayed on the vertical dotted lines joining a node to the word it stands for in the sentence. The total order assumed in the example is:

$$d \prec df \prec n \prec mf \prec vc \prec v \prec xf$$

Order on edge labels induces a partial order on the daughters of each node and, by extension, on the subtrees rooted at these daughters. For example ‘Mann’ and ‘können’ have respectively edge labels mf and xf. Since $mf \prec xf$, ‘einen Mann’ precedes ‘lieben können’. The order is partial because siblings with the same edge label are not respectively ordered: this is the basis for the account of scrambling in the Mittelfeld. The label assigned to a node allows to position it with respect to its daughters. For example ‘wird’ has node label v and, since $mf \prec v \prec xf$, it must occur between ‘einen Mann’ and ‘lieben können’. Thus the topological tree licenses 2 linearizations:

1. (dass) Maria einen Mann wird lieben können
2. (dass) einen Mann Maria wird lieben können

We will now distinguish a set $\mathcal{L}_E$ of edge labels and a set $\mathcal{L}_N$ of node labels, and assume given a total order $\prec$ on $\mathcal{L} = \mathcal{L}_E \uplus \mathcal{L}_N$. Given $(V, E) \in \mathbf{T}(V, \mathcal{L}_E)$, an assignment $I : V \rightarrow \mathcal{L}_N$ of node labels to nodes, and a total order $<$ on V , we say that

$G = (V, E, I, <)$ is a well-ordered tree if it satisfies the following conditions:

$$w - \ell_1 \rightarrow_G w_1 \wedge w - \ell_2 \rightarrow_G w_2 \wedge \ell_1 \prec \ell_2 \Rightarrow w_1 < w_2 \quad (9.1)$$

$$w_1 \rightarrow_G^* w'_1 \wedge w_2 \rightarrow_G^* w'_2 \wedge w_1 < w_2 \Rightarrow w'_1 < w'_2 \quad (9.2)$$

$$w - \ell_1 \rightarrow_G w_1 \wedge I(w) = \ell_2 \wedge \ell_1 \prec \ell_2 \Rightarrow w_1 < w \quad (9.3)$$

$$w - \ell_1 \rightarrow_G w_1 \wedge I(w) = \ell_2 \wedge \ell_2 \prec \ell_1 \Rightarrow w < w_1 \quad (9.4)$$

We define the additional functions $\text{proj}_G^\ell : V \rightarrow 2^V$ for $\ell \in \mathcal{L}$:

$$\begin{aligned} \text{proj}_G^\ell &= \rightarrow_G^* \circ -\ell \rightarrow_G && \text{for } \ell \in \mathcal{L}_E \\ \text{proj}_G^\ell(w) &= \begin{cases} \{w\} & \text{if } I(w) = \ell \\ \emptyset & \text{otherwise} \end{cases} && \text{for } \ell \in \mathcal{L}_N \end{aligned}$$

$\text{proj}_G^\ell(w)$ for $\ell \in \mathcal{L}_E$ is the set of nodes in the subtrees rooted at w 's ℓ -daughters. We overload $<$ to obtain a partial order on 2^V as follows:

$$S_1 < S_2 \quad \equiv \quad \forall w_1 \in S_1, \forall w_2 \in S_2 \quad w_1 < w_2 \quad \forall S_1, S_2 \subseteq V$$

It can be shown easily that the well-ordering conditions (9.1–9.4) are satisfied iff the following property holds:

$$\forall \ell_1, \ell_2 \in \mathcal{L}, \forall w \in V \quad \ell_1 \prec \ell_2 \Rightarrow \text{proj}_G^{\ell_1}(w) < \text{proj}_G^{\ell_2}(w) \quad (9.5)$$

Thus the well-ordering conditions can be simply realized by sequentiality constraints between sets. The well-ordered labeled trees with nodes V , edge labels in $\mathcal{L}_E$, node labels in $\mathcal{L}_N$, and respecting the total order $\prec$ on $\mathcal{L}_E \uplus \mathcal{L}_N$ are in bijection with the solutions of $\Phi_{\text{LP}}(V, \mathcal{L}_E, \mathcal{L}_N, \prec)$ shown in Figure 10.

