Universität des Saarlandes
Fachrichtung 6.2 – Informatik

Diplomarbeit

Computational Aspects of Non-Projective Dependency Grammars

Robert Grabowski

April 2006

Angefertigt unter der Leitung von Prof. Dr. Gert Smolka
Betreuung durch Marco Kuhlmann

Erklärung

Hiermit erkläre ich, Robert Grabowski, an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Saarbrücken, den 13. April 2006

Abstract

Lexicalised Configuration Grammars (LCG) is a declarative framework for non-projective dependency grammars. Lexical entries in LCG are local well-formedness conditions for drawings (trees equipped with a total order). The framework is powerful enough to encode a large number of existing grammar formalisms declaratively.

The expressivity of LCG, however, comes at the cost of NP-complete word problems and parsing complexities that are hard to analyse. This diploma thesis shows that by borrowing efficient parsing concepts from generative formalisms, the computational factors that contribute to complexity can be formalised and isolated. A variety of possibilities to restrict these factors is elaborated in order to reduce complexity to polynomial time.

Moreover, a precise analysis is presented for a specifc LCG grammar formalism with precedence and discontinuity constraints. By combining the complexity results, that formalism is proven to allow for declarative definitions of mildly context-sensitive languages, which are believed to sufficiently account for many natural languages.

Acknowledgements

I would like to thank Professor Dr. Gert Smolka for proposing this subject, and for giving me the freedom to explore it in detail.

I also want to thank Marco Kuhlmann, my supervisor, for his comprehensive support during the entire work. Large parts of this thesis are based on our regular discussions; he always brought in new, motivating ideas when I found myself stuck in a dead-end formalisation. Also, I would like to acknowledge his help with typesetting in LaTeX, and his ability to spot even microtypographic issues.

I am grateful to him, Mathias Möhl and Manuel Bodirsky for their work on well-nested drawings; their results gave me a better understanding of the essence of non-projective structures.

Furthermore, Ralph Debusmann is thanked for extending once again the LaTeX package for the visualisation of drawings. Probably to his relief, I found out how to modify the package on my own.

I am generally grateful to the people at Programming Systems Lab, whose feedback on my talks positively influenced my direction of exploration.

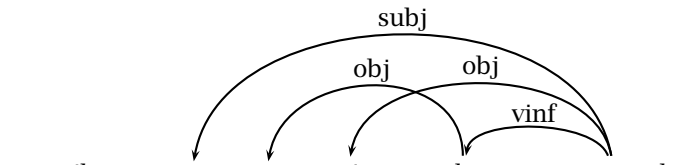
Finally, I want to thank my parents and Marianne for their support, patience, and encouragement during the last weeks and months.

Contents

Chapter 1

Introduction

Dependency structures are models that describe the syntactic structure of a sentence by stating *dependency relations* between parts of the sentence. According to its role in a relation, a part is called *head* or *dependent*. The following example shows a possible dependency analysis for a German subordinate clause, describing the linear precedence of the words and the syntactical dominance between them:[1]



. . . because he promised me to read it

Constraint-based dependency grammars allow for descriptive specifications of dependency structures. Rather than generating them with rules and operations, the structures are described declaratively using local well-formedness conditions: for a given word w, they specify for example the allowed types (categories) of w, the types of the dependents of w, and the possible orders in which the dependents may appear in the sentence.

Though there exists a number of powerful dependency formalisms that are able to describe a rich variety of natural language phenomena by dependency relations, such as XDG [5], rather little attention has been paid to the impact of their expressivity on parsing performance. The problem is that the time complexity of these constraint-based parsers is often rather difficult to analyse, since the formal computational parameters are usually far from being obvious.

This stands in contrast to the field of generative grammar formalisms, where complexity aspects are generally well understood. For many of these rule-based formalisms, efficient parsers have been developed and thoroughly analysed. One of the reasons for their efficiency is that they often make use of a so-called *chart*, in which intermediate parse results are stored, to avoid computational duplication. An interesting question is then to what extent this technique can be transferred to constraint-based dependency grammars.

[1] To simplify matters, the words *'zu'* and *'lesen'* are regarded as one entity.

Previous work A milestone complexity result has already been achieved at least for grammars describing *projective* dependency structures. A structure is projective if each head with all of its dependents (immediate or indirect) forms a continuous substring of the sentence, such that the structure can be drawn without crossing edges. Projective structures therefore correspond to the classical linguistic notion of an ordered syntactic tree. As shown by Gaifman [7], projective dependency grammars are strongly equivalent to lexicalised context-free grammars, and thus recognisable with an $O(n^3)$ parser.

However, there are languages for which projective structures and hence context-free grammars are considered insufficient or at least inadequate [15]. For example, the German scrambling phenomena might require a non-projective analysis as shown in the structure above: the head *'zu lesen'* and its only dependent *'es'* do not cover a continuous range of the sentence.

Contents of this thesis This thesis explores possibilities to extend Gaifman's complexity result to non-projective declarative dependency grammars. As a reference dependency framework, we use Lexicalised Configuration Grammars (LCG) [9], a descriptive framework for the syntactic analysis of natural language.

The expressivity of LCG allows to encode a large variety of existing grammar formalisms in the framework. Nevertheless, it contains by design several restrictions that make it suitable for chart-based parsing. Therefore, we present an abstract algorithm for a general LCG chart parser in form of a *parsing schema* [17], analyse its complexity, and formalise the computational parameters. This is followed by an examination of how particular properties of LCG grammars can be exploited to restrict these parameters such that the efficiency of the parser improves.

The main challenge is that parsing efficiency highly depends on the ability to make use of the constraints defined in the grammar during parsing. The more the constraints can be incorporated into the actual parsing algorithm, the better does the parser satisfy the constraints automatically, and the less 'useless' structures have to be produced using a generate-and-test mechanism.

A recurring formalism in this work is the *Linear Specification Language* (LSL) [18], which generates possibly non-projective structures by context-free rules with linear constraints. Since these rules can be expressed as lexical entries in LCG, we will compare and verify the observations made in LCG with the complexity results for LSL.

Another important topic is *Tree Adjoining Grammars* (TAG) [12]. We use recent insights concerning the structure of TAG derivation trees [14] to encode the formalism as an LCG theory, and to outline a parser that has the usual $O(n^6)$ TAG parsing complexity. Tree Adjoining Languages also motivate to focus on mildly context-sensitive languages, a class of languages that is believed to closely match the expressivity of many natural languages. With the previous results, the notion of mild context-sensitivity can be easily transferred to dependency grammars.

Structure The thesis is structured as follows:

The following chapter formally introduces the LCG framework and its dependency structures, *labelled drawings*. Moreover, it provides several examples for LCG encodings of well-known grammar formalisms.

Chapter 3 approaches the computational properties of the framework by determining the complexity class of specific fragments of LCG.

Chapter 4 motivates and presents a schema for a chart-based LCG parser and analyses its complexity. The computational factors are isolated, and various general possibilities are explored to restrict these factors in order to improve parsing performance.

Chapter 5 presents specific LCG formalisms with precedence and discontinuity constraints, and shows how these constraints can be integrated into the parser to achieve a polynomial parsing complexity.

Chapter 6 provides a characterisation of mildly context-sensitive dependency languages, and shows that the formalisms presented in the previous chapter allows to define mildly context-sensitive LCG grammars.

Main contributions The main contributions of this thesis are

- a formalisation of the Lexicalised Configuration Grammars framework and its dependency structures,
- the reformulation of the proof that the fixed recognition problem for LSL is NP-complete,
- the analysis of the factors that contribute to parsing complexity, and the characterisation of a class of LCG grammars with a polynomial fixed recognition problem,
- the concept of linearisations of valency parts and discontinuities, and a corresponding LCG formalism that allows to encode (fragments of) LSL and TAG, and
- the formalisation of a class of mildly context-sensitive dependency grammars.

Chapter 2

Lexicalised Configuration Grammars

This chapter gives a short presentation of the Lexicalised Configuration Grammars (LCG) dependency framework. The first part defines labelled drawings, the dependency structures used in LCG. In the second part, the actual framework is formalised, and the notions of theories and grammars are introduced. The last part gives some examples for LCG encodings of well-known grammar formalisms.

The framework has been initially developed as joint work with Marco Kuhlmann and Mathias Möhl [9].

2.1 Labelled drawings

This section introduces *labelled drawings* [3], which are a class of relational structures representing two essential syntactic dimensions: dominance structure and word order. Dominance structure captures the idea that a natural language expression (head) can govern a set of smaller expressions (dependents), while word order concerns the possible linearisations of syntactic material. This section presents the basic terminology for drawings and cites some previous results.

Set notions The *powerset* over S, written $\mathfrak{P}(S)$, is the set of all subsets of S. A *bag* or *multiset* over a base set S is a set that can contain elements in S multiple times. More formally, it is a function in $S \to \mathbb{N}_0$ that assigns every 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 a base set S is a set that either contains exactly one element of S, or is empty. The set of all options over S is defined as follows:

$$\mathfrak{O}(S) := \{\, \{s\} \mid s \in S \,\} \cup \{\emptyset\}$$

2.1.1 Relational structures

A *relational structure* consists of a non-empty, finite set V of *nodes* and a number of relations on V. In the context of drawings, only binary relations on the nodes are considered, for which the standard terminology and notations are used. In particular, given a relation R, the notation R^+ refers to the transitive closure, and R^* to the reflexive-transitive closure of R. Furthermore, Ru stands for the relational image of a node u under R:

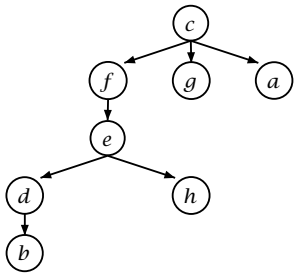


Figure 2.1: A sample tree.

it is the set of all nodes v such that $(u, v) \in R$. We will use the infix notation uRv for $(u, v) \in R$. The *identity relation* is defined as $\mathrm{Id} := \{ (v, v) \mid v \in V \}$.

