Lexicalised Configuration Grammars*

Robert Grabowski and **Marco Kuhlmann** and **Mathias Möhl**
Programming Systems Lab
Saarland University
Saarbrücken, Germany

Abstract

This paper introduces Lexicalised Configuration Grammars (LCGs), a new declarative framework for natural language syntax. LCG is powerful enough to encode a large number of existing grammar formalisms, facilitating their comparison from the perspective of graph configuration (Debusmann et al., 2005). Once a formalism has been encoded as an LCG, the framework offers various means to increase its expressivity in a controlled manner. Trading expressive power for computational complexity, this makes it possible to model syntactic phenomena in novel ways. Parsing algorithms for LCGs lend themselves to a combination of chart-based and constraint-based processing techniques, allowing both to bring in their strengths.

1 Introduction

Formal accounts of natural language syntax may differ in their understanding of *grammar*. In *generative frameworks*, grammars are systems of *derivation rules*; well-formed expressions correspond to successful derivations in these systems. In *descriptive frameworks*, grammars are complex constraints on syntactic structures; well-formed structures are those that satisfy a grammar. This paper presents Lexicalised Configuration Grammars (LCGs), a new descriptive framework for the syntactic analysis of natural language.

* This paper is the extended version of an article that appears in the proceedings of the 2nd International Workshop on Constraint Solving and Language Processing, Barcelona, Spain, 2005. 2006-05-24

Structures and constraints LCG does not replace existing grammar formalisms; it offers a formal landscape into which these formalisms can be embedded to study them and their relations from a different angle: as description languages for syntactic structures. To be expressed as an LCG, a grammar formalism needs to be characterised by two choices: (1) What structures does it describe? and (2) What constraints does it use to describe them? To illustrate this, we will show how context-free grammars (CFGs) fits into LCG.

Following McCawley (1968), CFGs can be seen as description languages for ordered, labelled trees (Choice 1). More precisely, let $G = (N, T, P, S)$ be a CFG with N and T being the alphabets of non-terminal and terminal symbols, respectively, P the set of productions, and $S \in N$ the start symbol. A node u *satisfies* G if either (a) u is a leaf node labelled with a terminal symbol, or (b) u is an inner node with successors $u_1, \ldots, u_k$ (in that order), P contains a rule $A \to \alpha_1 \cdots \alpha_k$ (where $A \in N$ and $\alpha_i \in N \cup T$), u is labelled with A, and each successor u_i of u is labelled with α_i; that is, the order of the successors of u is compatible with the order specified by the rule (Choice 2). An ordered, labelled tree *satisfies* G if its root node is labelled with π, its frontier is s, and all of its nodes satisfy G.

Global and local constraints The choice of a class of reference structures for an LCG grammar formalism (Choice 1) imposes a *global* constraint on the formalism's expressivity. For example, by committing itself to ordered, labelled trees, no grammar specified in the LCG version of CFG can possibly account for syntactic structures with discontinuous configura-

tions, and no possible choice for the constraint language (Choice 2) can change that. Similarly, in previous work (Bodirsky et al., 2005), we have identified a class of discontinuous structures that is 'just right' for a descriptive view on Lexicalised Tree Adjoining Grammar (LTAG) (Joshi and Schabes, 1997). Adopting this class commits an LCG formalism to subsets of those syntactic structures describable by an LTAG.

The choice of the class of reference structures is the only non-lexical constraint expressible in LCG. This sets LCG formalisms apart from other formalisms employing constraints to restrict syntactic configurations, like the ID/LP format of Generalised Phrase Structure Grammar (Gazdar et al., 1985) or Constraint Dependency Grammar (CDG) (Maruyama, 1990). Both of these formalisms allow for the statement of non-lexical constraints at the level of individual *grammars* (order constraints in ID/LP grammars, all constraints in CDG). In contrast, global constraints in LCG can be imposed only by the choice of reference structures (Choice 1), which is a choice made at the level of the *formalism*. All remaining constraints are *local*: they apply to a word and the words in its immediate syntactic neighbourhood. In this sense, LCG is a *lexicalised* framework. The next section discusses the notion of locality employed in LCG and the role of lexical constraints in more detail.

Valencies and lexical constraints Locality is modelled through the concept of *valency*. The valency of a word w specifies the possible types of w (*accepted types*) and the number and types of words that w must connect with to form a complete expression (*required types*). The concept of valency is universal among lexicalised formalisms; it is implemented by nonterminal symbols in lexicalised CFG, syntactic roles in dependency grammar, and slashed categories in categorial grammar. When we say that lexical constraints apply to words and their immediate syntactic neighbourhoods, we mean that constraints in the lexical entry for w are relations over the words permitted by the valency of w. These words can be referred to by the accepted and required types of w.

We illustrate the idea behind lexical constraints by finalising our encoding of CFG as an LCG formalism. Assuming that we chose ordered, labelled trees as the reference class of structures (Choice 1), rules in a (lexicalised) CFG can be rewritten as LCG lexical entries using a single binary constraint relation $\prec$ to express linear precedence (Choice 2). For example, the rule $A \to B_1 w B_2 B_3$ (where $A, B_i \in N$ and $w \in T$) corresponds to the lexical entry

$$\langle \{A\}, \{B_1, B_2, B_3\} ; \{B_1 \prec \star, \star \prec B_2, B_2 \prec B_3\} \rangle .$$

The first component of this entry specifies the types accepted by w, the second component specifies the required types; thus, in a tree satisfying this entry, the node labelled with w must have a predecessor of type A and successors of types B_1, B_2, B_3. The third component of the entry contains the lexical constraints on the valency; for the example entry, the node labelled with w (denoted by $\star$ here) and its successors (referred to by their types) must be ordered as prescribed by the right hand side of the context-free rule. Note that this semantics exactly corresponds to McCawley's conception of CFG.

Increasing the expressivity Given that the LCG framework is parametrised with respect to the choice of the class of reference structures and the choice of the lexical constraint languages, there are two obvious ways how the expressivity of an LCG formalism can be increased:

- choose a more permissive class of structures (for example, the LTAG structures mentioned above instead of the ordered, labelled trees employed for the encoding of CFG);
- choose other constraint languages (for example, languages with structural constraints other than precedence, like isolation or adjacency (Suhre, 1999), or languages allowing for non-structural constraints such as agreement).

It turns out that LCG facilitates a rather detailed analysis of the implications that these two changes have in terms of the generative

capacity and the processing complexity of the resulting formalisms.