$$\begin{aligned} \Phi_{\text{LP}}(V, \mathcal{L}_E, \mathcal{L}_N, \prec) &\equiv \Phi_{\text{ID}}(V, \mathcal{L}_E) \wedge \\ &\quad \forall w \in V \\ &\quad \{w\} = \uplus \{\ell(w) \mid \ell \in \mathcal{L}_N\} \\ &\quad \wedge \forall \ell \in \mathcal{L}_N \quad \text{proj}_G^\ell(w) = \ell(w) \wedge |\ell(w)| \neq 0 \equiv \ell = I(w) \\ &\quad \wedge \forall \ell \in \mathcal{L}_E \quad \text{proj}_G^\ell(w) = \cup \{\text{eqdown}(w') \mid w' \in \ell(w)\} \\ &\quad \wedge \forall \ell_1, \ell_2 \in \mathcal{L}_E \uplus \mathcal{L}_N \quad \ell_1 \prec \ell_2 \Rightarrow \text{proj}_G^{\ell_1}(w) < \text{proj}_G^{\ell_2}(w) \end{aligned}$$

FIG. 10. well-formedness condition for ordered labeled trees

It is possible, and in practice desirable, to improve propagation by formulating stronger constraints. For example, the property that all projections must be convex (i.e. intervals without holes) may be written:

$$\forall \ell \in \mathcal{L}_E, \forall w \in V \quad \text{convex}(\text{proj}_G^\ell(w)) \quad (9.6)$$

The declarative semantics of $\text{convex}(S)$ 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 $w \in S$.

10 Constraint-Based Dependency Parsing Revisited

In this section, we revisit the treatment of dependency parsing of [4] which we briefly outlined in Section 2. We show how to instantiate the framework developed in the preceding sections to obtain a mathematical characterization of admissible dependency syntax trees which also has a reading as a constraint program. Modulo issues of programming language syntax, the result is a parser. As in Section 2, we define a dependency grammar $\mathcal{G}$ by a 6-tuple:

$$\mathcal{G} = (Strs, Cats, Agrs, Roles, Lexicon, Rules)$$

where $Strs = \{\text{"Buch"}, \text{"hat"}, \dots\}$ is a finite set of strings notating the fully inflected forms of words, $Cats = \{\text{d}, \text{n}, \text{v_fin}, \dots\}$ is a finite set of categories which without loss of generality we can identify with (i.e. encode as) integers, $Agrs = Gender \times Number \times Person \times Case = \{\langle \text{masc sing 1 nom} \rangle, \dots\}$ is a finite set of agreement tuples which again we can identify with integers, $Roles = \{\text{det}, \text{adj}, \text{subject}, \text{object}, \text{dative}, \dots, \text{zuvinf}\}$ is a finite set of grammatical predicates to be used as edge labels. A lexical entry is an attribute value matrix with the following signature:

$$\begin{bmatrix} \text{str} & : & Strs \\ \text{cat} & : & Cats \\ \text{agrs} & : & 2^{Agrs} \\ | \cdot |_{\text{det}} & : & 2^{\mathbb{N}} \\ & \vdots & \\ | \cdot |_{\text{zuvinf}} & : & 2^{\mathbb{N}} \end{bmatrix}$$

which specifies phonology, category, possible agreements, and valency restrictions. The *Lexicon* is a finite set of lexical entries, and we choose a function $\text{lookup} : Strs \rightarrow Lexicon^*$ such that $\forall s \in Strs$, $\text{lookup}(s)$ is a sequence without duplicates formed from the elements of $\{e \mid e \in Lexicon \wedge \text{str}(e) = s\}$, i.e. such that:

$$e \in \text{lookup}(s) \quad \equiv \quad \text{str}(e) = s$$

An example lexical entry for ‘Buch’ is:

$$\begin{bmatrix} \text{str} & : & \text{"Buch"} \\ \text{cat} & : & \text{n} \\ \text{agrs} & : & \{\langle \text{masc sing 3 nom} \rangle, \\ & & \langle \text{masc sing 3 acc} \rangle, \\ & & \langle \text{masc sing 3 dat} \rangle\} \\ | \cdot |_{\text{det}} & : & \{1\} \\ | \cdot |_{\text{adj}} & : & \{0, \dots, \mu\} \\ | \cdot |_{\text{subject}} & : & \{0\} \\ & \vdots & \end{bmatrix}$$

It requires one determiner, permits any number of adjectives, forbids a subject, etc. . . Attribute agrs is identified as a disjunctive attribute and we write agr for the_{agrs} . Finally, *Rules* is a family (Γ_ℓ) of binary predicates indexed by roles and $\Gamma_\ell(w, w')$ must be satisfied to license a syntactic dependency labeled ℓ from head w to argument w' . Posing $\text{NP} = \{\text{n}, \text{pro}\}$ for the set of noun phrase categories and $\text{NOM} = \text{Gender} \times \text{Number} \times \text{Person} \times \{\text{nom}\}$ for the set of agreements with case nominative, we define these predicates using the language of Section 7. For illustration, here are the predicates for roles *det* and *subject*:

$$\begin{aligned}\Gamma_{\text{det}} &\equiv \lambda w, w'. \text{ cat}(w') = \text{d} \wedge \text{agr}(w) = \text{agr}(w') \\ \Gamma_{\text{subject}} &\equiv \lambda w, w'. \text{ cat}(w') \in \text{NP} \wedge \text{agr}(w) = \text{agr}(w') \wedge \text{agr}(w') \in \text{NOM}\end{aligned}$$

