Lexicalised Configuration Grammars

Robert Grabowski, 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. 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.

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 [1], CFGs can be seen as description languages for ordered, labelled trees (Choice 1). More precisely, let $G = (\Sigma, \Pi, R, \pi)$ be a CFG with Σ and Π being the alphabets of terminal and non-terminal symbols, respectively, R the set of rules, and $\pi \in \Pi$ the start symbol. A node u *satisfies* G if either (a) u is a leaf node and is labelled with a terminal symbol, or (b) u is an inner node with successors $u_1, \dots, u_k$ (in that order), R contains a rule $\alpha \to \beta_1 \cdots \beta_k$ (where $\alpha \in \Pi$ and $\beta_i \in \Sigma \cup \Pi$), u is labelled with α, and each successor u_i of u is labelled with β_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 configurations, and no possible choice for the constraint language (Choice 2) can change that. Similarly, in previous work [2], we have identified a class of discontinuous structures that is 'just right' for a descriptive view on Lexicalised Tree Adjoining Grammar (LTAG) [3]. Adopting this class commits an LCG formalism to subsets of those syntactic structures that are obtainable 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 [4] or Constraint Dependency Grammar (CDG) [5]. 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 a word w (*accepted types*) and the number and types of other words that w must connect with to form a complete expression (*required types*). The concept of valency is universal among lexicalised grammar formalisms; it is implemented by non-terminal 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 a word 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 $\alpha \to \beta_1 w \beta_2 \beta_3$ (where $\alpha, \beta_i \in \Pi$ and $w \in \Sigma$) would correspond to the lexical entry

$$\langle \{\alpha\}, \{\beta_1, \beta_2, \beta_3\} \,;\, \beta_1 \prec \iota \wedge \iota \prec \beta_2 \wedge \beta_2 \prec \beta_3 \rangle\,.$$

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 α and successors of types $\beta_1, \beta_2, \beta_3$. The third component of the entry contains the lexical constraints on the valency; for the example entry, the node labelled with w (denoted by ι 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 stratified 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 [6], 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 desireable to be able to express context-free and non-context-free phenomena in the same formalism, investing extra formal and computational resources only in cases where they really are required. We claim that LCG is suitable for such endeavours.

Another reason why we think that LCG is an interesting 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. [7], the question whether a formalism can model scrambling boils down to asking whether it can generate the indexed language

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

where the indices (written as superscripts) match up verbs (vs) with their noun arguments (ns). It has been shown [7] 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.

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

Structure We start our exposition by introducing *labelled drawings* as the universal reference class of structures for LCGs (Section 2). Section 3 presents the stratified 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.

2 Labelled drawings

We introduce LCGs as description languages for (labelled) *drawings* [2], 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 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', $\mathcal{C}(V')$, is the smallest interval containing V'. A *gap* in a set V' is a maximal, non-empty interval in $\mathcal{C}(V') - V'$; the number of gaps in V' is the *gap degree* of V'.

2.2 Drawings

Drawings are forests whose nodes are totally ordered.

Definition 1. *A* drawing *is a relational structure* $(V; \lhd, \prec)$ *in which* $(V; \lhd)$ *forms a forest and* $(V; \prec)$ *forms a total order. If the forest structure underlying a drawing forms a tree, the drawing is called* arborescent.

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 $\triangleleft^* v$; for example, the gap degree of a node v is the gap degree of $\triangleleft^* v$. 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 forest structure but with different gap degrees.

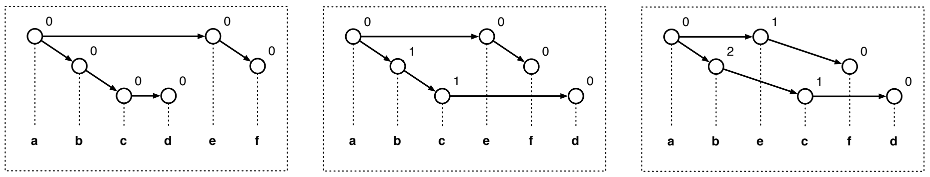


Fig. 1. Drawings in $\mathcal{D}_0$ (projective drawings; left), $\mathcal{D}_1 - \mathcal{D}_0$ (gap degree 1; middle) and $\mathcal{D}_2 - \mathcal{D}_1$ (gap degree 2; right). An integer at a node states that node's gap degree.