Two types of relational structures are particularly important for the representation of syntactic configurations: *trees* and *total orders*.

Trees

A relational structure $(V; \lhd)$ is a *forest* if $\lhd$ is an immediate dominance relation, i.e. if $\lhd$ is acyclic and for every node v in V, there is at most one node u such that $u \lhd v$. In other words, every node in a forest has at most one $\lhd$-predecessor. Nodes without a $\lhd$-predecessor are called *roots*. A *tree* is a forest with exactly one root. For a node v, the set $\lhd^* v$ is called the *yield* of v, i.e. the set of all nodes that are reachable from v by the $\lhd$ relation, including v itself. The yield of a node v, together with the immediate dominance relation restricted to the nodes in the yield, forms a subtree rooted at v.

Two nodes $v_1, v_2 \in V$ are *disjoint*, written $v_1 \perp v_2$, if their yields are disjoint sets. A node v_1 *dominates* another node v_2 if $v_1 \lhd^+ v_2$. Note that any two nodes $v_1, v_2 \in V$ are in exactly one of these four relations: $v_1 = v_2$, $v_1 \lhd^+ v_2$, $v_2 \lhd^+ v_1$ or $v_1 \perp v_2$.

For example, consider the tree shown in Figure 2.1, consisting of the set of nodes $V = \{a, b, c, d, e, f, g, h\}$. The immediate dominance relation $\lhd$ is visualised by the directed edges between the nodes. The root of the tree is c, and the set of its immediate successors is $\lhd c = \{a, f, g\}$. The node f, for example, dominates the node d: $f \lhd^+ d$. The yields of the nodes e and g, $\lhd^* e = \{b, d, e, h\}$ and $\lhd^* g = \{g\}$, are disjoint sets, hence it holds $e \perp g$.

Total orders

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 $v_2 \prec v_1$. We define the relation $\preceq$ such that $v_1 \preceq v_2$ if and only if $v_1 \prec v_2$ or $v_1 = v_2$.

Given a total order $(V; \prec)$, the *interval* between two nodes $v_1, v_2 \in V$ is the set of all nodes v such that $v_1 \preceq v \preceq v_2$. A subset $V' \subseteq V$ is *convex* if it is an interval. The *cover* of V', written $C(V')$, is the smallest interval containing V'.

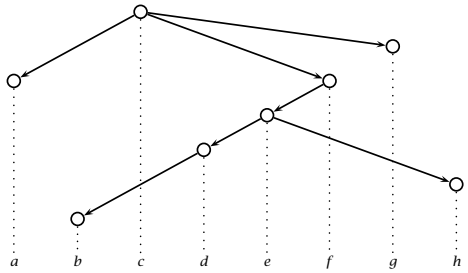


Figure 2.2: A sample drawing.

A subset V' that is not convex contains *holes*, which are nodes in the cover of V' that are not in V' itself. A *gap* in V' is a maximal, non-empty interval in the set of holes; the number of gaps in V' is the *gap degree* of V', written $\mathsf{gd}(V')$.

The gaps in V' partition the set V' into $\mathsf{gd}(V') + 1$ disjoint *parts*:

$$V' \quad = \quad [V']^0 \uplus [V']^1 \uplus \cdots \uplus [V']^{\mathsf{gd}(V')}$$

such that each part $[V']^i$ is a maximal, non-empty interval in V'. Parts are indexed from left to right with respect to $\prec$, starting with zero: for all i, j with $0 \leq i < j \leq \mathsf{gd}(V')$, $[V']^i \times [V']^j \subseteq \prec$. For $i < 0$ or $i > \mathsf{gd}(V')$, we define $[V']^i := \emptyset$.

For example, we consider the relational structure $(\{a, b, c, d, e, f, g, h\}; \prec)$, where $\prec$ is the alphabetical order, and a subset $V' = \{b, d, e, h\}$. The cover of that subset is $C(V') = \{b, c, d, e, f, g, h\}$. There are two gaps in V', $\{c\}$ and $\{f, g\}$. The gap degree $\mathsf{gd}(V')$ is therefore 2. The gaps partition V' into three parts: $[V']^0 = \{b\}$, $[V']^1 = \{d, e\}$ and $[V']^2 = \{h\}$. Note that a set is convex iff it contains no gaps, or, in other words, iff it consists of exactly one part.

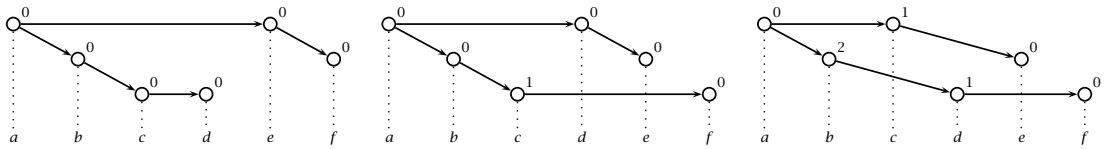
2.1.2 Drawings

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. The set of all drawings is denoted as $\mathcal{D}_*$. ⊣

Note that drawings differ from ordered trees: in an ordered tree, only sibling nodes are ordered; in drawings, the order is total for *all* of the nodes.

As an example, Figure 2.2 shows a drawing whose dominance relation forms the same tree and whose nodes are in the same total order as in the previous examples. To visualise the order, the node names are written at the bottom in one line, and connected to the nodes using vertical *projection lines*. Note that the yield of the node e, $\lhd^* e = \{b, d, e, h\}$, is the subset discussed in the example for total orders.



(a) gap degree 0 (b) gap degree 1 (c) gap degree 2

Figure 2.3: Drawings in $\mathcal{D}_0$ (left), $\mathcal{D}_1 \setminus \mathcal{D}_0$ (middle) and $\mathcal{D}_2 \setminus \mathcal{D}_1$ (right). An integer at a node states that node's gap degree.

The gap degree The notions of cover, part, gap, and gap degree can be applied to nodes in a drawing by identifying a node v with its yield $\lhd^* v$. In particular, the gap degree of a node v is the gap degree of its yield $\lhd^* v$. 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 class $\mathcal{D}_0$ contains exactly the *projective* drawings mentioned in the introduction: the yields in this class of drawings cannot contain a gap (discontinuity). In contrast, drawings with a gap degree of at least 1 are *non-projective*.

Figure 2.3 shows three drawings of the same tree structure but with different gap degrees. In the left drawing, all nodes have a gap degree of 0, because the yield of each node is convex. In the middle drawing, the yields of the nodes b and c both contain one gap consisting of the nodes e and f. In the right drawing, node b has the maximal number of gaps (2), one of them containing the node e, the other one containing f.

Well-nestedness The notion of gap degree provides 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 them is *well-nestedness*, which constrains the possible relations between gaps [3].

Definition 2.2 Let $\mathfrak{D}$ be a drawing. The yields Y_1 and Y_2 of two disjoint nodes in $\mathfrak{D}$ *interleave* if there are nodes $l_1, r_1 \in Y_1$ and $l_2, r_2 \in Y_2$ such that $l_1 \prec l_2 \prec r_1 \prec r_2$, or, vice versa, $l_2 \prec l_1 \prec r_2 \prec r_1$. The drawing $\mathfrak{D}$ is called *well-nested* if it does not contain any interleaving yields. ⊣

In other words, a drawing is well-nested if the covers of any two *disjoint* trees T_1, T_2 within the drawing do not overlap. This is the case if T_1 is either entirely within one gap of T_2, or vice versa, or both trees appear side by side (i.e. their covers are disjoint). In Figure 2.3, the first drawing is well-nested, because it is projective: all yields in a projective drawing are convex, therefore two disjoint yields cannot interleave. The second drawing is also well-nested; the subtree consisting of the nodes e, f is contained entirely within the gap of the trees rooted at c and at b. The third drawing, however, contains at least two interleaving trees: one of them containing the nodes b, c, d, the other one consisting of e, f.

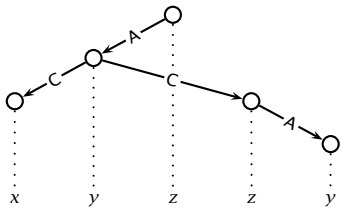


Figure 2.4: A labelled drawing, marked with node labels from $\{x, y, z\}$ and edge labels from $\{A, B, C\}$.

The notation $\mathcal{D}_{wn}$ is used to refer to the class of all well-nested drawings. The well-nestedness property is generally independent from the gap degree of a drawing. An exception are projective drawings, which are always well-nested, as they do not contain any interleaving trees.

2.1.3 Labelled drawings

A labelled drawing is a drawing equipped with two total functions that label all nodes and edges.

Definition 2.3 Let Σ and Π be alphabets of *node labels* and *edge labels*, respectively. A *labelled drawing* $\mathfrak{D}$ is a quintuple $(V; \lhd, \prec, \ell_V, \ell_E)$, where $(V; \lhd, \prec)$ forms a drawing, $\ell_V \in V \to \Sigma$ is a function marking all nodes with labels in Σ, and $\ell_E \in \lhd \to \Pi$ is a function marking all edges with labels in 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:

$$\mathcal{D}_{\Sigma,\Pi} := \{ (V; \lhd, \prec, \ell_V, \ell_E) \mid (V; \lhd, \prec) \in \mathcal{D} \text{ and } \ell_V \in V \to \Sigma \text{ and } \ell_E \in \lhd \to \Pi \}$$

As an example, Figure 2.4 shows a drawing from the class $\mathcal{D}_{\{x,y,z\},\{A,B,C\}}$.
In labelled drawings, *labelled successor relations* can be defined as follows:

$$\lhd_\pi := \{ (u, v) \in V \times V \mid u \lhd v \text{ and } \ell_E(u, v) = \pi \}$$

The term $\lhd_\pi u$ refers to all immediate successors of u that are reachable via an edge marked with π. There can be more than one such an edge; the labelled successor relation does not allow to distinguish the different π-successors.