Constraint Formulation. We now provide the translation scheme which transforms an input sentence $s_1 \dots s_n$ into a constraint $\llbracket s_1 \dots s_n \rrbracket$ that precisely characterizes its valid dependency analyses according to our grammar $\mathcal{G}$. The translation follows the framework developed in the preceding sections. We introduce a set $V = \{w_1, \dots, w_n\}$ of nodes, one for each word of the input sentence. We pose $\mathcal{L} = \text{Roles}$, $\mathcal{E} = \text{Lexicon}$, $\mathcal{A} = \{\text{cat}, \text{agrs}, | \cdot |_{\text{det}}, \dots, | \cdot |_{\text{zuvinf}}\}$, $\mathcal{A}_d = \{\text{agrs}\}$, and $\text{lex}(w_i) = \text{lookup}(s_i)$ for $1 \leq i \leq n$. All valid analyses are given by the solutions of:

$$\Phi_{\text{ID}}(V, \mathcal{L}) \wedge \Phi_{\text{LEX}}(V, \mathcal{E}, \mathcal{A}, \text{lex}) \wedge \Phi_{\text{VAL}}(V, \mathcal{L}, \mathcal{E}, \mathcal{A}) \wedge \Phi_{\text{P}}(V, \mathcal{L}, \mathcal{E}, \mathcal{A}, (\Gamma_\ell))$$

thus we arrive at the translation shown in Figure 11.

In order to obtain a concrete parser, we must also stipulate a search strategy. In practice, the following strategy has proven quite satisfactory: first, apply distribution rules to determine all daughter sets $\ell(w)$ for $w \in V$, $\ell \in \mathcal{L}$, then apply distribution rules to determine all other attributes.

The account of dependency parsing presented here is only concerned with syntactic dependencies and ignores word-order. We have pursued two avenues of approach for the treatment of linear precedence:

Statistically Preferred Reading. The first approach for the treatment of word-order is based on statistical methods and was developed jointly with Thorsten Brants. Assuming the input sentence is a well-formed utterance, of all the analyses which our program is able to derive, some are more likely to correspond to the actual linearization than others. Thus, the idea is to inform the search strategy using statistics derived from a corpus. At each choice point where we need to decide an edge, we pick first the one which the statistical oracle ranks as most likely. This approach has proven quite successful at deriving first the intended reading, even in the presence of repeated extrapositions.

Topological Dependency Trees. The second approach aims at formalizing the principles of linear precedence and is the subject of current research with Ralph Debusmann. Corresponding to the non-ordered tree of syntactic dependencies, we postulate

$$\begin{aligned}
\llbracket s_1 \dots s_n \rrbracket = & \\
& V = \text{roots} \uplus \uplus \{\text{daughters}(w) \mid w \in V\} \\
\wedge & \quad |\text{roots}| = 1 \\
\wedge & \quad \forall w \in V \\
& \quad \text{eqdown}(w) = \{w\} \uplus \text{down}(w) \\
\wedge & \quad \text{down}(w) = \cup \langle \text{eqdown}(w_1), \dots, \text{eqdown}(w_n) \rangle [\text{daughters}(w)] \\
\wedge & \quad \text{daughters}(w) = \text{det}(w) \uplus \dots \uplus \text{zuvinf}(w) \\
\wedge & \quad \text{cat}(w) = \langle \text{cat} \mid \text{lex}(w) \rangle [\text{entry}(w)] \\
\wedge & \quad \text{agrs}(w) = \langle \text{agrs} \mid \text{lex}(w) \rangle [\text{entry}(w)] \\
\wedge & \quad |w|_{\text{det}} = \langle |\cdot|_{\text{det}} \mid \text{lex}(w) \rangle [\text{entry}(w)] \\
& \quad \vdots \\
\wedge & \quad |w|_{\text{zuvinf}} = \langle |\cdot|_{\text{zuvinf}} \mid \text{lex}(w) \rangle [\text{entry}(w)] \\
\wedge & \quad |\text{det}(w)| \in |w|_{\text{det}} \\
& \quad \vdots \\
\wedge & \quad |\text{zuvinf}(w)| \in |w|_{\text{zuvinf}} \\
\wedge & \quad \text{agr}(w) \in \text{agrs}(w) \\
\wedge & \quad \forall w' \in V \quad w' \in \text{det}(w) \Rightarrow \Gamma_{\text{det}}(w, w') \\
& \quad \vdots \\
& \quad \wedge w' \in \text{zuvinf}(w) \Rightarrow \Gamma_{\text{zuvinf}}(w, w')
\end{aligned}$$

FIG. 11. Translation into constraint

a tree of topological dependencies (see Figure 9) that is partially ordered and projective. These two structures are mutually constraining. This approach instantiates the framework developed in the present article including the extension of Section 9 for projective partially ordered analyses. In [6], we used it to provide an elegant account of the challenging phenomena in the German verb complex.