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 [2].

Definition 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: one from its nodes to an alphabet Σ of *node labels* and a second one from its edges to an alphabet Π of *edge labels*. Since it will always be clear from the context whether we mean the node labelling or the edge labelling function, we will use the symbol ℓ for both: for any node v, $\ell(v)$ refers to the node label associated to v; for any edge (u, v), $\ell(u, v)$ refers to the associated edge label. 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:

$$\triangleleft_\pi \;:=\; \{\, (u, v) \in V \times V \mid u \triangleleft v \text{ and } \ell(u, v) = \pi \,\}\,.$$

To reduce the complexity of our presentation, we assume the existence of a special edge label ι called 'self', distinct from all other labels, and define $\lhd_\iota := \mathrm{Id}$.

The *projection* of a labelled drawing $\mathfrak{D}$, $\mathit{proj}(\mathfrak{D})$, is the string obtained by concatenating the node labels of the drawing in the order of their corresponding nodes; this is in analogy to the notion of the frontier of an ordered labelled tree.

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 $\ell(u)$ 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* $\mathit{ar}(R)$, 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. *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 formulae ϕ of the following form:*

$$\phi \ ::= \ \mathbf{t} \mid R(\pi_1, \dots, \pi_k) \mid \phi_1 \wedge \phi_2 \,, \quad \textit{where } R \in \mathcal{R},\ \mathit{ar}(R) = k, \textit{ and } \pi_i \in \Pi$$

We write $\mathcal{L}_\mathcal{R}$ for the class of all lexical constraint languages with signature $\mathcal{R}$.

The literal $\mathbf{t}$ is read as 'true'. We call literals of the form $R(\pi_1, \dots, \pi_k)$ *relational constraints*. Binary relational constraints will be written using infix notation, so the notation $\pi_1 \, R \, \pi_2$ will stand for $R(\pi_1, \pi_2)$.

Semantics The satisfaction 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. For formulae of the form $\mathbf{t}$ and $\phi_1 \wedge \phi_2$, the definition of the satisfaction relation is the same for all lexical constraint languages:

$$\begin{aligned} \mathfrak{D}, u &\models \mathbf{t} && \text{always} \\ \mathfrak{D}, u &\models \phi_1 \wedge \phi_2 && \text{iff} \quad \mathfrak{D}, u \models \phi_1 \text{ and } \mathfrak{D}, u \models \phi_2 \end{aligned}$$

Satisfiability of relational constraints must be defined individually for a specific language. However, there are two restrictions on the possible definitions; these restrictions define lexical constraint languages in the wider sense of the term: a definition of the satisfiability relation $\mathfrak{D}, u \models R(\pi_1, \ldots, \pi_k)$ may only refer to the labelled successor relations $\{\triangleleft_{\pi_1}, \ldots, \triangleleft_{\pi_k}\}$,[2] and the question whether the defining condition applies must be decidable in time polynomial in the number of nodes in $\mathfrak{D}$. LCG does not impose any further restrictions; it allows for defining arbitrary constraint languages for labelled drawings, as long as the constraints meet the above criteria.

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.

Definition 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 $\Sigma \to \mathfrak{P}(LE_{\mathcal{R}}(\Pi))$.*

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 5. *A lexical entry describes a node in a drawing. It is a triple*

$$\langle I, \Omega\,; \phi \rangle \quad \in \quad \mathfrak{B}(\Pi) \times \mathfrak{B}(\Pi) \times \mathcal{L}_{\mathcal{R}}(\Pi) =: LE_{\mathcal{R}}(\Pi)\,,$$

where the bags I and Ω contain edge labels, and ϕ is a lexical constraint. 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*

$$\text{for all } \pi \in \Pi,\ |(\triangleleft_\pi)^{-1}u| = I(\pi) \text{ and } |(\triangleleft_\pi)u| = \Omega(\pi)\,, \quad \text{and} \quad \mathfrak{D}, u \models \phi\,.$$