To reduce the complexity of the presentation, we introduce the *accessibility relation* as an abbreviation for the set of nodes reachable from a π-successor:

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

The *projection* of a labelled drawing $\mathfrak{D}$, $\mathsf{proj}(\mathfrak{D})$, 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. Given such a string, $\mathsf{pos}(i)$ refers to the node label at position i in the string. For example, the drawing in Figure 2.4 has the projection $\mathsf{proj}(\mathbb{D}) = xyzzy$, and the node label at the fifth position is $\mathsf{pos}(5) = y$.

2.2 The LCG framework

The Lexicalised Configuration Grammars framework uses labelled drawings as models of syntactic structure. Their labels get a syntactic interpretation: The projection of a labelled drawing is the *sentence* it represents; its node labels are the *words* of the sentence. Each node therefore stands for the word it is labelled with. Furthermore, the incoming edge label of a node is the (syntactic) *type* of the word the node represents, and the outgoing edge labels are the types of its dependents.

Within LCG, we make a distinction between *theories* and *grammars*. They declaratively describe the set of allowed syntactical structures on different levels.

An LCG theory defines a class of drawings, e.g. the class of well-nested drawings, thus imposing global constraints on the licensed structures. Furthermore, it defines an abstract class of lexical constraint languages, used to express local well-formedness conditions on drawings. As we will see, a theory in our framework corresponds to a 'grammar formalism' in the classical sense of the word.

An LCG grammar adopts and instantiates a theory: It labels the allowed class of drawings by choosing concrete alphabets Σ and Π for the node labels (words) and edge labels (types). Also, it defines a lexicon containing, for each word, lexical constraints for a node representing the word. A drawing is licensed by a grammar if each node fulfils the conditions given in the lexicon.

2.2.1 Lexical constraint languages

For a set of edge labels Π, a *lexical constraint language* defines lexical constraints that express local conditions on drawings with edge labels from Π. The definitions of satisfiability are given with respect to a specific node in a given drawing. For example, for a node u in a drawing, a lexical constraint between two labels A and B might impose a restriction on the nodes reachable from u by the labelled successor relations $\lhd_A$ and $\lhd_B$.

An LCG theory defines a *class of lexical constraint languages*, providing an abstract syntax and semantics for the constraints. In an LCG grammar, these abstract constraints are then instantiated to a specific set of edge labels, resulting in a concrete lexical constraint language.

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* $\mathsf{ar}(R)$, forms the

signature of the language. We will leave the arity function implicit, and use the letter $\mathcal{R}$ to refer to signatures.

Definition 2.4 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 $R \in \mathcal{R}$, $\mathsf{ar}(R) = k$, 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. Note that $\mathcal{L}_{\emptyset}$ is the class containing the language without 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. We write $\mathfrak{D}, u \vDash \phi$ if the node u in $\mathfrak{D}$ satisfies ϕ.

The LCG framework imposes only one restriction on the definition of the satisfiability relations: the question whether a constraint holds must be decidable in time polynomial in the number of nodes in $\mathfrak{D}$. As long as this requirement is met, the framework allows arbitrary definitions of constraint languages for labelled drawings.

Example As an example, we consider the constraint language class $\mathcal{L}_{\mathcal{R}}$, with the signature $\mathcal{R} = \{\prec\}$. Satisfiability of the $\prec$ constraint is defined as follows:

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

By instantiating the class to a concrete lexical constraint language, for example to $\mathcal{L}_{\{\prec\}}(\{A, B, C\})$, we get a set of constraints to express precedence conditions for a specific class of labelled drawings. For example, the constraint $A \prec C$ is satisfied at a node u in a drawing $\mathfrak{D}$ if and only if each A-successor of node u precedes each C-successor with respect to the total order.

The self label To simplify the definition of constraint languages, we introduce a special edge label ι called 'self' in the syntax, distinct from all other labels in Π. It is used to constrain the node u itself in the same fashion as its successors. Subsequently, the yield and accessibility for ι are defined as

$$\triangleleft_{\iota} := \mathrm{Id} \quad \text{and} \quad \diamond_{\iota} := \mathrm{Id}$$

In the example above, this allows for a constraints like $A \prec \iota$. This constraint is satisfied at a node u if all A-successors of u precede the node u. Note that the self label is a purely notational concept, since it can always be 'compiled' into the constraint language class: we could e.g. add two unary constraints $\succ_{\iota} \pi$ and $\prec_{\iota} \pi$ to the language class instead.

2.2.2 Lexical entries

As already outlined in the introduction, a dependency grammar defines for each word w the syntactic type of w and the types of its dependents, and imposes additional restrictions. In our formal model, such a local description is given by a *lexical entry*, which specifies for a node u representing w the labels of its incoming and outgoing edges, and applies lexical constraints from a constraint language.

Definition 2.5 A lexical entry describes a node in a drawing. It is a triple

$$\langle I, \Omega ; \Phi \rangle \quad \in \quad \mathfrak{O}(\Pi) \times \mathfrak{B}(\Pi) \times \mathfrak{P}(\mathcal{L}_{\mathcal{R}}(\Pi))$$

where the option I and the bag Ω contain edge labels, and Φ is a set of lexical constraints. ⊣

The option I of a lexical entry contains the label that the incoming edge is required to have, i.e. the syntactic type of the word represented by the node, and is thus called the *type description*. For root nodes, I is the empty set. Note that it suffices to use an option, since each node in a drawing has at most one $\lhd$-predecessor. The labels on the outgoing eges are called *valencies*; they specify the types of the dependents of a node.[1] Accordingly, the bag Ω is a *valency description*. The pair consisting of a type description and a valency description is essentialy the same as a 'category' in categorial grammars [1], but without directionality.

Definition 2.6 A lexical entry $\langle I, \Omega ; \Phi \rangle$ is *well-typed* if $\Phi \in \mathfrak{P}(\mathcal{L}_{\mathcal{R}}(\Omega))$. The set of well-typed lexical entries over an edge label set Π is denoted as $LE_{\mathcal{R}}(\Pi)$. ⊣

The notion of well-typedness is introduced to ensure that constraints used in a lexical entry may only contain the valencies Ω of that entry. The main idea is to capture a notion of *locality*: at least on the side of the syntax, constraints may only distinguish specific nodes by the labels that are actually locally 'visible' from the node.

Definition 2.7 A node u in a drawing $\mathfrak{D} \in \mathcal{D}_{\Sigma,\Pi}$ *satisfies* a lexical entry $\langle I, \Omega ; \Phi \rangle \in LE_{\mathcal{R}}(\Pi)$ if

- for all $\pi \in \Pi$, $(\lhd_\pi)^{-1} u = I$,
- for all $\pi \in \Pi$, $|(\lhd_\pi) u| = \Omega(\pi)$, 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 is exactly the (empty or singleton) set of ingoing edge labels of u, Ω is exactly the bag of outgoing edge labels, and u satisfies all constraints in the set of constraints Φ.

[1] Historically, the term 'valencies' refers to arguments of a verb only. In this thesis, however, the meaning is extended to arguments of *any* word; this is an extension commonly made throughout dependency theories.

2.2.3 Theories and grammars

As mentioned above, a theory T defines a class of drawings, thereby imposing global structural constraints. Also, it defines the lexical constraint languages to be used in the grammars. A grammar of type T decribes drawings from a corresponding class of labelled drawings: it specifies the set of node and edge labels, and provides lexical entries imposing local conditions on the licensed drawings.

Definition 2.8 An LCG theory T is a pair $(\mathcal{D}, \mathcal{L}_\mathcal{R})$ where $\mathcal{D}$ is a class of drawings and $\mathcal{L}_\mathcal{R}$ is a class of lexical constraint languages. 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 function of type $\Sigma \to \mathfrak{P}(LE_\mathcal{R}(\Pi))$, which is called a *lexicon*. ⊣

A labelled drawing is then licensed by a grammar if each node is licensed by a lexical entry for the word represented by the node:

Definition 2.9 A node u in a drawing $\mathfrak{D} \in \mathcal{D}_{\Sigma,\Pi}$ *satisfies* a lexicon $Lex \in \Sigma \to \mathfrak{P}(LE_\mathcal{R}(\Pi))$ if 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 $\mathfrak{D} \in \mathcal{D}_{\Sigma,\Pi}$ *satisfies* a grammar $G = (\Sigma, \Pi, Lex)$, written $\mathfrak{D} \vDash G$, if every node $u \in \mathfrak{D}$ satisfies the lexicon of the grammar. ⊣

Definition 2.10 Let $G = (\Sigma, \Pi, Lex)$ be a grammar for the theory $(\mathcal{D}, \mathcal{L}_\mathcal{R})$. The *language of G* is defined as the set of labelled drawings that satisfy the grammar:

$$\mathcal{L}(G) := \{\, \mathfrak{D} \in \mathcal{D}_{\Sigma,\Pi} \mid \mathfrak{D} \vDash G \,\}$$ ⊣

The set of projections of drawings in $\mathcal{L}(G)$ is called the *string language* of G; it corresponds to the notion of 'language' in classical generative formalisms. To make a clearer distinction, $\mathcal{L}(G)$ is also called a *dependency language*.