11 Constraint Programming

In this section, we precisely describe the constraint programming support necessary and sufficient to express and solve the CSPs presented earlier in the article. We propose a formal system of constraints and inference rules, and define constraint propagation as deterministic inferential saturation.

Let $\Delta = \{0, \dots, \mu\}$ be an interval of integers for some sufficiently large practical limit μ . We assume an infinite set of FD variables written I, I', I_j with values in Δ and an infinite set of FS variables written S, S', S_j with values in 2^Δ . We write D, D', D_j for a ‘domain’, i.e. a fixed subset of Δ , and i, j, k, n for particular integers in Δ .

Initialization		$\rightarrow I \in \Delta$
		$\rightarrow \emptyset \subseteq S$
		$\rightarrow S \subseteq \Delta$
Clash		
	$I \in \emptyset$	$\rightarrow \text{false}$
	$D_1 \subseteq S \wedge S \subseteq D_2 \wedge D_1 \not\subseteq D_2$	$\rightarrow \text{false}$
Strengthen		
	$I \in D_1 \wedge I \in D_2$	$\rightarrow I \in D_1 \cap D_2$
	$D_1 \subseteq S \wedge D_2 \subseteq S$	$\rightarrow D_1 \cup D_2 \subseteq S$
	$S \subseteq D_1 \wedge S \subseteq D_2$	$\rightarrow S \subseteq D_1 \cap D_2$
Weaken		
	$I \in D$	$\rightarrow I \in D' \quad D \subseteq D' \subseteq \Delta$
	$D \subseteq S$	$\rightarrow D' \subseteq S \quad D' \subseteq D$
	$S \subseteq D$	$\rightarrow S \subseteq D' \quad D \subseteq D' \subseteq \Delta$

FIG. 12. Rules for basic constraints

11.1 Basic Constraints

Information about a satisfying assignment will be refined incrementally by alternating steps of deterministic ‘propagation’ and non-deterministic ‘distribution’. Therefore we need means to represent partial information about an assignment and this is realized by ‘basic constraints’.

A basic constraint for a FD variable I takes the form $I \in D$ for a domain D . On the other hand, the assignment to a FS variable S is approximated by lower and upper bounds, i.e. by basic constraints $D_1 \subseteq S$ and $S \subseteq D_2$.

Basic constraints are given by the following abstract syntax:

$$B ::= \text{false} \mid I \in D \mid D \subseteq S \mid S \subseteq D \mid B_1 \wedge B_2$$

and they are subject to the inference rules of Figure 12. Saturation under these rules guarantees that a constraint B is either inconsistent (i.e. contains false) or that for each FD variable I in B , there is a most specific basic constraint $I \in D$, and that for each FS variable S in B , there are most specific lower- and upperbound basic constraints $D_1 \subseteq S$ and $S \subseteq D_2$. Posing:

$$\begin{aligned} \lfloor I \rfloor &= \cap \{D \mid (I \in D) \in B\} \\ \lfloor S \rfloor &= \cup \{D \mid (D \subseteq S) \in B\} \\ \lceil S \rceil &= \cap \{D \mid (S \subseteq D) \in B\} \end{aligned}$$

when B is saturated, all of $I \in \lfloor I \rfloor$, resp. $\lfloor S \rfloor \subseteq S$ and $S \subseteq \lceil S \rceil$ are in B and are the most specific bounds on variables I , resp. S . Indeed, we should use B as a subscript

$$\begin{array}{lll}
i \in S & \equiv & \{i\} \subseteq S \\
i \notin S & \equiv & S \subseteq \Delta \setminus \{i\} \\
I = i & \equiv & I \in \{i\} \\
I \neq i & \equiv & I \in \Delta \setminus \{i\}
\end{array}
\qquad
\begin{array}{lll}
S = D & \equiv & D \subseteq S \wedge S \subseteq D \\
I \leq n & \equiv & I \in \{n, \dots, \mu\} \\
n \leq I & \equiv & I \in \{0, \dots, n\}
\end{array}$$

FIG. 13. Abbreviations

and write $\lfloor I \rfloor_B$, $\lfloor S \rfloor_B$ and $\lceil S \rceil_B$, but, since this is never ambiguous, we will omit it to avoid notational clutter. All inference rules given in the following are monotonously increasing in their premises: more specific premises yield more specific conclusions. For this reason, it is sufficient and simpler to express them in terms of $\lfloor I \rfloor$, $\lfloor S \rfloor$ and $\lceil S \rceil$.