The satisfaction property of a node can be lifted to the whole drawing:

Definition 6. *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 $\langle I, \Omega\,; \phi \rangle \in Lex(\ell(u))$ such that u satisfies $\langle I, \Omega\,; \phi \rangle$. $\mathfrak{D}$* satisfies *a grammar G of type T, written $\mathfrak{D} \models G$, iff every node $u \in \mathfrak{D}$ satisfies the lexicon of the grammar.*

3.3 Sample languages

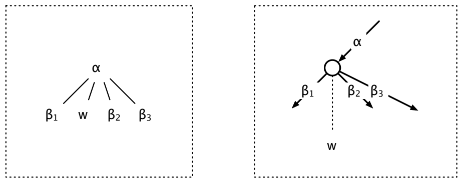
To provide an intuition for the formal concepts defined in the previous two sections, we will now translate three grammar formalisms into LCG theories. We start by adapting our previous encoding of LCFG to the new formal concepts.

[2] The definition may refer to arbitrary *unlabelled* relations in $\mathfrak{D}$.

Lexicalised Context-Free Grammars As already mentioned in the introduction, lexicalised context-free rules like $\alpha \rightarrow \beta_1 w \beta_2 \beta_3$ can be seen as local well-formedness conditions on node-labelled, ordered trees (see Fig. 2). 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 relational constraint $\prec$ 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.

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

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 α 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.



$$w : \langle \{\alpha\}, \{\beta_1, \beta_2, \beta_3\} \; ; \; \beta_1 \prec \iota \wedge \iota \prec \beta_2 \wedge \beta_2 \prec \beta_3 \rangle$$

Fig. 2. Encoding Lexicalised Context Free Grammars

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 $(\mathcal{D}_0, \emptyset)$, which is equivalent to the class of (lexicalised) unordered context-free grammars.

The scrambling language The following grammar derives drawings whose projections 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 $G_{\mathsf{SCR}} := (\{n, v\}, \{n, v\}, \mathit{Lex})$, where the lexicon Lex contains the entry $\langle \{n\}, \emptyset \,;\, \mathbf{t} \rangle$ for n and the following entries for v:

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

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}.)

$$\mathfrak{D}, u \models \pi_1 \prec \pi_2 \qquad \text{iff} \quad (\lhd_{\pi_1} \circ \lhd^*)u \times (\lhd_{\pi_2} \circ \lhd^*)u \subseteq \prec$$
$$\mathfrak{D}, u \models \pi_1 \bowtie \pi_2 \qquad \text{iff} \quad (\lhd_{\pi_1} \circ \lhd^* \cup \lhd_{\pi_2} \circ \lhd^*)u \text{ is convex}$$

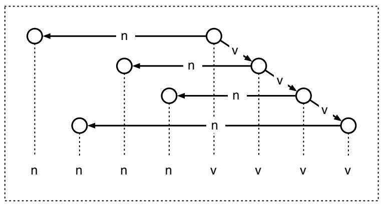


Fig. 3. Lexical constraint language and sample drawing for SCR

Linear Specification Language Suhre's LSL formalism [6] allows to generate languages with a free word order. It is inspired by ID/LP parsing [4], 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 $\mathcal{L}_{\text{LSL}}$ shown in Fig. 4. We define the following abbreviations:

$$\diamond_\pi := \lhd_\pi \circ \lhd^*, \quad \diamond_\iota := \text{Id}, \quad \diamond_\bullet := \lhd^* \,.$$

The last clause in the definition of the satisfiability relation in Fig. 4 corresponds to an isolation constraint applied to the left hand side of an LSL rule.

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

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

Fig. 4. Suhre's Linear Specification Language

string membership 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 general string membership problem for the most general LCG theory is NP-complete.