One of the main reasons why one might want to experiment with expressivity alternations is that for most traditional grammar formalisms, there is a small number of 'killer phenomena' for which it seems necessary to locally extend the expressiveness of the formalisms by just the right amount. In the case of English for example, while most syntactic configurations disallow discontinuities, a few (such as in *wh*-movement) require them. It seems desirable to be able to express context-free and non-context-free phenomena in the same formalism, investing extra formal and computational resources only in places where they really are required. We claim that LCG is suitable for such endeavours.

Another reason why we think that LCG is an interesting formal framework for modelling natural language is that it is able to handle linguistic phenomena that have proven to be particularly hard for other frameworks. As an example, we cite the permutation of nominal arguments in the German verb cluster known as *scrambling*. If we accept the linguistic analysis put forward by Becker et al. (Becker et al., 1992), the question whether a formalism can model scrambling boils down to asking whether it can generate the indexed language

$$\mathsf{SCR} = \{\, \pi(n^{[0]}, \ldots, n^{[k]}) v^{[0]} \cdots v^{[k]} \mid k \geq 0 \,\},$$

where π is some permutation, and the indices (written as superscripts) match up verbs (vs) with their noun arguments (ns). It has been shown (Becker et al., 1992) that no formalism in the class of Linear Context-Free Rewriting Systems[1] that produces a verb $v^{[i]}$ and the requirement for its matching noun argument $n^{[i]}$ in the same derivation step can generate SCR. In Section 3.3, we will present an LCG that does.

Previous work Lexicalised Configuration Grammar elaborates on our previous work on *graph configuration* (Debusmann et al., 2005), in which we have shown how a broad range of problems within computational linguistics can be modelled as tasks in which a finite number of elementary graph structures (*fragments*) has to be assembled into one reference structure, obeying both global and local constraints. In LCG, the fragments are specified by the lexical entries.

Structure We start our exposition by introducing *labelled drawings* as the universal class of reference structures for LCGs (Section 2). Section 3 presents the parametrised framework for constraint languages over drawings and gives some illustrative examples. In Section 4, we prove some limitative complexity results for LCG. Section 5 then addresses the issue of parsing LCGs and shows how the standard polynomial complexities for parsing can be obtained by appropriate restrictions on the structures and constraint languages. The paper concludes with an outlook on future work in Section 6.

Acknowledgements We thank Alexander Koller, Gert Smolka and the anonymous reviewers for useful comments on earlier versions of this paper. The work of Kuhlmann and Möhl is funded by the Collaborative Research Centre 378 of the Deutsche Forschungsgemeinschaft.

2 Labelled drawings

We introduce LCGs as description languages for (labelled) *drawings* (Bodirsky et al., 2005), a class of relational structures representing two essential syntactic dimensions: derivation structure and word order. Derivation structure captures the idea that a natural language expression can be composed of smaller expressions; word order concerns the possible linearisations of syntactic material. This section presents the basic terminology for drawings and cites some previous results.

2.1 Relational structures

A *relational structure* consists of a non-empty, finite set V of *nodes* and a number of relations on V. In this paper, we are mostly concerned with binary relations on the nodes. We use the standard terminology and notations available for binary relations. In particular, R^+ refers

[1] The class of Linear Context-Free Rewriting Systems includes, among other formalisms, Combinatory Categorial Grammar, LTAG, and local Multi-Component TAGs.

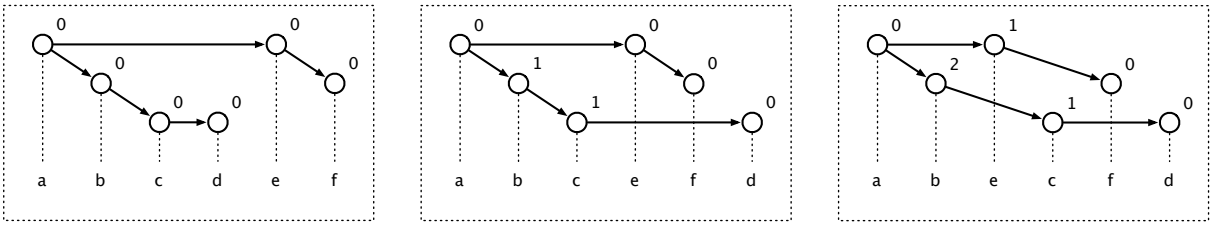


Figure 1: Drawings in $\mathcal{D}_0$ (projective drawings; left), $\mathcal{D}_1 - \mathcal{D}_0$ (gap degree 1; middle) and $\mathcal{D}_2 - \mathcal{D}_1$ (gap degree 2; right). An integer at a node states that node's gap degree.

to the transitive closure, R^* to the reflexive-transitive closure of R. The notation Ru stands for the relational image of u under R: the set of all v such that $(u, v) \in R$. Since relational structures with binary relations can also be seen as multigraphs, all the standard graph terminology can be applied to them.

Two types of relational structures are particularly important for the representation of syntactic configurations: *trees* and *total orders*. A relational structure $(V; \lhd)$ is a *forest* iff $\lhd$ is acyclic and every node in V has an indegree of at most one. Nodes with indegree zero are called *roots*. A *tree* is a forest with exactly one root. For a node v, we call the set $\lhd^* v$ the *yield* of v. A *total order* is a relational structure $(V; \prec)$ in which $\prec$ is transitive and for all $v_1, v_2 \in V$, exactly one of the following three conditions holds: $v_1 \prec v_2$, $v_1 = v_2$, or $v_2 \prec v_1$. Given a total order, the *interval* between two nodes v_1 and v_2 is the set of all v such that $v_1 \preceq v \preceq v_2$. A set is *convex* iff it is an interval. The *cover* of a set V', $C(V')$, is the smallest interval containing V'. A *gap* in a set V' is a maximal, non-empty interval in $C(V') - V'$; the number of gaps in V' is the *gap degree* of V'.

2.2 Drawings

Drawings are trees whose nodes are totally ordered.

Definition 2.1 A *drawing* is a relational structure $(V; \lhd, \prec)$ in which $(V; \lhd)$ forms a tree and $(V; \prec)$ forms a total order. ⊣

Note that drawings are not the same as ordered trees: in an ordered tree, only sibling nodes are ordered; in drawings, the order is total for *all* of the nodes.