11.2 Non-Basic Constraints

We now extend our constraint language with ‘non-basic’ constraints, also known as ‘propagators’, and express their semantics in the form of inference rules.

$$\begin{aligned}
C ::= & B \mid C_1 \wedge C_2 \mid I \in S \mid I \notin S \mid S = \{I\} \mid |S| = I \mid \\
& I_1 \leq I_2 \mid S_1 \subseteq S_2 \mid S_1 \prec S_2 \mid \text{convex}(S) \mid \\
& S = \cup \langle S_1, \dots, S_n \rangle [S'] \mid S = \cap \langle S_1, \dots, S_n \rangle [S']
\end{aligned}$$

To increase legibility, we adopt the abbreviations of Figure 13. For example the semantics of the membership constraint $I \in S$ is given by the following inference rules:

$$\begin{array}{ll}
& \rightarrow I \in \lceil S \rceil \\
I = i & \rightarrow i \in S
\end{array}$$

Figure 14 lists all primitive binary constraints and Figure 15 our two primitive selection constraints. We can further extend our constraint language with the following derived constraints defined in Figure 16:

$$\begin{aligned}
C ::= & \dots \mid I_1 = I_2 \mid S_1 = S_2 \mid S_1 \parallel S_2 \mid \\
& S = S_1 \cup \dots \cup S_n \mid S = S_1 \cap \dots \cap S_n \mid S_1 \prec \dots \prec S_n \mid \\
& S = \langle S_1, \dots, S_n \rangle [I'] \mid I = \langle I_1, \dots, I_n \rangle [I']
\end{aligned}$$

The curious reader will find in [18] a detailed account of set constraints in Oz [19].

11.3 Disjunctive Propagators

Finally, we extend our constraint language with ‘disjunctive’ propagators:

$$C ::= \dots \mid C_1 \text{ or } C_2$$

$I \in S$	$\equiv$	$\rightarrow I \in [S]$ $I = i \rightarrow i \in S$
$I \notin S$	$\equiv$	$I = i \rightarrow i \notin S$ $\rightarrow I \in \Delta \setminus [S]$
$I_1 \leq I_2$	$\equiv$	$\rightarrow I_2 \in \{\min(\lfloor I_1 \rfloor), \dots, \mu\}$ $\rightarrow I_1 \in \{0, \dots, \max(\lfloor I_2 \rfloor)\}$
$S_1 \subseteq S_2$	$\equiv$	$\rightarrow [S_1] \subseteq S_2$ $\rightarrow S_1 \subseteq [S_2]$
$ S = I$	$\equiv$	$\rightarrow [S] \leq I$ $\rightarrow I \leq [S] $ $n \leq I \wedge [S] = n \rightarrow [S] \subseteq S$ $I \leq n \wedge [S] = n \rightarrow S \subseteq [S]$
$S_1 \prec S_2$	$\equiv$	$[S_1] \neq \emptyset \rightarrow S_2 \subseteq \{\min(\mu, \max(\lfloor S_1 \rfloor) + 1), \dots, \mu\}$ $[S_2] \neq \emptyset \rightarrow S_1 \subseteq \{0, \dots, \max(0, \min(\lfloor S_2 \rfloor) - 1)\}$
$\text{convex}(S)$	$\equiv$	$[S] \neq \emptyset \rightarrow \{\min(\lfloor S \rfloor), \dots, \max(\lfloor S \rfloor)\} \subseteq S$
$S = \{I\}$	$\equiv$	$I \in S \wedge S = I' \wedge I' = 1$ $\rightarrow S \subseteq [I]$

FIG. 14. Constraints as sets of inference rules

The declarative semantics of C_1 or C_2 is simply that of disjunction. In Logic Programming, the only method for processing complex disjunctions is non-determinism. Thus in Prolog, writing $C_1 ; C_2$ operationally results in first trying C_1 , and, if that fails, backtracking and trying C_2 instead. This has several drawbacks: (1) it is not sound (failure to prove C_1 is not the same as proving $\neg C_1$), (2) it forces the computation to commit immediately to exploring either one alternative or the other.