2.3 Sample languages

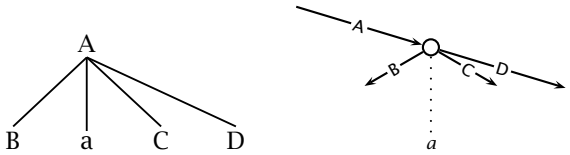
In order to provide an intuition for the concepts defined in the previous sections and to illustrate the expressivity of LCG, we will informally encode four well-known grammar formalisms as LCG theories in this section.

2.3.1 Lexicalised Context-Free Grammars

To express the Lexicalised Context-Free Grammars (LCFG) formalism as a theory in our framework, we first need to choose a class of drawings. LCFGs generate labelled, ordered trees, which are equivalent to projective drawings, as already mentioned in the introduction. Hence a proper choice for the class of drawings is $\mathcal{D}_0$.

A lexicalised context-free rule like

$$A \to BaCD$$



(a) partial tree (b) partial drawing

$$a \mapsto \langle \{A\}, \{B, C, D\} ; \{B \prec \iota, \iota \prec C, C \prec D\} \rangle$$

(c) lexical entry

Figure 2.5: Encoding of a lexicalised context-free rule.

can be seen as local well-formedness condition on a labelled tree, imposing a local order on the terminal a and the nonterminals B, C, and D (see the tree in Figure 2.5). Since these nonterminals are also the roots of projective subtrees, the order is imposed on these subtrees as well. The only structural constraint relevant to LCG encodings of LCFGs grammars is therefore linear precedence of immediate successors:

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

For translate each rule in the LCFG grammar into a corresponding lexical entry. Figure 2.5 shows the lexical entry for the sample rule, and a (partial) drawing satisfying the entry: The node labelled with the word a corresponds to the terminal in the LCFG rule. The nonterminals B, C, and D on the right hand side of the rule are the valencies of the node in LCG, while the nonterminal A on the left hand side corresponds to its type. If A was a start symbol in the original grammar, we would need to create a duplicate entry with an empty option I; such entries can only be satisfied at root nodes.

The precedence constraints in the entry enforce the order specified in the LCFG rule.[2] Note that nothing forces us to impose order constraints on *all* successors. The defined theory $(\mathcal{D}_0, \mathcal{L}_{\{\prec\}})$ therefore also allows for grammars with underspecified valency precedences.

2.3.2 Lexicalised Unordered Context-Free Grammar

A grammar of the theory above that does not use any precedence constraints corresponds to LCFGs with all possible permutations of the symbols on the the right hand sides. If we abandon the order constraints completely, we get the theory $(\mathcal{D}_0, \mathcal{L}_\emptyset)$. Since this theory only permits grammars with arbitrary valency permutations, it is equivalent to the class of lexicalised unordered context-free grammars (LUCFGs).

[2] The node itself is ordered using the 'self label' ι.

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

Figure 2.6: Linear Specification Language constraints in $\mathcal{L}_{\text{LSL}}$. The last line corresponds to an isolation constraint applied to the left hand side of an LSL rule.

2.3.3 Linear Specification Language

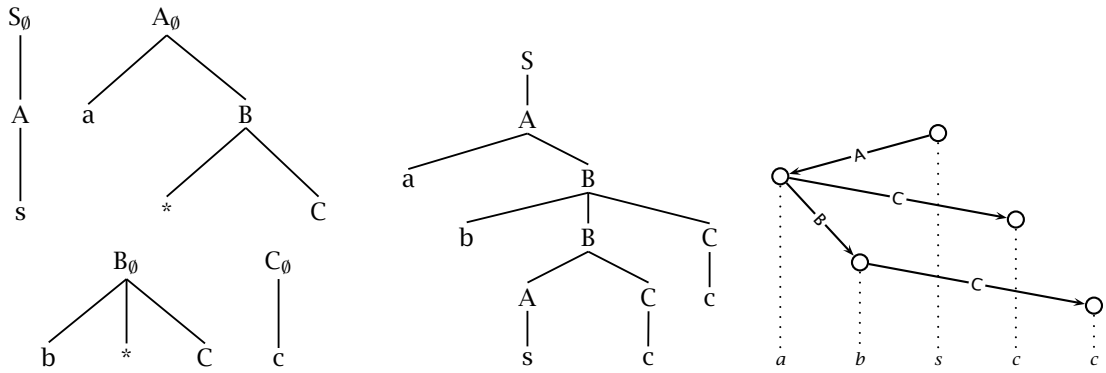
The Linear Specification Language (LSL) formalism by Suhre [18] allows to generate languages with a free word order. It is inspired by ID/LP parsing [8], but uses only context-free rules with local constraints, which makes it suitable for translation into LCG. The yields in LSL are generally discontinuous; therefore, an LCG theory for LSL needs to adopt the class of unrestricted drawings, $\mathcal{D}_*$. To restrict the possible orders, each LSL grammar rule can be annotated with three types of constraints:

- local precedence ($<$), which orders a substructure somewhere in front of another,
- immediate precedence ($\ll$), which additionally implies adjacency of both substructures, and
- isolation ($\langle\rangle$), which imposes a zero-gap constraint on a substructure. It can also be applied to the left hand side of the rule, thereby isolating the entire substructure defined by the rule.

Lexicalised LSL grammars are encoded into LCG as grammars of the theory $(\mathcal{D}_*, \mathcal{L}_{\text{LSL}})$. The rules are converted into lexical entries in a similar fashion as those in context-free grammars. The local constraints mentioned above can be defined directly in the theory as shown in Figure 2.6. Note that two variants of the isolation constraint are needed for its two types of applications.

2.3.4 Tree Adjoining Grammars

Joshi's Tree Adjoining Grammars (TAG) [12] is a formalism that is more expressive than context-free grammars. In short, a TAG grammar consists of a set of *elementary trees*, of which there are two types: *initial trees* and *auxiliary trees*. A *derived tree* is achieved by combining these elementary trees using the operations of *substitution* and *adjunction*. The projections of all derivable trees is the language of the grammar. A *derivation tree* is an unordered tree that records information about how tree structures are combined during a derivation: each node stands for an elementary tree, and the edges indicate which



(a) a TAG grammar (b) a derived tree (c) corresponding TAG drawing

Figure 2.7: Representing TAG derivation trees as drawings.

operation took place. The information in a derivation tree, together with the grammar, suffices to reconstruct the derived tree.

For our encoding, we focus on lexicalised TAG grammars in which each elementary tree contains exactly one terminal (leaf) node, the *anchor*. To keep things simple, we also exclude so-called left and right adjunctions, though it is possible to integrate them using a more complicated encoding.

A TAG grammar encoded in LCG describes TAG *drawings*, as defined by Möhl [14]. A TAG drawing corresponds to a derivation tree of a lexicalised TAG, with the difference that the nodes are ordered such that they project the same sentence as the derived tree they describe. It can be shown [14] that all TAG drawings are well-nested and have a gap degree of at most 1. We therefore choose $\mathcal{D}_{1,wn}$ as the class of drawings for our theory.

Each node in a TAG drawing has the following properties:

- The node is labelled with the anchor of the tree it represents.
- Its incoming edge is labelled with the root node nonterminal.
- For each substitution and adjunction nonterminal in the tree, there is a respective outgoing edge.

A sample TAG grammar, a possible derived tree, and a TAG drawing corresponding to the respective derivation tree are shown in Figure 2.7. Note that the projections of the drawing and the derived tree are the same.

A given tree is encoded as a lexical entry such that the type and valency descriptions match the root and the substitution/adjunction nonterminals, respectively. If adjunction is optional at a node, two entries have to be created, one including and one excluding the nonterminal in Ω. The constraints have to be chosen such that they ensure the following properties of the yields in the drawing implied by the structure of the elementary tree:

$$
\begin{array}{lll}
\mathfrak{D}, u \models \mathsf{Init}(\bullet) & \text{iff} & \lhd^* u \text{ is convex} \\
\mathfrak{D}, u \models \mathsf{Aux}(\bullet) & \text{iff} & \lhd^* u \text{ is not convex} \\
\mathfrak{D}, u \models \pi_1 \prec \pi_2 & \text{iff} & \diamond_{\pi_1} u \times \diamond_{\pi_2} u \subseteq \prec \\
\mathfrak{D}, u \models \pi_1 \lhd \pi_2 & \text{iff} & C(\diamond_{\pi_2} u) \subset C(\diamond_{\pi_1} u)
\end{array}
$$

Figure 2.8: Constraints for encoding TAG trees into LCG.

- A node in the TAG drawing represents an initial tree if and only if it has a gap degree of 0; it stands for an auxiliary tree iff it has exactly one gap.
- The order of sibling nodes in the tree must be reflected by the order of the respective entities in the drawing.
- If an adjunction node π_1 dominates another node π_2 in the tree, the yield of the π_1-successor is 'wrapped around' the yield of π_2-successor in the drawing.

The foot node in a tree is reflected by a gap in the yield of the node representing the tree. To simplify the definition of constraints that deal with gaps, we assume a special 'gap valency' $\sqcup$, and define the nodes accessible via that valency as the set of all nodes in the gap: $\diamond_{\sqcup} u := C(\lhd^* u) - \lhd^* u$.[3]

The constraints presented in Figure 2.8 are sufficient to describe the tree such that the properties given above are satisfied: the gap degree can be determined using $\mathsf{Init}(\bullet)$ or $\mathsf{Aux}(\bullet)$, the order of sibling nodes is described with the precedence constraint $\prec$, and the dominance relation in a tree can be expressed with $\lhd$.