The notions of cover, gap and gap degree can be applied to nodes in a drawing by identifying a node v with its yield $\lhd^* v$; for example, the gap degree of a node v is the gap degree of $\lhd^* v$. The gap degree of a drawing is the maximum among the gap degrees of its nodes. We write $\mathcal{D}_g$ for the class of all drawings whose gap degree does not exceed g. The drawings in $\mathcal{D}_0$ are called *projective*. Fig. 1 shows three drawings of the same tree structure but with different gap degrees.

The notion of gap degree yields a scale along which the non-projectivity of a drawing can be quantified. Orthogonal to that, there are linguistically relevant *qualitative* restrictions on non-projectivity. One of these is *well-nestedness*, which constrains the possible relations between gaps (Bodirsky et al., 2005).

Definition 2.2 Let $\mathfrak{D}$ be a drawing. Two disjoint trees T_1 and T_2 in $\mathfrak{D}$ *interleave* iff there are nodes $l_1, r_1 \in T_1$ and $l_2, r_2 \in T_2$ such that $l_1 \prec l_2 \prec r_1 \prec r_2$. The drawing $\mathfrak{D}$ is called *well-nested* iff it does not contain any interleaving trees. ⊣

We use the notation $\mathcal{D}_{wn}$ to refer to the class of all well-nested drawings. In Fig. 1, the first and the second drawing are well-nested; the third drawing contains two pairs of interleaving trees, rooted at b, e and c, e, respectively.

2.3 Labelled drawings

A labelled drawing is a drawing equipped with two total functions ℓ_V and ℓ_E: the function ℓ_V marks each node with a *node label* from an alphabet Σ, and ℓ_E marks each edge with an *edge label* from an alphabet Π. We write $\mathcal{D}_{\Sigma,\Pi}$ for the class of labelled drawings obtained by decorating drawings from class $\mathcal{D}$ with node labels from Σ and edge labels from Π.

In labelled drawings, *labelled successor relations* can be defined as follows:

$$\lhd_\pi \;:=\; \{\,(u,v) \mid u \lhd v \text{ and } \ell_E(u,v) = \pi\,\}.$$

To reduce the complexity of our presentation, we define the *accessibility relation* as the set of nodes reachable via an edge labelled with π:

$$\diamond_\pi := \lhd_\pi \circ \lhd^*$$

The *projection* of a labelled drawing $\mathfrak{D}$, $proj(\mathfrak{D})$, is the string obtained by concatenating the node labels of the drawing in the order of their corresponding nodes. Thus, the projection of a drawing in $\mathcal{D}_{\Sigma,\Pi}$ is a string over Σ.

3 Lexical constraint languages

The choice of a particular class of drawings imposes a global constraint on the syntactic structures allowed by an LCG formalism. In this section, we formalise the mechanism of lexical (local) constraints. As we illustrated in the introduction, the *lexical entry* for a given word w specifies the type of w and the types of the words connected to w, and imposes additional structural restrictions using constraints from a *lexical constraint language*. In our formal model, words will correspond to node labels, and types of nodes will correspond to edge labels. A lexical constraint between two types π_1, π_2 in the entry of a word $\ell_V(u)$ will be interpreted on the nodes reachable from u by the labelled successor relations named by π_1 and π_2.

3.1 Syntax and semantics

Syntax The syntax of a lexical constraint language is defined relative to an alphabet $\mathcal{R}$ of relation symbols and an alphabet Π of edge labels. The alphabet $\mathcal{R}$, together with a function *ar* that assigns every symbol $R \in \mathcal{R}$ a non-negative *arity* $ar(R)$, forms the *signature* of the language. We will leave the arity function implicit, and use the letter $\mathcal{R}$ to refer to signatures.

Definition 3.1 Let $\mathcal{R}$ be a signature, and let Π be an alphabet of edge labels. A *lexical constraint language* with signature $\mathcal{R}$ over Π, written $\mathcal{L}_\mathcal{R}(\Pi)$, consists of literals of the form $R(\pi_1, \ldots, \pi_k)$, where $R \in \mathcal{R}$, $ar(R) = k$, and $\pi_i \in \Pi$. We write $\mathcal{L}_\mathcal{R}$ for the class of all lexical constraint languages with signature $\mathcal{R}$. ⊣

Binary constraint literals will be written using infix notation, so the notation $\pi_1\ R\ \pi_2$ will stand for $R(\pi_1, \pi_2)$. Note that $\mathcal{L}_\emptyset$ is the class of languages that contains no constraints.

Semantics The satisfiability relation associated to a lexical constraint language $\mathcal{L}_\mathcal{R}(\Pi)$ is a ternary relation between a formula ϕ, a drawing $\mathfrak{D} \in \mathcal{D}_{\Sigma,\Pi}$ and a node u in that drawing. In this paper, we focus on satisfiability relations that meet the following requirement: the question whether a defining condition of a constraint applies must be decidable in time polynomial in the number of nodes in $\mathfrak{D}$. LCG as we define it here does not impose any further restrictions; it allows for defining arbitrary constraint languages for labelled drawings, if they meet the complexity condition.

To simplify our presentation, we assume the existence of a special edge label $\star$ called 'self', distinct from all other labels, and define $\lhd_\star := \mathrm{Id}$ and $\diamond_\star := \mathrm{Id}$. The self label allows for constraint definitions in which successors and the node u itself are referenced in the same manner. It constitutes a notational trick: instead, one could define additional special contraints with references to u integrated into their definition.

3.2 Theories and grammars

Within LCG, we distinguish between *theories* and *grammars*. Formally, an LCG *theory* is a pair of a class of (unlabelled) drawings and a class of lexical constraint languages. An LCG theory corresponds to a 'grammar formalism' in the usual sense of the word. An LCG *grammar* adopts a theory and instantiates it by choosing concrete alphabets for the node and edge labels, and a lexicon. An LCG *lexicon* is a mapping from node labels to sets of *lexical entries*. The type of a lexical entry depends on the signature of its constraint language and the alphabet of edge labels that the lexical constraints may refer to.

Definition 3.2 A *bag* (multiset) over a base set S is a function in $S \to \mathbb{N}_0$ that assigns ev-

ery element in S a natural number that tells how often the element appears in the bag. The set of all bags over S is written $\mathfrak{B}(S)$. An *option* over S is a bag over S that assigns 0 to all elements in S, possibly except for exactly one element which is assigned 1. An option is therefore equivalent to a set that contains either exactly one element of S or is empty. The set of all options over S is written $\mathfrak{O}(S)$.⊣

Definition 3.3 A lexical entry describes a node in a drawing. It is a triple $\langle I, \Omega ; \Phi \rangle \in LE_{\mathcal{R}}(\Pi)$, where the option $I \in \mathfrak{O}(\Pi)$ and the bag $\Omega \in \mathfrak{B}(\Pi)$ contain edge labels, and $\Phi \in \mathfrak{P}(\mathcal{L}_{\mathcal{R}}(\Pi))$ is a set of lexical constraints.