Early commitment is a poor strategy. It is often preferable to delay a choice until sufficient information is available to reject one of the alternatives. That is the intuition underlying the disjunctive propagator: C_1 or C_2 is a propagator not a choice point. It blocks until either C_1 or C_2 becomes inconsistent with respect to the current store of basic constraints: at that point, the propagator commits, i.e. reduces to, the remaining alternative. In this way, a disjunctive propagator has the declarative semantics of sound

$S = \cup\langle S_1, \dots, S_n \rangle[S'] \equiv$	$\rightarrow S' \subseteq \{1, \dots, n\}$
	$\rightarrow S \subseteq \cup\{[S_j] \mid j \in [S']\}$
	$\rightarrow \cup\{[S_j] \mid j \in [S']\} \subseteq S$
$[S_j] \not\subseteq [S] \rightarrow j \notin S'$	
$[S] \setminus \cup\{[S_j] \mid j \in [S'] \setminus \{k\}\} \neq \emptyset \rightarrow$	
$k \in S' \wedge [S] \setminus \cup\{[S_j] \mid j \in [S'] \setminus \{k\}\} \subseteq S_k$	
$S = \cap\langle S_1, \dots, S_n \rangle[S'] \equiv$	$\rightarrow S' \subseteq \{1, \dots, n\}$
	$\rightarrow \cap\{[S_j] \mid j \in [S']\} \subseteq S$
$[S'] \neq \emptyset \rightarrow S \subseteq \cap\{[S_j] \mid j \in [S']\}$	
$[S] \not\subseteq [S_j] \rightarrow j \notin S'$	
$\cap\{[S_j] \mid j \in [S'] \setminus \{k\}\} \setminus [S] \neq \emptyset \rightarrow$	
$k \in S' \wedge S_k \subseteq \Delta \setminus (\cap\{[S_j] \mid j \in [S'] \setminus \{k\}\} \setminus [S])$	

FIG. 15. Selection constraints rules

$$\begin{aligned}
I_1 = I_2 &\equiv I_1 \leq I_2 \wedge I_2 \leq I_1 \\
S_1 = S_2 &\equiv S_1 \subseteq S_2 \wedge S_2 \subseteq S_1 \\
S = S_1 \cup \dots \cup S_n &\equiv S = \cup\langle S_1, \dots, S_n \rangle[S'] \wedge S' = \{1, \dots, n\} \\
S = S_1 \cap \dots \cap S_n &\equiv S = \cap\langle S_1, \dots, S_n \rangle[S'] \wedge S' = \{1, \dots, n\} \\
S_1 \prec \dots \prec S_n &\equiv \bigwedge\{S_i \prec S_j \mid 1 \leq i < j \leq n\} \\
S_1 \parallel S_2 &\equiv S = S_1 \cap S_2 \wedge S = \emptyset \\
S = \langle S_1, \dots, S_n \rangle[I'] &\equiv S = \cup\langle S_1, \dots, S_n \rangle[S'] \wedge S' = \{I'\} \\
I = \langle I_1, \dots, I_n \rangle[I'] &\equiv \\
S = \cup\langle S_1, \dots, S_n \rangle[I'] &\wedge S = \{I\} \wedge \bigwedge\{S_j = \{I_j\} \mid 1 \leq j \leq n\}
\end{aligned}$$

FIG. 16. Derived constraints

logical disjunction, unlike Prolog's ';' operator which implements merely negation-as-failure. The operational semantics are given by the rules below:

$$\frac{B \wedge C_1 \rightarrow^* \text{false}}{B \wedge (C_1 \text{ or } C_2) \rightarrow C_2} \quad \frac{B \wedge C_2 \rightarrow^* \text{false}}{B \wedge (C_1 \text{ or } C_2) \rightarrow C_1}$$

where we write $C \rightarrow^* \text{false}$ to mean that false is in the deterministic saturation of C under all propagation rules. In practice, disjunctive propagators are frequently useful for expressing implicational constraints such as (7.1):

$$w' \in \ell(w) \Rightarrow \Gamma_\ell(w, w') \equiv w' \in \ell(w) \wedge \Gamma_\ell(w, w') \text{ or } w' \notin \ell(w)$$

A precursor of disjunctive propagators was the idea of a 'deep guard', investigated for

example in the Logic Programming language AKL [14]. Deep guards have since been generalized and subsumed by first class notions of encapsulated speculative computations, supported in the constraint programming language Oz [19] via ‘computation spaces’ [23].

While disjunctive propagators are extremely powerful, many implicational constraints do not require their full power and may be adequately expressed by reified constraints [16, 18] which are more widely supported.