The tree with the root A in the example is translated into the following two lexical entries:

$$
\begin{array}{lll}
a & \mapsto & \langle \{A\}, \{B, C\} \,;\, \{\iota \prec B, \sqcup \prec C, B \lhd \sqcup, B \lhd C\} \rangle \\
a & \mapsto & \langle \{A\}, \{C\} \,;\, \{\iota \prec \sqcup, \sqcup \prec C\} \rangle
\end{array}
$$

Using that schema, we get an encoded LCG grammar whose licensed drawings are equivalent to the derivation trees of the TAG grammar. Since the nodes of the drawings are ordered according to the corresponding derived trees, the string language of our grammar is exactly the language of the original TAG grammar.

[3] Like the self label, the gap valency is a notational trick that could be compiled into additional constraints.

Chapter 3

Recognition Complexity Classes

The previous chapter has demonstrated that the LCG framework is rather expressive. The question is how this expressiveness affects the performance of a parser for LCG, i.e. how much time it costs to decide whether a given sentence is the projection of a drawing licensed by a given grammar.

The first section of the present chapter formulates the recognition problem as a constraint satisfaction problem. The following part approaches the complexity discussion by examining various LCG theories and classifying them according to the worst-case time needed to solve the recognition problem.

It turns out that there are notable computational differences between the theories. In general, the power of LCG comes at the cost of a high parsing complexity, though some restricted theories can be parsed very efficiently.

3.1 The recognition problem

The *recognition problem* is to decide whether a given input string s is in the language of a given grammar G. Solving the problem is called *recognising* a string with respect to a given grammar.

In generative grammars, the problem is usually equivalent to the question whether it is possible to derive the string s (or a structure that represents s) by applying finitely many rewriting rules of G, starting with a designated symbol (or an initial structure). In LCG, the recognition problem is stated as a satisfaction problem: it is the question whether there is a drawing that satisfies a grammar G and has the projection s.

Definition 3.1 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 *general recognition problem* for G and s is the problem to decide whether there is a drawing in the language of G that projects s, i.e. whether the following set is non-empty:

$$\mathfrak{C}(G, s) := \{\, \mathfrak{D} \in \mathcal{L}(G) \mid \mathsf{proj}(\mathfrak{D}) = s \,\}$$

Elements of this set are called *configurations* of (G, s). For a particular grammar G, the *fixed recognition problem* for s is to decide whether a drawing that projects s is in the language of G. ⊣

The fixed recognition problem differs from the general problem in that the grammar G is not considered part of the input. One could think of a solver for that problem as

a deterministic Turing machine that has a grammar G 'built in' and takes an input sentence to recognise. The time it needs is measured with respect to the sentence only.

Relation to the word problem In classical theories, the recognition problem is often called *word problem*. This is because elements of a language are called *words*, which are sequences of *symbols*. LCG terminology, however, follows a more linguistic perspective: projections are *sentences*, which are sequences of *words*. In order to avoid confusion concerning the notion of 'word', we use the term 'recognition problem'.

Relation to the membership problem The *membership problem for a language L* is to decide whether L contains a given sentence. The membership problem is polynomial if there is a deterministic Turing machine that decides the problem in polynomial time with respect to the sentence length. Such languages are also called PTIME languages.

The membership problem is defined independently of any grammar. Nevertheless, if a language is generated or described by a grammar for which the fixed recognition problem is polynomial, then there is a polynomial-time decision function for the language.

Complexity classes of LCG theories It is clear that all recognition problems for LCG are in NP: a non-deterministic Turing machine can simply guess a labelled drawing that projects the given sentence, and check the valency descriptions as well as the lexical constraints in polynomial time.[1]

Most of the following proofs show that the general and the fixed recognition problems for different LCG theories are actually NP-hard. These proofs mostly follow the usual technique of reduction from another problem that is known to be NP-hard.

3.2 The general recognition problem

Lemma 3.2 The general recognition problem for $(\mathcal{D}_*, \mathcal{L}_\emptyset)$ is NP-hard. □

PROOF There is a polynomial reduction of HAMILTON PATH to the general recognition problem for $(\mathcal{D}_*, \mathcal{L}_\emptyset)$. More specifically, for each input graph $H = (V; E)$ to the HAMILTON PATH problem, it is possible to construct 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, i.e. a path that visits every node in V exactly once. The construction of G_H and s_H can be done in time linear in the size of the input graph.

[1] Remember that LCG constraints must be decidable in polynomial time with respect to the size of the grammar.

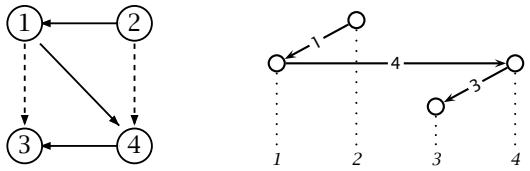


Figure 3.1: An input graph H for HAMILTON PATH and a drawing licensing Lex_H. The Hamilton Path in H is marked by solid edges.

Let s_H be some string over V in which each node in V occurs exactly once. The grammar G_H is defined as follows:

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

Each Hamilton Path in H forms a linear tree on V, i.e. a tree in which each node has at most on successor. Each such tree can be configured using G_H by choosing, for each node v in H, an entry from either $\mathsf{start}(v)$, $\mathsf{end}(v)$, or $\mathsf{inner}(v)$, depending on the position of v in the Hamilton Path. (Note that each edge is marked with the label of the node it points to.) Conversely, in each configuration of (G_H, s_H), each node has at most one predecessor and at most one successor by definition of the lexicon. Therefore, each such configuration is a drawing whose successor relation forms a linear tree. As every node in the drawing corresponds to a node in the input graph H, the path from the root to the leaf identifies a Hamilton Path in H. ∎

To illustrate the encoding, Figure 3.1 gives a sample input graph H and a corresponding configuration. The depicted drawing satisfies the following lexicon Lex_H. (The lexical entry satisfied at each node is underlined.)

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

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

Corollary 3.3 The general 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. To see this, consider the configuration on the right in Figure 3.1: node 1 has a gap degree of 1 due to the position of the parent node. With an increasing number of nodes, more gaps become possible.

One may wonder if restricting the gap degree reduces the complexity of the general recognition problem. This, however, is not even the case for drawings without gaps:

Lemma 3.4 The general recognition problem for $(\mathcal{D}_0, \mathcal{L}_\emptyset)$ is NP-hard. □

PROOF As mentioned in Section 2.3, each $(\mathcal{D}_0, \mathcal{L}_\emptyset)$ grammar can be transformed into an equivalent lexicalised unordered context-free grammar. Therefore, the general recognition problem for $(\mathcal{D}_0, \mathcal{L}_\emptyset)$ is equivalent to the respective problem for LUCFGs, which is known to be NP-hard [2]. ∎

3.3 The fixed recognition problem

The fixed recognition problem asks the same question as the general problem, but the grammar is not considered part of the input. Therefore, it is not possible to apply the reductions used in the previous sections, as these reductions *constructed* a new grammar for every input problem, while any reduction to the fixed recognition problem needs to assume one fixed grammar for every problem instance encoded as input string.

Lemma 3.5 The fixed recognition problem for $(\mathcal{D}_0, \mathcal{L}_{\{\prec\}})$ is polynomial. □

PROOF The grammars of the given theory are lexicalised context-free grammars with a possibly underspecified valency precedence. Each grammar of this type can be compiled into a strongly equivalent (ordered) context-free grammar: for each lexical entry, a set of corresponding context-free rules is created, one for each valency order that is licensed by the constraints. Ordered context-free grammars can be recognised in polynomial time. The compilation may be exponential with respect to the size of the original grammar, but this factor that can be ignored for the fixed recognition problem. ∎

There are, however, more complex LCG theories that are NP-hard even for the fixed recognition problem. The next subsection shows this for the Linear Specification Language.

3.3.1 NP-hardness of the Linear Specification Language

Lemma 3.6 The fixed recognition problem for $(\mathcal{D}_*, \mathcal{L}_{\text{LSL}})$ is NP-hard. □

PROOF The proof presented here is essentially a simplification of the proof given by Holzer et. al in an unpublished manuscript. We show that there is a polynomial-time reduction from the TRIPARTITE MATCHING (TPM) problem to the general recognition

problem for LSL. More precisely, we give an LSL grammar and an encoding of TPM problems as strings such that a string is in the language of the grammar iff there is a solution to the corresponding TPM problem. Note that the grammar is fixed and independent of any particular string encoding. As the grammar is strongly lexicalised, the proof can be done in a very similar fashion for the equivalent LCG theory $(\mathcal{D}_*, \mathcal{L}_{\text{LSL}})$.

TRIPARTITE MATCHING (TPM) Given three sets A, B, and C of the same size n, and some relation $T \subseteq A \times B \times C$ between the three sets. Without loss of generality, we can assume $A = B = C = \{1, \ldots, n\}$. The TPM problem is to decide whether there is a subrelation $T' \subseteq T$ such that $|T'| = n$ and no two elements of T' agree in any coordinate.

Encoding an instance as input string Every triple $(i, j, k) \in T$ is encoded as the string $\$d^i\$e^j\$f^k$, where d, e, f, and \$ are terminals, and x^m means 'the terminal x written m times'. This encoding is obviously unique, i.e. no two different triples can have the same encoding. We will call the blocks of ds, es, and fs the *coordinate blocks* of a triple. The encoding of T, written $\langle T \rangle$, is the concatenation of all encodings of every triple in T (in an arbitrary order).