A node u in $\mathfrak{D} \in \mathcal{D}_{\Sigma,\Pi}$ *satisfies* a lexical entry $\langle I, \Omega ; \Phi \rangle \in LE_{\mathcal{R}}(\Pi)$ iff: for all $\pi \in \Pi$, $|(\lhd_\pi)^{-1}u| = I(\pi)$ and $|(\lhd_\pi)u| = \Omega(\pi)$; and $\mathfrak{D}, u \vDash \phi$ for all $\phi \in \Phi$. ⊣

A node u thus satisfies a lexical entry $\langle I, \Omega ; \Phi \rangle$ if and only if I contains exactly the ingoing edge of u, Ω is exactly the bag of outgoing edges, and u satisfies all constraints in the set of constraints Φ.

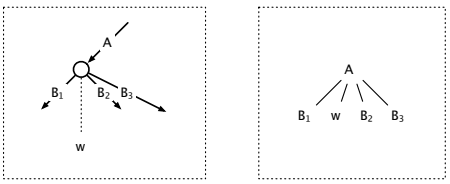
Definition 3.4 Let $T = (\mathcal{D}, \mathcal{L}_{\mathcal{R}})$ be a theory. A grammar of type T is a triple $G_T = (\Sigma, \Pi, Lex)$ such that Σ is an alphabet of node labels, Π is an alphabet of edge labels, and Lex is a lexicon of type $\Sigma \to \mathfrak{P}(LE_{\mathcal{R}}(\Pi))$. ⊣

Definition 3.5 A node $u \in \mathfrak{D} \in \mathcal{D}_{\Sigma,\Pi}$ *satisfies* a lexicon $Lex \in \Sigma \to \mathfrak{P}(LE_{\mathcal{R}}(\Pi))$ iff there is a lexical entry $\langle I, \Omega ; \Phi \rangle \in Lex(\ell_V(u))$ such that u satisfies $\langle I, \Omega ; \Phi \rangle$. A drawing $\mathfrak{D}$ *satisfies* a grammar G, written $\mathfrak{D} \vDash G$, iff every node $u \in \mathfrak{D}$ satisfies the lexicon of G. ⊣

3.3 Sample languages

To provide an intuition for the formal concepts defined in the previous two sections, we will now translate three grammar formalisms into LCG theories. We start by adapting our previous encoding of LCFG to the new formal concepts.

Lexicalised Context-Free Grammars As mentioned in the introduction, lexicalised context-free rules like $A \to B_1 w B_2 B_3$ can be seen as local well-formedness conditions on node-labelled, ordered trees (see Fig. 2).



$w : \langle \{A\}, \{B_1, B_2, B_3\} ; \{B_1 \prec \star, \star \prec B_2, B_2 \prec B_3\} \rangle$

Figure 2: Encoding Lexicalised Context Free Grammars

To express these conditions in the formal framework defined above, we first need to choose a class of drawings suitable as models for LCFGs. Since the yields of each non-terminal are continuous, a proper choice is $\mathcal{D}_0$, the class of projective drawings. Second, we need to choose a signature for the lexical constraint language that we want to use. As we already mentioned in the introduction, the only structural constraint relevant to LCFGs is linear order. Therefore, it suffices to have a single literal $\prec$ that imposes an order on the immediate successors of a node; since the language is interpreted on projective drawings, this order induces an order on the subtrees.

$$\mathfrak{D}, u \vDash \pi_1 \prec \pi_2 \qquad \text{iff} \quad \lhd_{\pi_1} u \times \lhd_{\pi_2} u \subseteq \prec$$

Fig. 2 shows a node-labelled tree, the corresponding lexical entry for the word w, and a (partial) drawing satisfying the entry. Note that (instances of) non-terminals in the LCFG rule correspond to edge labels in LCG. If A was a start symbol of the underlying grammar, the first component of the corresponding LCG entry would have to be the empty set; such entries can only be satisfied at root nodes.

Lexicalised Unordered Context-Free Grammar Since nothing forces us to impose order constraints on *all* types, we can write grammars corresponding to LCFGs with arbitrary permutations of the right hand sides of the rules. If we abandon the order constraints completely, we get the theory $(\mathcal{D}_0, \mathcal{L}_\emptyset)$, which is equivalent to the class of (lexicalised) unordered context-free grammars.

The scrambling language The following grammar derives drawings whose projections

$\mathfrak{D}, u \models \pi_1 \prec \pi_2$ iff $\diamond_{\pi_1} u \times \diamond_{\pi_2} u \subseteq \prec$

$\mathfrak{D}, u \models \pi_1 \bowtie \pi_2$ iff $(\lhd_{\pi_1} \cup \lhd_{\pi_2}) u$ convex

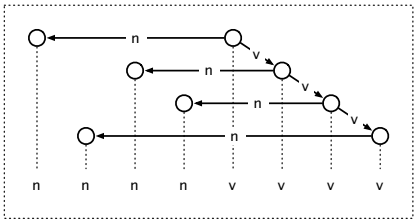


Figure 3: Lexical constraint language and sample drawing for SCR

form the scrambling language presented in the introduction. The underlying theory uses the unrestricted class of drawings and a constraint language with two literals $\prec$ (linear precedence) and $\bowtie$ (adjacency), whose semantics are specified in Fig. 3. The grammar is $G_{\mathsf{SCR}} := (\{n, v\}, \{n, v\}, Lex)$, where the lexicon Lex contains the entry $\langle\{n\}, \emptyset; \emptyset\rangle$ for n and the following entries for v:

$$\langle \emptyset, \{n, v\}\,; \{n \prec \star, \star \prec v, \star \bowtie v\}\rangle\,,$$
$$\langle \{v\}, \{n, v\}\,; \{n \prec \star, \star \prec v, \star \bowtie v\}\rangle\,,$$
$$\text{and} \quad \langle \{v\}, \{n\}\,; \{n \prec \star\}\rangle\,.$$

The precedence constraints place each v in between its n-successor and its v-successor. The adjacency constraint prevents material from entering between a v and its v-successor. Therefore, all nodes labelled with n must be placed to the left of all nodes labelled with v, and while the vs are ordered, the ns can appear in any permutation. (Fig. 3 shows a sample drawing licensed by G_{SCR}.)