11.4 Search By Propagation And Distribution

The operational semantics of a constraint C is given by a system of inference rules as described above. Propagation is defined as inferential saturation under this system of inference rules and we write C^* for the saturation of C . A FD variable I is said to be ‘determined’ in C^* when $\lfloor I \rfloor_{C^*}$ is a singleton. A FS variable S is determined when $\lfloor S \rfloor_{C^*} = \lceil S \rceil_{C^*}$. When C^* does not contain false and all variables are determined, we have found a satisfying assignment β defined as follows for all variables I, S in C :

$$\{\beta(I)\} = \lfloor I \rfloor_{C^*} \quad \beta(S) = \lfloor S \rfloor_{C^*} \quad (11.1)$$

Constraint propagation alone may not be sufficient to determine all variables. In such an eventuality, it is necessary to perform a non-deterministic choice: this is what we call a ‘distribution’ step. We formalize this notion using distribution rules. A distribution rule has the form $\phi \rightarrow \psi_1 \vee \psi_2$ and non-deterministically infers either ψ_1 or ψ_2 when precondition ϕ is satisfied. We extend the system of inference rules for constraint C with distribution rules as shown below for all variables I, S in C :

$$\begin{aligned} i \in \lfloor I \rfloor &\rightarrow i = I \vee i \neq I \\ i \in \lceil S \rceil \setminus \lfloor S \rfloor &\rightarrow i \in S \vee i \notin S \end{aligned}$$

A saturation of C under both propagation and distribution rules either contains false or determines all variables.

THEOREM 11.1

C is satisfiable iff it has a consistent saturation.

All rules given are valid implications, therefore C is equivalent to the disjunction of its saturations. Consequently, if C is satisfiable, at least one of its saturations does not contain false (*Soundness*). Conversely, every consistent saturation defines an assignment β of values to variables as shown in (11.1). This assignment defines a model of C . We show this for the ‘selection union’ constraint whose declarative semantics is given by:

$$S = \cup\{S_1, \dots, S_n\}[S'] \quad \equiv \quad S = \cup\{S_j \mid 1 \leq j \leq n, j \in S'\}$$

The first three propagations rules stipulating its operational semantics are:

$$\begin{aligned} &\rightarrow S' \subseteq \{1, \dots, n\} \\ &\rightarrow S \subseteq \cup\{\lceil S_j \rceil \mid j \in \lceil S' \rceil\} \\ &\rightarrow \cup\{\lfloor S_j \rfloor \mid j \in \lfloor S' \rfloor\} \subseteq S \end{aligned}$$

Therefore in a consistent saturation we have:

$$\beta(S') \subseteq \{1, \dots, n\} \quad \beta(S) = \cup\{\beta(S_j) \mid j \in \beta(S')\}$$

which proves that $\beta \models S = \cup\{S_j \mid 1 \leq j \leq n, j \in S'\}$. We can proceed similarly for every constraint in our language and thus establish by induction that, if C has a consistent saturation, it is satisfiable (*Completeness*).

Search Strategies. The formal framework remains uncommitted as to when to apply a non-deterministic distribution rule and which one to choose. It is clear that in order to minimize search, distribution rules should be postponed as long as possible. Only when propagation has reached a fixed point should we consider applying a distribution rule.

Which distribution rule to choose is the province of a search strategy. A well-known traditional search strategy is ‘first-fail’: it chooses a non-determined variable with the smallest number of remaining possible values and enumerates its assignments. The intent is to keep the branching factor low in the search tree.

12 Conclusion

In this article, we developed a concise mathematical formalization of finite trees with labeled edges. We have argued that this formulation can be regarded as the specification of a constraint satisfaction problem and that the latter can be solved directly and efficiently with constraint programming, provided we empower our mathematical instruments with a computational reading which grants them the operational semantics of constraint propagators.

We entertained various refinements of our framework of interest to the computational linguist: lexical attributes and the treatment of lexical ambiguity, lexicalized valency constraints, disjunctive attributes and the improvement of lexical economy, the formulation of grammatical principles, and a framework for a class of partially ordered projective trees.

A novelty of our approach is the central importance given to finite sets and constraints expressed in terms of variables denoting finite sets. Selection constraints, which we first proposed in [4], allow to give concise mathematical expressions a direct computational reading and are the foundation for an effective treatment of ambiguity that takes full advantage of constraint propagation and constructive disjunction. An important contribution of this article is the identification of a family of aggregative selection constraints. In particular, the ‘selection union’ constraint is revealed as particularly expressive and versatile.

Finally we addressed the issue of efficient processing and characterized the services expected of a constraint programming system through a formal system of constraints and inference rules of propagation and distribution. We showed that the search for solutions of a constraint satisfaction problem could be defined as the derivation of consistent saturations. It is our hope that having provided precise formal blueprints of the constraint propagation mechanisms required will facilitate their adoption and integration in other systems.

The general framework presented here can be variously instantiated. It underlies both our treatment of immediate dependence [4] and of linear precedence [6] for parsing with a dependency grammar.