For a given set T and a size n, the input string for the LSL grammar is defined as:[2]

$$a^1\#a^2\#\ldots\#b^n\#b^1\#b^2\#\ldots\#b^n\#c^1\#c^2\#\ldots\#c^n\#\langle T\rangle\ m^n.z^{|T|-n}$$

The blocks of as, bs, and cs encode the elements of the input sets A, B, and C, respectively, and will be called *element blocks*. Furthermore, there are as many ms as there are triples in the solution T', and as many zs as there are triples that do not appear in T'. The transformation of a TPM problem into an input string can be done in polynomial time.

Example Let $n = 2$, $A = B = C = \{1, 2\}$ and $T = \{(1, 1, 2), (2, 1, 2), (2, 2, 1)\}$. The following line is an encoding of the problem:

$$a\#aa\#b\#bb\#c\#cc\#\ \$d\$e\$ff\ \$dd\$e\$ff\ \$dd\$ee\$f\ mm.z$$

The grammar The grammar is defined as follows:

$$G = (\{S, M, Z, A, B, C, D, E, F\}, \{a, b, c, d, e, f, m, z, \#, \$, .\}, P, L, S)$$

where the rules in P are defined as:

[2] a, b, c, m, z, and # are terminals.

S	$\rightarrow$	$M.Z$	; ϵ				
M	$\rightarrow$	$ABCmM$	; ϵ	Z	$\rightarrow$	$DEFzZ$	; $D \ll E \ll F$
M	$\rightarrow$	$ABCm$	; ϵ	Z	$\rightarrow$	$DEFz$	; $D \ll E \ll F$
A	$\rightarrow$	aAd	; $a \ll A \ll d$	D	$\rightarrow$	Dd	; $D \ll d$
A	$\rightarrow$	$a\#\$d$	; $a \ll \# < \$ \ll d$	D	$\rightarrow$	$\$d$	; $\$ \ll d$
B	$\rightarrow$	bBe	; $b \ll A \ll e$	E	$\rightarrow$	Ee	; $E \ll e$
B	$\rightarrow$	$b\#\$e$	; $b \ll \# < \$ \ll e$	E	$\rightarrow$	$\$e$	; $\$ \ll e$
C	$\rightarrow$	cCf	; $c \ll A \ll f$	F	$\rightarrow$	Ff	; $F \ll f$
C	$\rightarrow$	$c\#\$f$	; $c \ll \# < \$ \ll f$	F	$\rightarrow$	$\$f$	; $\$ \ll f$

Note that this is a simplified notation without the preterminals required in LSL. For each terminal x, one actually needs to include a preterminal $\overline{x}$, replace x with $\overline{x}$ in each rule in P, and add a rule $\overline{x} \rightarrow x$ to L.

Explanation The S-rule starts the parsing process in which the M- and the Z-rules are applied one or more times each (both of them have a recursive version). The Z-rules match each terminal z with a complete, isolated encoding of a triple $\$d^i\$e^j\$f^k$ (D-, E-, and F-rules), thereby 'eliminating' the $|T| - n$ triples that do not belong to the solution.

There are exactly n triples remaining, i.e. n coordinate blocks of ds, es, and fs, respectively. Besides, there are still the n element blocks of as, bs, and cs, respectively, as well as n times the terminal m. In TPM, we have to find each triple number exactly once in the corresponding input set. In the encoding, this means we have to match each coordinate block with an equally long element block of the corresponding set. The M-rules perform for each triple[3] just that matching with three element blocks and an m in the following way:

The A-rules match two continuous substrings $a^i\#$ and $\$d^i$ of the same length. By the constraints given in the rules, there may be a gap between the a- and d-blocks, but the blocks themselves are isolated. The B- and C-rules do likewise for the strings $b^j\#$ and $\$e^j$ as well as $c^k\#$ and $\$f^k$. Figure 3.2 shows a sample derivation for an a-block of length 2 and an equally long matched d-block. Both blocks are convex but not necessarily adjacent.

Correctness and Completeness The following properties hold:

- No block could possibly be matched only partially, as the remaining part of the block does not contain a # or $ separator, rendering it 'useless' .

[3] Actually, the M-rules do not require the three matched coordinate blocks to be of the same triple, i.e. adjacent. This is not needed anyway, since it suffices to match each coordinate block with a corresponding element block.

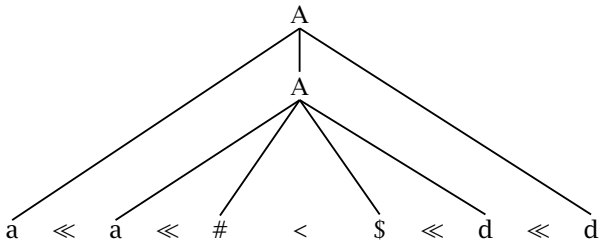


Figure 3.2: A sample matching of an element block with a coordinate block.

- The choice of different terminal symbols ensures e.g. that a first coordinate (block of ds) can only match an element of set A (not B or C).

Thus if there is a parse for the input string, it also represents a solution for the original tripartite matching problem. If there is a solution, on the other hand, it is possible to find it with the given encoding and grammar: the $|T| - n$ triples not belonging to the solution can be matched by the Z-rules, while it is possible to combine the n triples of the solution with the encodings of the elements of the sets A, B, and C. Thus the reduction is sound and complete. ∎

Remarks on the proof In the main part, the parser searches for matchings between coordinates and elements, and combines found matchings. Since each matching has a discontinuity (gap), the number of discontinuities in an intermediate parse result grows with the number of combined matchings. In fact, completeness of the reduction relies on the unboundedness of the number of discontinuities in an intermediate parse result. In other words, for each $g \in \mathbb{N}$, one can find a problem instance (input sentence) that is large enough so that each parse solution contains a subtree with more than g discontinuities. This unboundedness is the main reason why the fixed recognition problems for LSL are NP-hard. Vice versa, if the number of discontinuities is restricted, the fixed recognition problem actually becomes polynomial, as shown by Suhre [18].

3.3.2 Polynomiality of the LCG theory without constraints

The proof above exploits the fact that LSL grammars contain no restrictions on the number of gaps. One may assume that the fixed recognition problem for *every* theory with an unrestricted class of drawings is NP-hard. However, this is disproven by the following remarkable result.

Lemma 3.7 The fixed recognition problem for the most general LCG theory $(\mathcal{D}_*, \mathcal{L}_\emptyset)$ is polynomial. □

The proof is presented in Section 4.5. It shows that polynomiality can be achieved *because* there are no global or local constraints in this theory.

3.4 Conclusion

So far, we have explored the complexity classes of general and fixed recognition problems for several specific LCG theories. Although the analysis provides only a rough complexity estimate, it reveals large computational differences between theories within the framework, for reasons that are not entirely obvious. In the next chapter, we will give a precise analysis and isolate the factors that contribute to recognition complexity.

Chapter 4

Computational Factors

The last chapter has shown that the complexity classes of the fixed and general recognition problems vary between different theories. In this chapter, we analyse the complexity of LCG parsing in more detail and identify the computational factors contributing to recognising complexity.

It turns out that constraint solvers, although they seem self-evident for parsing LCGs, are inappropriate for a complexity analysis. Instead, computational aspects of LCG grammars are much easier to investigate by using chart-based algorithms, a technique that is usually associated with generative formalisms. We will therefore present an abstract chart parser for LCG, and isolate the factors influencing complexity. This is followed by an exploration of possibilites to restrict these factors in order to gain a better parsing performance, leading to a characterisation of theories for which the fixed or the general recognition problem is polynomial. The last section examines how efficient parsers for particular theories exploit the restrictions of computational factors to achieve a better time complexity.

Parsing and recognising In the following, the term 'parsing' is used interchangeably with 'recognising', i.e. the process of deciding the mere existence of a satisfying drawing for a given grammar and sentence. In a wider sense of the term, 'parsing' also includes the enumeration of the actual syntactic analyses. This additional step may take exponential time even for context-free grammars, and is not examined here.

4.1 Parsing techniques

Constraint solvers The most obvious approach for the implementation of LCG parsers are constraint solvers. The constraint problem for a given sentence and grammar can be elegantly formulated as a task to find a matching drawing: the number of nodes, their labels and total order are determined by the input sentence; the solver has to search for a lexical entry at each node such that the type and valency descriptions as well as constraints are satisfied.

Although constraint solvers appear as the 'natural' parsing technique for a declarative framework like LCG, they are not easy to handle when it comes to complexity issues. This is because a constraint solver is usually a generic algorithm with an inherent exponential time complexity. An improvement in efficiency is very difficult to achieve and

not to be expected. Even if a faster algorithm can be found by implementing optimisations for special cases, one can hardly give reliable information about the exact time complexity, as the performance factors and parameters often strongly vary with the problem instances. It therefore does not look very promising to analyse the complexity of constraint solvers for LCG.

Nevertheless, there *are* fragments of LCG that can be translated back into well-known grammar formalisms, like the aforementioned lexicalised context-free grammars. For these formalisms, there exist parsers with a proven high efficiency [6]. In general, the computational aspects in the field of generative grammar formalisms are well understood, and there are thus many specialised parsers that directly make use of specific grammar properties to increase performance. These parsers often use a *tabular* or *chart parsing* technique.

Chart parsers A chart parser explores systematically the space of possible parses of the input string and represents already recognised substructures with *chart entries*. These entries are stored in a *chart*, which is a data structure for bookkeeping to eliminate backtracking and avoid computational duplication. The chart allows for an efficient lookup and retrieval of stored entries. During the parsing process, the parser successively builds larger parse structures by combining the items according to specific rules. The parsing strategy finally ensures that no combination is done twice. As the chart parser saves and reuses the results of solved subproblems (parses), it can be seen as an example of dynamic programming.