Linear Specification Language Suhre's LSL formalism (Suhre, 1999) allows to generate languages with a free word order. It is inspired by ID/LP parsing (Gazdar et al., 1985), but allows for local constraints only, which makes it more suitable for translation into LCG. The yields in LSL are generally discontinuous; therefore, a theory for LSL needs to adopt the class of unrestricted drawings as its models. To restrict the possible linearisations, each LSL grammar rule can be annotated with local precedence and 'isolation' (zero-gap) constraints. These constraints can be translated into constraints from the lexical constraint language $\mathcal{L}_{\mathrm{LSL}}$ shown in Fig. 4.

4 Limitative complexity results

The previous section has demonstrated that the LCG framework is rather expressive. This expressive power does not come without a price. It is clear that all recognition problems for LCG are in NP: we can simply guess a labelled drawing and check the lexical constraints in polynomial time. The main result of the present section is the proof that the universal recognition problem for the most general LCG theory is NP-complete.

4.1 The universal recognition problem

Definition 4.1 Let $G = (\Sigma, \Pi, Lex)$ be a grammar for the theory $(\mathcal{D}, \mathcal{L}_{\mathcal{R}})$, and let s be a string over Σ. The *universal recognition problem* for G and s, written (G, s), is the problem to decide whether the following set is non-empty:

$$\mathfrak{C}(G, s) := \{\mathfrak{D} \in \mathcal{D}_{\Sigma,\Pi} \mid \mathfrak{D} \models G \text{ and } \mathit{proj}(\mathfrak{D}) = s\}$$

Elements of this set are called *configurations* of (G, s). ⊣

Lemma 4.2 The universal recognition problem for $(\mathcal{D}, \mathcal{L}_\emptyset)$ is NP-hard. □

PROOF We will present a polynomial reduction of HAMILTON PATH to the universal recognition problem for $(\mathcal{D}, \mathcal{L}_\emptyset)$. More specifically, for each input graph $H = (V; E)$ to HAMILTON PATH, we will construct (in time linear in the size of the input graph) a grammar G_H and a string s_H such that $\mathfrak{C}(G_H, s_H)$ is non-empty iff H has a Hamilton Path. Let s_H be some string over V, and define

$$\begin{aligned}
\Sigma_H, \Pi_H &:= V \\
S(v) &:= \{\langle \emptyset, \{v'\}\,; \emptyset\rangle \mid v \to v' \in H\} \\
I(v) &:= \{\langle \{v\}, \{v'\}\,; \emptyset\rangle \mid v \to v' \in H\} \\
E(v) &:= \{\langle \{v\}, \emptyset\,; \emptyset\rangle \mid v \to v' \in H\} \\
Lex_H &:= \{v \mapsto S(v) \cup I(v) \cup E(v) \mid v \in V\} \\
G_H &:= (\Sigma_H, \Pi_H, Lex_H)
\end{aligned}$$

$$
\begin{array}{lll}
\mathfrak{D}, u \models \pi_1 < \pi_2 & \text{iff} & \diamond_{\pi_1} u \times \diamond_{\pi_2} u \subseteq \prec \\
\mathfrak{D}, u \models \pi_1 \ll \pi_2 & \text{iff} & \diamond_{\pi_1} u \times \diamond_{\pi_2} u \subseteq \prec \text{ and } C(\diamond_{\pi_1} u) \cup C(\diamond_{\pi_2} u) \text{ convex} \\
\mathfrak{D}, u \models \langle \pi \rangle & \text{iff} & \diamond_{\pi} u \text{ is convex} \\
\mathfrak{D}, u \models \langle \bullet \rangle & \text{iff} & \lhd^* u \text{ is convex}
\end{array}
$$

Figure 4: Suhre's Linear Specification Language. The last clause corresponds to an isolation constraint applied to the left hand side of an LSL rule.

Each Hamilton Path in H forms a linear tree on V. Each such tree can be configured using G_H by choosing, for each node v in H, an entry from either $S(v)$, $E(v)$, or $I(v)$, depending on the position of v in the Hamilton Path (start, inner, or end). Conversely, in each configuration of (G_H, s_H), each node has at most one predecessor and at most one successor qua lexicon. Therefore, each such configuration is a drawing whose successor relation forms a linear tree, and the path from the root to the leaf identifies a Hamilton Path in H. ∎

To illustrate the encoding used in the proof, we show an example for an input graph H and a corresponding configuration in Fig. 5. The depicted drawing satisfies the following lexicon Lex_H. (The lexical entry satisfied at each node is underlined.)

Well-nested drawings Since linear drawings do not contain disjoint trees, all solutions to the configuration problem for Hamilton Graphs are well-nested.

Corollary 4.3 The universal recognition problem for $(\mathcal{D}_{wn}, \mathcal{L}_\emptyset)$ is NP-hard. □

Projective drawings While all solutions to the configuration problem for Hamilton Graphs are well-nested, there is no limit on their gap degree. One may wonder if restricting the gap degree reduces the complexity of the universal recognition problem. This, however, is not even the case for drawings without gaps:

Lemma 4.4 The universal recognition problem for $(\mathcal{D}_0, \mathcal{L}_\emptyset)$ is NP-hard. □

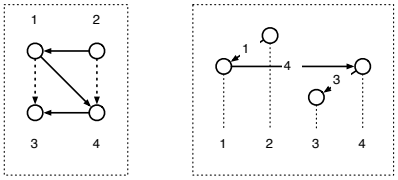
PROOF As indicated in section 3.3, each $(\mathcal{D}_0, \mathcal{L}_\emptyset)$ grammar can be transformed into an equivalent lexicalised unordered context-free grammar. Therefore, the universal recognition problem for $(\mathcal{D}_0, \mathcal{L}_\emptyset)$ can be reduced from the corresponding problem for LUCFGS; this problem is NP-hard (Barton, 1985). ∎

4.2 The fixed recognition problem

The fixed recognition problem asks the same question as the universal problem, but the grammar is not considered part of the input. This invalidates the reduction that we used in the previous section, as this reduction *constructed* a new grammar for every input, while any reduction for the fixed recognition problem needs to assume one fixed grammar for every input string.