References

- [1] Gunnar Bech. *Studien über das deutsche Verbum infinitum*. Munksgaard, Copenhagen, 1955.
- [2] Mehmet Dincbas, Pascal Van Hentenryck, Helmut Simonis, Abderrahmane Aggoun, Thomas Graf, and F. Berthier. The constraint logic programming language CHIP. In *Proceedings of the International Conference on Fifth Generation Computer Systems FGCS-88*, pages 693–702, Tokyo, Japan, December 1988.
- [3] Jochen Dörre and Andreas Eisele. Feature logic with disjunctive unification. In *COLING-90*, volume 2, pages 100–105, 1990.
- [4] Denys Duchier. Axiomatizing dependency parsing using set constraints. In *Sixth Meeting on Mathematics of Language*, pages 115–126, Orlando, Florida, July 1999.
- [5] Denys Duchier. Set constraints in computational linguistics – solving tree descriptions. In *Workshop on Declarative Programming with Sets (DPS’99)*, pages 91–98, September 1999.
- [6] Denys Duchier and Ralph Debusmann. Topological dependency trees: A constraint-based account of linear precedence. In *39th Annual Meeting of the Association for Computational Linguistics (ACL 2001)*, Toulouse, France, 9–11 July 2001.
- [7] Denys Duchier and Claire Gardent. A constraint-based treatment of descriptions. In H.C. Bunt and E.G.C. Thijsse, editors, *Third International Workshop on Computational Semantics (IWCS-3)*, pages 71–85, Tilburg, NL, January 1999.
- [8] Denys Duchier and Joachim Niehren. Dominance constraints with set operators. In *Proceedings of the First International Conference on Computational Logic (CL2000)*, LNCS. Springer, July 2000.
- [9] Denys Duchier and Stefan Thater. Parsing with tree descriptions: a constraint-based approach. In *Sixth International Workshop on Natural Language Understanding and Logic Programming (NLULP’99)*, pages 17–32, Las Cruces, New Mexico, December 1999.
- [10] Markus Egg, Joachim Niehren, Peter Ruhrberg, and Feiyu Xu. Constraints over lambda-structures in semantic underspecification. In *Joint Conf. COLING/ACL*, pages 353–359, 1998.
- [11] Dale Gerdemann. *Parsing and Generation of Unification Grammars*. PhD thesis, University of Illinois, 1991.
- [12] Carmen Gervet. Interval propagation to reason about sets: Definition and implementation of a practical language. *Constraints*, 1(3):191–244, 1997.
- [13] John Griffith. Modularizing contexted constraints. In *COLING-96*, 1996.
- [14] Sverker Janson. *AKL - A Multiparadigm Programming Language*. PhD thesis, SICS Swedish Institute of Computer Science, SICS Box 1263, S-164 28 Kista, Sweden, 1994. SICS Dissertation Series 14.
- [15] Ronald M. Kaplan and Joan Bresnan. Lexical-functional grammar: A formal system for grammatical representation. In Joan Bresnan, editor, *The Mental Representation of Grammatical Relations*, pages 173–281. MIT Press, 1982.

- [16] Kim Marriott and Peter J. Stuckey. *Programming with Constraints. An Introduction*. The MIT Press, Cambridge, MA, USA, 1998.
- [17] John T. Maxwell and Ronald M. Kaplan. An overview of disjunctive constraint satisfaction. In *Proceedings of the International Workshop on Parsing Technologies*, pages 18–27, 1989.
- [18] Tobias Müller. *Constraint Propagation in Mozart*. Doctoral dissertation, Universität des Saarlandes, Naturwissenschaftlich-Technische Fakultät I, Fachrichtung Informatik, Saarbrücken, Germany, 2001. In preparation.
- [19] Mozart Consortium. The Mozart programming system, 1999. <http://www.mozart-oz.org/>.
- [20] Tobias Müller and Martin Müller. Finite set constraints in Oz. In François Bry, Burkhard Freitag, and Dietmar Seipel, editors, *13. Workshop Logische Programmierung*, pages 104–115, Technische Universität München, 1997.
- [21] Carl Pollard and Ivan A. Sag. *Information-Based Syntax and Semantics*, volume 1 of *CSLI Lecture Notes*. CSLI, 1987.
- [22] Carl Pollard and Ivan A. Sag. *Head-driven Phrase Structure Grammar*. Studies in Contemporary Linguistics. The University of Chicago Press, 1994.
- [23] Christian Schulte. *Programming Constraint Services*. Doctoral dissertation, Universität des Saarlandes, Naturwissenschaftlich-Technische Fakultät I, Fachrichtung Informatik, Saarbrücken, Germany, 2000. To appear in *Lecture Notes in Artificial Intelligence*, Springer-Verlag.

Received 20 May 2001