4.1 The general string membership problem

Definition 7. *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* general string membership problem *for G and s, written (G, s), is the problem to decide whether the following set is non-empty:*

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

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

Lemma 1. *The general string membership problem for $(\mathcal{D}, \mathcal{L}_{\emptyset})$ is NP-hard.*

Proof. We will present a polynomial reduction of HAMILTON PATH to the general string membership 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

$$\begin{array}{rcl}
\Sigma_H, \Pi_H & := & V \\
\mathit{start}(v) & := & \{\, \langle \emptyset, \{v'\} \,;\, \mathbf{t} \rangle \mid v \to v' \in H \,\} \\
\mathit{inner}(v) & := & \{\, \langle \{v\}, \{v'\} \,;\, \mathbf{t} \rangle \mid v \to v' \in H \,\} \\
\mathit{end}(v) & := & \{\, \langle \{v\}, \emptyset \,;\, \mathbf{t} \rangle \mid v \to v' \in H \,\} \\
\mathit{Lex}_H & := & \{\, v \mapsto \mathit{start}(v) \cup \mathit{inner}(v) \cup \mathit{end}(v) \mid v \in V \,\} \\
G_H & := & (\Sigma_H, \Pi_H, \mathit{Lex}_H)
\end{array}$$

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 $\mathit{start}(v)$, $\mathit{end}(v)$, or $\mathit{inner}(v)$, depending on the position of v in the Hamilton Path. 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 Hamilton Path in H is marked by solid edges. The depicted drawing satisfies the following lexicon Lex_H. (The lexical entry satisfied at each node is underlined.)

$$\begin{aligned}
1 &\mapsto \{\langle\emptyset,\{3\}\,;\mathbf{t}\rangle,\langle\emptyset,\{4\}\,;\mathbf{t}\rangle,\langle\{1\},\{3\}\,;\mathbf{t}\rangle,\underline{\langle\{1\},\{4\}\,;\mathbf{t}\rangle},\langle\{1\},\emptyset\,;\mathbf{t}\rangle\}\\
2 &\mapsto \{\underline{\langle\emptyset,\{1\}\,;\mathbf{t}\rangle},\langle\emptyset,\{4\}\,;\mathbf{t}\rangle,\langle\{2\},\{1\}\,;\mathbf{t}\rangle,\langle\{2\},\{4\}\,;\mathbf{t}\rangle,\langle\{2\},\emptyset\,;\mathbf{t}\rangle\}\\
3 &\mapsto \{\underline{\langle\{3\},\emptyset\,;\mathbf{t}\rangle}\}\\
4 &\mapsto \{\langle\emptyset,\{3\}\,;\mathbf{t}\rangle,\underline{\langle\{4\},\{3\}\,;\mathbf{t}\rangle},\langle\{4\},\emptyset\,;\mathbf{t}\rangle\}
\end{aligned}$$

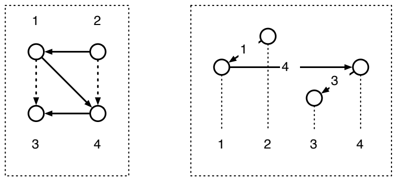


Fig. 5. An input graph H for Hamilton Path and a drawing licensing Lex_H

4.2 The fixed string membership problem

The fixed string membership problem asks the same question as the general problem, but the grammar is not considered part of the input. This fact 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 word problem needs to assume one fixed grammar for every input string. The proof of the following result is omitted due to space limitations:

Lemma 2. *The fixed membership problem for* $(\mathcal{D},\mathcal{L}_\emptyset)$ *is polynomial.*

It would seem desirable to have a framework in which extending the signature of the constraint language may only *reduce* the complexity of the membership 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* string membership 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 [8] 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 [8] 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:

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

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:

$$\frac{s_1 : \langle I_1, \emptyset \rangle \qquad s_2 : \langle I_2, \emptyset \rangle}{s_1 \oplus s_2 : \langle I_1 \cup I_2, \emptyset \rangle}\ \textsc{group} \qquad \frac{s_1 : \langle I_1, \Omega \uplus I_2 \rangle \qquad s_2 : \langle I_2, \emptyset \rangle}{s_1 \oplus s_2 : \langle I_1, \Omega \rangle}\ \textsc{plug}$$

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.

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 [9] 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 [2].