Lemma 4.5 The fixed recognition problem for $(\mathcal{D}, \mathcal{L}_\emptyset)$ is polynomial. □

PROOF We assume the existence of a chart parser that solves the fixed recognition problem for $(\mathcal{D}, \mathcal{L}_\emptyset)$ and uses parse items to represent partial derivations. These parse items have the form $s : \langle I, \Omega \rangle$, where s represents the *span* of the input sentence covered by the partial derivation, and I and Ω represent the incoming and outgoing valencies that still need to be saturated. (We will show such a general chart-based parsing schema for LCGS in Section 5.) The time complexity of such a parser is polynomial in the number of possible parse items, but since for the fixed recognition problem, we may ignore the size of the grammar, we are only interested in the number of different spans. The string languages of $(\mathcal{D}, \mathcal{L}_\emptyset)$ are closed under permutation; therefore, each span can be represented by a Parikh vector (a mapping from node labels to natural numbers). As each $\sigma \in \Sigma$ can occur between zero and n times, the number of different such Parikh vec-



$1 \mapsto \{\langle \emptyset, \{3\}\,;\emptyset\rangle, \langle \emptyset, \{4\}\,;\emptyset\rangle, \langle \{1\}, \{3\}\,;\emptyset\rangle, \underline{\langle \{1\}, \{4\}\,;\emptyset\rangle}, \langle \{1\}, \emptyset\,;\emptyset\rangle\}$

$2 \mapsto \{\underline{\langle \emptyset, \{1\}\,;\emptyset\rangle}, \langle \emptyset, \{4\}\,;\emptyset\rangle, \langle \{2\}, \{1\}\,;\emptyset\rangle, \langle \{2\}, \{4\}\,;\emptyset\rangle, \langle \{2\}, \emptyset\,;\emptyset\rangle\}$

$3 \mapsto \{\underline{\langle \{3\}, \emptyset\,;\emptyset\rangle}\}$

$4 \mapsto \{\langle \emptyset, \{3\}\,;\emptyset\rangle, \underline{\langle \{4\}, \{3\}\,;\emptyset\rangle}, \langle \{4\}, \emptyset\,;\emptyset\rangle\}$

Figure 5: An input graph H for HAMILTON PATH and a drawing licensing Lex_H. The Hamilton Path in H is marked by solid edges.

tors is $(n+1)^{|\Sigma|} \in O(n^{|\Sigma|})$, which is polynomial in n. (An upper bound for Σ is the size of the grammar, which is considered constant for the fixed recognition problem.) ∎

It would seem desirable to have a framework in which extending the signature of the constraint language may only *reduce* the complexity of the recognition problem, but never increase it. For LCGs, however, this is not necessarily the case: in an unpublished manuscript, Holzer et. al. show—by a reduction of TRIPARTITE MATCHING—that for the Linear Specification Language, even the *fixed* recognition problem is NP-complete (p.c.); consequently, by the encoding of LSL presented in Section 3.3, the same result applies to LCGs.

5 Parsing Lexicalised Configuration Grammars

This section presents a general schema for chart-based approaches to parsing LCGs. Parsing schemata (Sikkel, 1997) provide us with a declarative specification of concrete parsing algorithms, and allow us to analyse the complexity of these algorithms on a high level of abstraction, hiding the algorithmic details. The complexity and even the completeness heavily depend on the class of drawings that the schema is applied to. Hence we get a detailed picture of how parsers can benefit from the global constraints that are implicit in a class of drawings and up to what limits the class can be extended without losing efficiency.

5.1 A general parsing schema

Parsing schemata (Sikkel, 1997) view parsing algorithms as inference systems. The general parsing schema for LCG derives *parse items* representing partial drawings licensed by a given grammar and sentence. These parse items have the form $s : \langle I, \Omega\rangle$, where s is a *span* (a non-empty subset of the words in the sentence) and I and Ω are bags of edge labels. Each parse item represents the information that the grammar licenses a partial drawing covering the words of the input sentence specified by s; for this drawing to be complete, one still needs to connect its root nodes using incoming edges labelled with the labels in I and outgoing edges labelled with the labels in Ω. A parse item in which Ω is empty is *fully saturated*. An item $s : \langle \emptyset, \emptyset\rangle$ in which s contains all the words in the sentence is *complete*.

The lookup rule The parsing schema contains three rules called LOOKUP, GROUP and PLUG. The LOOKUP rule creates a new parse item with a singleton span for a word w_i in the input sentence:

$$\frac{\langle I, \Omega\,;\Phi\rangle \in Lex(w_i)}{\{i\} : \langle I, \Omega\rangle}\ \textsc{lookup}$$

The combination rules The GROUP and PLUG rules derive new parse items from existing ones. The first rule, GROUP, combines two fully saturated items into a new fully saturated item. The PLUG rule saturates a bag of valencies in a parse item by combining it with another item

accepting these valencies on incoming edges pointing to its root nodes:

$$\frac{s_1 : \langle I_1, \emptyset \rangle \qquad s_2 : \langle I_2, \emptyset \rangle}{s_1 \oplus s_2 : \langle I_1 \cup I_2, \emptyset \rangle}\ \text{GROUP}$$

$$\frac{s_1 : \langle I_1, \Omega \uplus I_2 \rangle \qquad s_2 : \langle I_2, \emptyset \rangle}{s_1 \oplus s_2 : \langle I_1, \Omega \rangle}\ \text{PLUG}$$

The span of a parse item in the conclusion of the GROUP or PLUG rule ($s_1 \oplus s_2$) is the *union* of the spans in the premises (s_1, s_2). The $\oplus$ relation is a subset of the disjoint union relation. On which pairs of spans it is defined depends on the class of drawings that the schema is applied to, e.g. for $\mathcal{D}_1$ it would only be defined on pairs of spans whose union has at most one gap. We regard these restrictions as implicit side conditions of the GROUP and PLUG rule.

Chart-based parsing A concrete parsing algorithm using the general schema would test whether the inferential closure of the three rules contains a complete item. Computing the inferential closure can be done efficiently by using a *chart*, indexed by the spans, to record parse items already derived, and by choosing a control strategy that guarantees that no two items are combined twice.

Alternatively a grammar could be translated into a definite-clause grammar (DCG): each instance of the LOOKUP rule as well as the GROUP and the PLUG rule can be represented by DCG rules. A DCG parser implemented as proposed in (Shieber et al., 1995) will perform the same operations as the chart parser sketched above.