Chart-parsing LCG The design of the LCG framework makes it possible to parse arbitrary LCG grammars using a chart parser:

- Since the type and valency descriptions in an LCG lexicon define a tree structure, they can be used for bottom-up parsing.
- For the moment, the lexical constraints will be used as filters only: intermediate parsing results which violate constraints are removed.[1]

4.2 General parsing schema

Instead of presenting a concrete chart-based parsing algorithm, we provide a declarative specification of a class of such algorithms in form of a *parsing schema* [17]. Parsing schemata allow us to analyse the complexity of these algorithms on a high level of abstraction, hiding the algorithmic details. They make it easy to identify the computational factors, and to explore how parsers can benefit from restricting these factors.

[1] In the next chapter, we will see particular types of constraints and how to build them directly into the parser, so that the parser fulfils them 'blindly' by the way it works.

Parse items and inference rules

Parsing schemata 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. Each of these parse items has the form

$$s : \langle I, \Omega \rangle$$

where s is a *span*, i.e. a non-empty subset of sentence positions, and I and Ω are an option and a bag of edge labels, respectively. Each parse item stands for the information that the grammar licenses a partial drawing covering the words of the input sentence specified by s. To complete a drawing, one needs to connect its root node using an incoming edge and outgoing edges labelled with the labels in I and Ω, respectively. A parse item $s : \langle I, \emptyset \rangle$, in which Ω is empty, is called *fully saturated*. Such an item stands for an entire subdrawing, possibly with an incoming edge.[2] An item $s : \langle \emptyset, \emptyset \rangle$ is called *complete*; it is also *final* if s contains all the positions in the sentence. A final item encodes a parse solution, as it stands for a drawing licensed by the grammar and covering the whole sentence.

The lookup rule The schema contains two rules called LOOKUP and COMBINE. The LOOKUP rule creates a new parse item with a singleton span for a word w at position i in the input sentence by retrieving type and valency information from a lexical entry for the word:

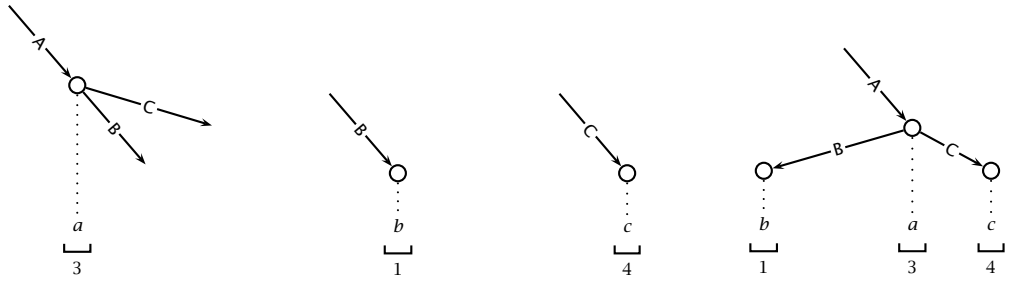
$$\frac{\mathsf{pos}(i) = w \qquad \langle I, \Omega\,; \Phi \rangle \in \mathit{Lex}(w)}{\{i\} : \langle I, \Omega \rangle}\ \textsc{lookup}$$

The combination rule The COMBINE rule derives new parse items from existing ones. It fully saturates all open valencies of an item at once by combining it with other items accepting these valencies on the incoming edges:

$$\frac{s_0 : \langle I_0, I_1 \uplus I_2 \uplus \cdots \uplus I_m \rangle \qquad s_1 : \langle I_1, \emptyset \rangle \qquad s_2 : \langle I_2, \emptyset \rangle \qquad \ldots \qquad s_m : \langle I_m, \emptyset \rangle}{s_0 \oplus s_1 \oplus s_2 \oplus \cdots \oplus s_m : \langle I_0, \emptyset \rangle}\ \textsc{combine}$$

The first premise always stands for a single node, since unsaturated items are only derived by LOOKUP. The m fully saturated premises to the right represent subdrawings whose incoming edges match the types required by the outgoing edge label description of the node they are 'plugged' into. None of these m premises may be a complete item representing a finished drawing without incoming edges: otherwise, the derived item would stand for a forest-like set of drawings, which we do not want to allow. Therefore I_1 to I_m must be singleton sets. In contrast, the option I_0 may also be empty.

[2] Fully saturated items correspond to the notion of *inactive edges* in classical parsers, while unsaturated or partly saturated items are equivalent to *active edges*.



$\{3\} : \langle \{A\}, \{B, C\} \rangle$ $\{1\} : \langle \{B\}, \emptyset \rangle$ $\{4\} : \langle \{C\}, \emptyset \rangle$ $\{1, 3, 4\} : \langle \{A\}, \emptyset \rangle$

Figure 4.1: Three initial items derived by LOOKUP (left) and the combined item derived by COMBINE (right), with the partial drawings they stand for. A number below a node label indicates the sentence position.

The parse item in the conclusion represents the combination of the premises and thus covers the *union* of their spans $(s_0, s_1, \ldots, s_m)$. The union is computed using the $\oplus$ operator, which is a specialisation of the disjoint union of sets: As the derived item must represent a tree structure, the operator $\oplus$ requires the spans to be disjoint, and is undefined if they overlap. For concrete theories, the operator is restricted even further depending on the class of drawings that the schema is applied to. In a theory over $\mathcal{D}_2$, for example, every fully saturated item stands for a drawing with at most two gaps; hence $\oplus$ is only defined on spans whose union has at most two gaps.

Instantiating the schema The schema can be instantiated to a *parsing system*. More precisely, for a given input sentence and a given grammar, the LOOKUP rule is instantiated for each lexical entry of each word in the sentence. The COMBINE rule is instantiated for each lexical entry and all possible spans such that $I_1 \uplus I_2 \uplus \cdots \uplus I_m$ is exactly the valency description, and I_0 is the type description of the entry.

Constraint checking Note that the schema presented above abstracts away from the verification of lexical constraints. Instead, constraint checking is seen as a matter of implementation of an actual parser. As each instance of COMBINE corresponds to a lexical entry with local constraints, a parser can check the validity of a combination step immediately by verifying the premise items against the constraints. If the constraints do not hold, the derived item is filtered out.

Example

We consider a sample sentence *b d a c e* and a grammar containing the following entries in the lexicon:

$$\begin{aligned} a &\mapsto \langle \{A\}, \{B, C\} ; \emptyset \rangle \\ b &\mapsto \langle \{B\}, \emptyset ; \emptyset \rangle \\ c &\mapsto \langle \{C\}, \emptyset ; \emptyset \rangle \end{aligned}$$

Figure 4.1 shows a sample derivation: The LOOKUP rule derives three items representing the single nodes a, b, and c from their lexical entries and positions in the sentence. The COMBINE rule then 'plugs' the partial drawings of type B and C into the node a.

Implementation

A concrete chart parsing algorithm using the general schema would test whether the inferential closure of the rules contains a final item. This can be done efficiently by recording derived items into the chart, and by using a parsing strategy ensuring that no item combination is performed twice.

For example, an implementation is proposed by Shieber, Schabes and Pereira [16]: The parser makes use of a chart (list) and a special parse item queue called *agenda*. Initially, the parser adds all items derived by the LOOKUP rule to the agenda. In the main loop, the parser takes the front item p from the agenda. It tries to combine it with m items in the chart. (Note that p does not need to be the left-most item in the COMBINE rule.) Finally, p is moved to the chart. Constraint checkers verify the consistency of derived parse items with all lexical constraints. If the item is valid, it is appended to the agenda, unless the very same item already exists in the agenda or chart. No combination is done twice for the following reasons: all items first appear in the agenda and are then moved to the chart, each combination contains exactly one agenda item, and no duplicates are allowed.

Soundness and completeness

We need to ensure that all the inferences are valid and that every licensed drawing can be derived with them, i.e. that the schema is sound and complete. This is easy to show for the general case presented here.

A chart parser instantiating and implementing the schema follows a strict bottom-up parsing strategy. It starts with initial parse items for each word (LOOKUP rule). The COMBINE rule fills open valencies with matching, fully saturated items. As derived items are also fully saturated, they are available for combination with other, unsaturated items.

The parsing process abstracts the configuring of a drawing licensing the grammar: an initial parse item stands for a single node, with labelled adjacent edges. The outgoing

edges are connected with entire subdrawings that satisfy the requested type and lexical constraints. The resulting subdrawing is licensed by the grammar; the derived fully saturated item representing it is therefore sound.

The bottom-up parser is also complete: We consider a combined partial drawing that is licensed by the grammar. It is licensed because a) the root node and its adjacent edges match a lexical entry, b) the subdrawings are licensed by the grammar and c) the lexical constraints in the entry are satisfied. Then an item for that combined partial drawing is derived because a) all possible unsaturated items for single nodes are initally derived, b) items for the subdrawing are derived by induction over the size of the drawings represented by the parse items, and c) the constraints are verified.

4.3 Complexity analysis

This section first presents the dimensions in which complexity is measured. This is followed by an analysis of the parsing complexity for the general schema above. We will isolate the factors that influence performance, and give upper complexity bounds for the worst case.

4.3.1 Dimensions of Complexity

A parser recognises a sentence with respect to a given grammar. The time complexity is measured with respect to the following dimensions:

- n, the length of the input sentence,
- $|G|$, the size of the input grammar, and
- k, the maximal number of valencies in an applicable entry.

The value n is defined as the number of words in the input sentence. We assume that the grammar size $|G|$ is the sum of the sizes of the lexical entries; the size of a lexical entry is the size of its valency set.