Definition 8. *Let* $(V; \triangleleft, \prec)$ *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* $\mathfrak{gf}(v) = (S; \sqsupset, <)$*:*

$$\begin{aligned} S &:= \{\{v\}, G_1(v), \dots, G_g(v)\} \cup \{\, \triangleleft^* w \mid v \triangleleft w \,\} \\ \sqsupset &:= \textit{transitive reduction of } \{\, (s_1, s_2) \in S \times S \mid \mathcal{C}(s_1) \supset s_2 \,\} \\ < &:= \{\, (s_1, s_2) \in S \times S \mid \forall v_1 \in s_1 \colon \forall v_2 \in s_2 \colon v_1 \prec v_2 \,\} \end{aligned}$$

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 3. *Let* G *be an* LCG *grammar and let* $\mathfrak{D}$ *be a well-nested arborescent drawing on nodes* V *with gap degree at most* g*. Then* $\mathfrak{D} \models G$ *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 arborescent drawing on V such that $\mathfrak{D} \models G$. If $V = \{u\}$, then $\langle \emptyset, \emptyset \,;\, \phi \rangle \in \mathit{Lex}(\ell(u))$. 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, $1 \leq i \leq k$, where each child v_i is the root of an arborescent drawing $\mathfrak{D}_i$. Then

$$\langle \emptyset, P \,;\, \phi \rangle \in \mathit{Lex}(\ell(u)), \quad \text{where} \quad P = \cup_{1 \leq i \leq k} \; \{\, \pi_i \mid \langle \{\pi_i\}, \Omega_i \,;\, \phi_i \rangle \in \mathit{Lex}(\ell(v_i)) \,\} \,.$$

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 will 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 gap 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 strategy 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 3 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 $O(n^k) \cdot O(n^k) = O(n^{2k})$.

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

Lemma 4. *Let $\mathcal{D}$ be a class of well-nested drawings whose gap degree is bounded by g, and let $\mathcal{L}_{\mathcal{R}}$ be a lexical constraint language. Then the general string membership problem for $(\mathcal{D}, \mathcal{L}_{\mathcal{R}})$ has complexity $O(2^{|G|}n^{3g+3})$.*

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 $O(n^6)$. Notice that both of these complexities ignore the size of the grammar. For LCFGs, however, our parsing framework can be as efficient as e.g. the Earley parser:

Lemma 5. *The general string membership 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 $O(2^{|G|}n^3)$ 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.

5.4 The size of the grammar

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 however, a crucial point is the complexity with respect to the size of the grammar. 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 it 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 stratified with respect to two parameters: 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 general string membership 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

1. McCawley, J.D.: Concerning the base component of a transformational grammar. Foundations of Language **4** (1968) 243–269
2. Bodirsky, M., Kuhlmann, M., Möhl, M.: Well-nested drawings as models of syntactic structure. In: 10th Conference on Formal Grammar and 9th Meeting on Mathematics of Language, Edinburgh, Scotland, UK (2005)
3. Joshi, A., Schabes, Y.: Tree Adjoining Grammars. In: Handbook of Formal Languages. Volume 3. Springer (1997) 69–123
4. Gazdar, G., Klein, E., Pullum, G.K., Sag, I.A.: Generalized Phrase Structure Grammar. Havard University Press, Cambrige, MA (1985)
5. Maruyama, H.: Structural disambiguation with constraint propagation. In: 28th Annual Meeting of the Association for Computational Linguistics (ACL 1990), Pittsburgh, Pennsylvania, USA (1990) 31–38
6. Suhre, O.: Computational aspects of a grammar formalism for languages with freer word order. Diploma thesis, Universität Tübingen (1999)
7. Becker, T., Rambow, O., Niv, M.: The derivational generative power, or, scrambling is beyond lcfrs. Technical Report IRCS-92-38, University of Pennsylvania (1992)
8. Sikkel, K.: Parsing Schemata: A Framework for Specification and Analysis of Parsing Algorithms. Springer-Verlag (1997)
9. Shieber, S.M., Schabes, Y., Pereira, F.C.N.: Principles and implementation of deductive parsing. Journal of Logic Programming **24** (1995) 3–36