5.2 Completeness

Before we look at the complexity of parsing LCGs in more detail, we first need to ensure that the presented parsing schema is sound and complete, i.e., that all the inferences are valid and that every drawing can be derived with them. While this is easy to show in the general case, chart-based parsing requires a crucial invariant on the parsing rules: all spans derived during parsing must have a uniform representation. More specifically, assume that each span in the premises of a combination rule has at most g gaps and thus can be represented using $2(g+1)$ integer indices (denoting the start and end positions of the $g+1$ intervals that the span consists of). Then the union of two spans must also have at most g gaps. Under this side condition, the general parsing schema is no longer complete: there are drawings whose gap degree is bounded by g that cannot be derived using parse items whose gap degree is bounded by g.

Completeness for well-nested drawings We will now show that for *well-nested drawings* (cf. Section 2.2), the general parsing schema *is* complete even in the presence of the gap invariant. For the proof of this result, we need the concept of the *gap forest* of a well-nested drawing (Bodirsky et al., 2005).

Definition 5.1 Let $(V\,;\lhd,\prec)$ be a well-nested drawing and let $v \in V$ be a node with g gaps. The *gap forest for* v is defined as the ordered forest $\mathfrak{gf}(v) = (S\,;\sqsupset,<)$:

$$\begin{aligned} S &:= \{\{v\}, G_1^v, \ldots, G_g^v\} \cup \{\, \lhd^* w \mid v \lhd w \,\} \\ \sqsupset &:= \{\, (s_1, s_2) \in S \times S \mid C(s_1) \supset s_2 \,\} \\ < &:= \{\, (s_1, s_2) \in S \times S \mid s_1 \prec s_2 \,\} \end{aligned}$$

The elements of S are called *spans*. ⊣

(The notation G_i^v refers to the ith gap in the yield of v.) In a gap forest, sibling spans correspond to disjoint sets whose union has at most g gaps. Sibling spans belonging to the same convex region are called *span groups*.

Lemma 5.2 Let G be an LCG grammar and let $\mathfrak{D}$ be a well-nested drawing on nodes V with gap degree at most g. Then $\mathfrak{D} \vDash G$ implies the existence of a derivation of a parse item $V : \langle I, \emptyset \rangle$ that only involves parse items whose gap degree is bounded by g. □

PROOF Let G be a grammar and let $\mathfrak{D}$ be a well-nested drawing on V such that $\mathfrak{D} \vDash G$. If $V = \{u\}$, then $\langle \emptyset, \emptyset\,;\Phi\rangle \in Lex(\ell(u))$. In this case, the parse item $\{u\} : \langle \emptyset, \emptyset\rangle$ can be derived by one application of the LOOKUP rule. Now assume that $\mathfrak{D}$ consists of a root node u with children v_i, $1 \le i \le k$, where each child v_i is the root of a drawing $\mathfrak{D}_i$. Then

$$\langle \emptyset, \textstyle\bigcup_{1\le i\le k}\ Q_i\,;\Phi\rangle \in Lex(\ell_V(u)), \quad \text{where}$$
$$Q_i = \{\, \pi_i \mid \langle \{\pi_i\}, \Omega_i\,;\Phi_i\rangle \in Lex(\ell_V(v_i)) \,\}\,.$$

$$\begin{aligned}
&\sum_{i=1}^{n} (i^k - (i-1)^k)^2 + (i^k - (i-1)^k)(i-1)^k \\
&= \sum_{i=1}^{n} i^{2k} - 2i^k(i-1)^k + (i-1)^{2k} + i^k(i-1)^k - (i-1)^{2k} \\
&= \sum_{i=1}^{n} i^{2k} - i^k(i-1)^k \quad \leq \quad \sum_{i=1}^{n} i^{2k} - (i-1)^{2k} \quad = \quad n^{2k}
\end{aligned}$$

Figure 6: Complexity estimate for a left-to-right chart parsing strategy. The last equality holds because the sum in question is telescopic.

By induction, we may assume that each of the drawings $\mathfrak{D}_i$ was derived using parse items with gap degree at most g only; in particular, each complete drawing $\mathfrak{D}_i$ corresponds to such a parse item. The drawing $\mathfrak{D}$ then can be derived using the two combination rules, successively combining the parse items for the drawings $\mathfrak{D}_i$ and the item for the root node u (obtainable by the LOOKUP rule).

The interesting part of the proof is to show that the combining operations can be linearised in such a way that the gap degree of the intermediate parse items is bounded by g. We now present such a linearisation, based on a post-order traversal of the gap forest for the node u: In a horizontal phase of the traversal, we combine all parse items corresponding to a span group from left to right, ignoring any gap nodes. There are at most g such nodes in the complete gap forest; therefore, this phase of the traversal maintains the gap invariant. In a vertical phase, we combine the parse items from the preceding horizontal phase with the item corresponding to the parent node in the gap forest in order of their gap degree. Since the gap degree of the final item is bounded by g, this maintains the gap invariant. ∎

5.3 Complexity analysis

We now determine the complexity bounds of an implementation of our schema.

Space complexity To bound the number of parse items stored in the chart, we look at the number of possible values for the variables of a parse item $s : \langle I, \Omega \rangle$. As both I and Ω may represent arbitrary multisets over the edge labels, the number of parse items may be exponential in the size of the grammar. In the case that the drawings under consideration are unrestricted (so that a span s can be an arbitrary set), the number of parse items is also exponential in the length of the input sentence. However, in cases where Lemma 5.2 applies, spans can be represented by $k = 2(g+1)$ integers (cf. Section 5.2). Thus, there will be at most $O(n^k)$ different parse items in the chart.

Time complexity Since the chart-based architecture guarantees that no two parse items are combined twice, the space complexity can be used to bound the time complexity. Of course, if the number of parse items is exponential, the runtime of any algorithm faithfully implementing the general parsing schema will be exponential as well. In what follows, we will ignore the size of the grammar and focus on well-nested drawings with bounded gap degree. How many possibilities of combinations are there for parse items? Counted over the runtime of the complete algorithm, every parse item needs to be combined with every other item, so the time needed for these combinations is $O(n^k) \cdot O(n^k) = O(n^{2k})$.

To see this more clearly, assume that the parser works from left to right. At each position i in the input string, the chart contains i^k different spans. For the combination rule, we need to combine the parse items that we have not yet seen among each other, and with all the items previously present in the chart. Since the number is monotonically increasing, the estimate in Fig. 6 holds.

A refined analysis This $O(n^{2k})$ time estimate is too pessimistic still. To see this, notice that in both of the combination rules, k indices used to represent the spans only occur in the premises: since both the spans in the premises and the span in the conclusion can be represented using k indices each, $2k-k$ cannot 'make it' into the conclusion. As the union operation on spans does not 'forget' about any material, the value of $k/2$ of these indices are determined by other indices in the premises. Thus, a better upper bound for the time complexity for the algorithm is $O(n^{2k-k/2})$. Remembering that $k = 2(g+1)$, we get the following result:

Lemma 5.3 Let $\mathcal{D}$ be a class of well-nested drawings whose gap degree is bounded by g, and let $\mathcal{L}_R$ be a lexical constraint language. Then the universal recognition problem for $(\mathcal{D}, \mathcal{L}_R)$ has complexity $O(2^{|G|}cn^{3g+3})$, where c is the (polynomial) time it takes to check lexical constraints, measured relative to $|G|$ and n. □

For context-free grammars ($g = 0$), this lemma gives the familiar $O(n^3)$ parsing result; for TAGs ($g = 1$), we get a parser that takes time $O(n^6)$. Note that both of these complexities ignore the size of the grammar, and the complexity of the lexical constraints. For LCFGs, however, our parsing framework can be as efficient as e.g. the Earley parser:

Lemma 5.4 The universal recognition problem for totally ordered grammars of type $(\mathcal{D}_0, \mathcal{L}_{\{\prec\}})$ has complexity $O(|G|^2n^3)$. □

PROOF By the previous lemma, we know that $O(2^{|G|}cn^3)$ is an upper bound. The restriction that the valency of each lexical entry are totally ordered implies that we can represent valencies as lists instead of bags. The precedence constraints can be expressed entirely as side conditions on the span variables, hence we can ignore complexity of these constraints. ∎

5.4 The size of the grammar and the complexity of the constraints

The previous section offered insights in how far the model class used by a certain grammar formalism influences the completeness and the complexity with respect to the length of the input sentence. To develop an efficient parser of practical relevance based on our parsing schema, two crucial points are the parsing complexity with respect to the size of the grammar and the complexity of the lexical constraints.

Grammar size is an often neglected factor for the performance of parsing algorithms: a standard sentence of, say, 25 words, is usually several orders of magnitude shorter than a lexicalised grammar. While grammar size thus is significant even for frameworks in which the grammar only contributes linearly or quadratically to the speed of the parsing algorithm (such as context-free grammar), it is definitely an issue in a framework like LCG, where for reasons of expressive power it cannot in general be avoided. It seems then, that it is desirable to complement the chart-based parsing architecture by methods to avoid the worst-case complexity in the size of the grammar whenever possible.

This is where we propose to use constraint propagation: lexical constraints can be used to control the chart-based parser. To give a very simple example: in the presence of order constraints, far from all of the possible combinations of parse items need to be considered when applying the PLUG rule: if an item i has open valencies $\pi_1 \prec \pi_2$, there is no need to try to plug π_2 with an item adjacent to i—any item plugging π_1 precedes any item plugging π_2 in all licensing drawings. How exactly the interaction between constraint propagation and chart parsing is realized and how much a parser can benefit from each single constraint are open questions that we are currently addressing.

6 Conclusion

This paper presented Lexicalised Configuration Grammars (LCGs), a novel framework for the descriptive analysis of natural language. LCG is parametrised by the choice of a class of reference structures (a global constraint), and the choice of a lexical (i.e., local) constraint language used to describe those structures that should be considered grammatical. Translating grammar formalisms into LCG makes it

possible to study these formalisms and their relations from a new perspective, and to experiment with gradual and local alternations of their expressivity and processing complexity. LCGs are expressive enough to generate the scrambling language, a language that cannot be generated by many traditional generative frameworks. The universal recognition problem for LCG is NP-complete; however, a broad class of linguistically relevant LCGs can be parsed in polynomial time.

Future work We plan to continue our research by investigating the potential of the processing framework outlined in Section 5 to combine chart-based and constraint-based processing techniques. Our immediate goal is the implementation of a parser for LCGs that uses constraint propagation to avoid the worst-case complexity of the chart-based parsing algorithm with respect of the size of the grammar. One of the major technical challenges in this is the constraint-based treatment of lexical ambiguity: handling disjunctive information is notorously difficult for constraint propagation. In a second line of work, we will try to relate LCGs to more and more traditional grammar formalisms by defining appropriate LCG theories and grammars and proving the necessary equivalence results.

References

G. Edward Barton. 1985. On the complexity of ID/LP parsing. *Comp. Ling.*, 11(4):205–218.

Tilman Becker, Owen Rambow, and Michael Niv. 1992. The derivational generative power, or, scrambling is beyond lcfrs. Technical Report IRCS-92-38, University of Pennsylvania.

Manuel Bodirsky, Marco Kuhlmann, and Mathias Möhl. 2005. Well-nested drawings as models of syntactic structure. In *10th Conference on Formal Grammar and 9th Meeting on Mathematics of Language*, Edinburgh, Scotland, UK.

Ralph Debusmann, Denys Duchier, and Marco Kuhlmann. 2005. Multi-dimensional graph configuration for natural language processing. In *Constraint Solving and Language Processing*, volume 3438 of *Lecture Notes in Computer Science*, pages 104–120. Springer.

Gerald Gazdar, Ewan Klein, Geoffrey K. Pullum, and Ivan A. Sag. 1985. *Generalized Phrase Structure Grammar*. Havard University Press, Cambrige, MA.

Aravind Joshi and Yves Schabes, 1997. *Handbook of Formal Languages*, volume 3, chapter Tree Adjoining Grammars, pages 69–123. Springer.

Hiroshi Maruyama. 1990. Structural disambiguation with constraint propagation. In *28th Annual Meeting of the Association for Computational Linguistics (ACL 1990)*, pages 31–38, Pittsburgh, Pennsylvania, USA.

James D. McCawley. 1968. Concerning the base component of a transformational grammar. *Foundations of Language*, 4(3):243–269.

Stuart M. Shieber, Yves Schabes, and Fernando C. N. Pereira. 1995. Principles and implementation of deductive parsing. *Journal of Logic Programming*, 24(1&2):3–36.

Klaas Sikkel. 1997. *Parsing Schemata: A Framework for Specification and Analysis of Parsing Algorithms*. Springer-Verlag.

Oliver Suhre. 1999. Computational aspects of a grammar formalism for languages with freer word order. Diploma thesis, Universität Tübingen.