The value k is usually the size of the largest lexical entry in the grammar, which gives $|G|$ as an upper bound. However, there may exist entries that have more than n valencies. These entries do not need to be considered when parsing a sentence of length n: as LCG grammars are lexicalised, such an entry describes a drawing with at least $n + 1$ nodes, which cannot be licensed by the sentence. We may therefore focus on 'applicable' entries with at most n valencies. The value k is thus also bounded by n. However, we will keep the factor k abstract as long as possible. The advantage is a higher precision throughout the analysis. In the end, k can be approximated with either n or $|G|$.

A note on the size of the grammar The impact of the grammar size on the performance of parsing algorithms is an often underrated or even ignored aspect. However, a lexicalised grammar for a natural language like English, for example, is usually many times larger than a standard English sentence of about 25 words. Small changes in the grammar are thus significant even for context-free formalisms, where grammar size might only contribute quadratically to the speed of the parsing algorithm. In a framework with a high expressive power, such as LCG, the size of the grammar is definitely an issue which is not to be ignored.

Relation to complexity classes The complexity class results from the previous chapters can be roughly expressed in terms of the complexity dimensions. Note that throughout this work, it is assumed that P ≠ NP.[3]

The fixed recognition problem is polynomial iff it can be solved in polynomial time with respect to the length of the input sentence. The general recognition problem is polynomial if and only if it can be solved in polynomial time with respect to both $|G|$ and n, which implies polynomiality of the fixed recognition problem.

If the general recognition problem is NP-hard, this means that for each input sentence, there is a grammar such that recognising the sentence cannot be done in deterministic polynomial time with respect to n *and* $|G|$.[4] If the fixed recognition problem is NP-hard, then there is at least one specific grammar for which recognising sentences generally cannot be polynomial with respect to their length. An NP-hard fixed recognition problem implies NP-hardness of the general recognition problem.

4.3.2 Time and space complexity

For the moment, we postpone the analysis of the complexity of constraints, and assume that the time to verify lexical constraints after each combination is bounded by some value c.

Time complexity The parser needs to regard the LOOKUP rule only in the beginning. For each word in the sentence, it searches the grammar for appropriate lexical entries. This rule thus contributes at most $O(n \cdot |G|)$ to the overall time complexity. As we will see, this summand is much smaller than the complexity of the COMBINE rule, thus we can ignore it.

For COMBINE, the parser needs to combine a parse item with up to k other items. Assuming that $\|P\|$ is the number of possible parse items, the upper bound for the number of combinations is therefore $\|P\|^{k+1}$, since the chart-based architecture guarantees that no combination is done twice. After each combination, the parser needs a time of at

[3] If, at some day, both classes are proven to be actually equal, this would have a severe impact on much more results than those presented here anyway.

[4] It still might be polynomial with respect to *either* $|G|$ *or* n.

most c to check the constraints. The overall time complexity is thus in $O(\|P\|^{k+1} \cdot c)$. As the number of possible items $\|P\|$ is also the maximal size of the chart, time complexity depends on space complexity.

Note that there is no 'hidden' complexity within the combination process, as can be seen in the implementation described above. Since the chart and the agenda are efficient data structures, adding and looking up items are constant time operations.[5]

Space complexity Since parse items have the form $s : \langle I, \Omega \rangle$, the maximal number of possible parse items depends on the number of possible values for the variables of a parse item: $\|P\| \leq \|s\| \cdot \|I\| \cdot \|\Omega\|$.

As the variable I contains one or no edge label, an upper bound for $\|I\|$ is $|G|$. The set Ω is a subset of an outgoing valency description, which is a multiset that contains at most k elements. We can therefore have at most $|G|^k$ different instantiations of Ω. In the most general case, the drawings under consideration are unrestricted so that a span s can contain arbitrary sentence positions. Hence s can be represented by a set of integers ranging from 1 to n, or alternatively by a vector of n bits that indicate the positions covered by the item. In any case, the number of different possible spans $\|s\|$ is 2^n.

In conclusion, the upper bound for the number of possible items is

$$\|P\| \leq \underbrace{2^n}_{\|s\|} \cdot \underbrace{|G|}_{\|I\|} \cdot \underbrace{|G|^k}_{\|\Omega\|}$$

The time complexity is therefore in $O(\|P\|^{k+1} \cdot c) = O((2^n \cdot |G|^{k+1})^{k+1} \cdot c)$.

A refined analysis Some of the variable parts in the premises of the COMBINE rule are actually determined: all premises but the first one are fully saturated, i.e. have an empty set Ω, and their incoming edge labels I_i must occur in the outgoing labels of the first item. If we equip the parser with indexing techniques [16], the search in the chart can be improved. For example, if the item $\langle \{A\}, \{B\} \rangle$ is on the agenda, the parser can directly retrieve all items of the form $\langle \{B\}, \emptyset \rangle$ from the indexed chart, of which there can be at most 2^n, as they only differ in the spans. This is much more efficient than traversing the entire chart. In this way, the time needed for finding the items only depends on the possibilites for the parts of the premises that are actually variable. We therefore get the following result:

Lemma 4.1 Let $\|R\|$ be the number of instantiations of the COMBINE rule. The time complexity of the parsing schema is in $O(\|R\| \cdot c)$. □

[5] More precisely, the operations can be performed in constant time by implementing the chart as a hash table. For the agenda (queue), one could maintain a separate hash table with the invariant that it always contains the same items as the agenda.

There are $k+1$ spans occurring in the premises of COMBINE, as well as at most $k+1$ different edge labels. Hence we only need to take the type and valency description of the left-most premise into account, and get a worst-case time complexity of

$$O(\|s\|^{k+1} \cdot \|I\| \cdot \|\Omega\| \cdot c) = O((2^n)^{k+1} \cdot |G|^{k+1} \cdot c)$$

Note that this result is still exponential in the maximal number of valencies, k. Unless k is limited by some constant, the only upper bound for k we may assume in the analysis is $|G|$ or n. Therefore, parsers described by the schema are, in general, exponential in either the grammar size or the sentence length.

4.3.3 Isolating computational factors

Considering the analysis above, three computational factors that influence recognition time complexity can be isolated:

- The time c needed for checking the constraints after a combination, which may depend (as will be shown) on both n and $|G|$,
- the representation of spans, which influences complexity with respect to n,
- the number of combined items, which is bounded by k.

4.4 Improving efficiency

We have seen that parsing of LCG grammars is exponential in both the grammar size and the length of the input sentence. In this section, we will analyse how the computational factors can be restricted for particular theories and cases, and how efficiency improves.

Restricting constraint complexity

Since we implement the constraints as filters, we need to check them after each combination step. More precisely, an implementation would store at each item a 'history' of references to the items it has been constructed of. When an item is derived that already exists, references to its origin items would be added to the history.

The satisfiability of a constraint may be defined on whole subdrawings; a constraint verifier therefore may need to backtrack the combination history of a parse item in the chart in order to 'reconstruct' the partial drawings it represents. One of the efficiency aspects of chart parsers, however, is the fact that a parse item may stand for an exponential number of drawings. Checking constraints by using the history to enumerate all represented drawings may therefore need exponential time.

However, if the semantics of each constraint can be defined in terms of the yields of the subdrawings it refers to, a constraint verifier only needs to consider the parse

items involved in a combination directly without backtracking. In this case, it can be implemented with a time complexity that is linear in the number and the size of the items, hence we get $c \in O((k+1)n + k + 1) = O(kn)$.[6]

Restricting span representations

If a theory is defined for a class of drawings with a maximal gap degree of g, this gap restriction can be used for an efficient representation of the spans. All drawings described by grammars of such theories have that gap limitation. Therefore, all items derived during parsing have at most g gaps in their spans, since there are only two types of items: initial items that contain only one word (gap 0), and fully saturated items that stand for saturated subdrawings, which cannot have a gap degree larger than g. Without losing completeness, all spans can then be represented using at most $2(g+1)$ integer indices, denoting the start and end positions of the $g+1$ intervals that the span consists of. For example, if all items can have at most one gap in the span, the sentence positions $\{2,3,4,5,8,9\}$ can be represented by the quadruple $(2,5,8,9)$. In general, the maximal number of such tuples of indices is then $\|s\| \leq n^{2(g+1)}$, i.e. polynomial in the length of the sentence.

We define a normalised form for the span tuples. A *valid span* is an tuple of $2 \cdot m$ positions in ascending order denoting m intervals, such that the start position of an interval does not immediately follow the end position of the preceding interval. The length of the tuples is therefore not unique: if the theory is defined on drawings with a gap degree of at most 1, for example, a span in a parse item may either be a pair or a quadruple. The definition of the union operation on spans is changed accordingly so that it produces only valid spans.

4.5 Polynomial fixed recognition problem

Combining the two results from above, we get the following sufficient characterisation of theories that have a polynomial fixed recognition problem:

Lemma 4.2 Let $T = (\mathcal{D}_g, \mathcal{L}_\mathcal{R})$ be an LCG theory such that $\mathcal{D}_g$ is a class of drawings with gap degree of at most g, and the semantics of all constraints in $\mathcal{L}_\mathcal{R}$ can be defined in terms of the yields of the referred subdrawings only. Then the fixed recognition problem of T is polynomial. □

PROOF The gap restriction allows for an efficient span representation, while the condition an the semantics of constraints allows for fast constraint checking. The overall time complexity is thus limited by

$$\underbrace{(n^{2(g+1)})}_{\|s\|}^{k+1} \cdot \underbrace{|G|}_{\|I\|} \cdot \underbrace{|G|^k}_{\|\Omega\|} \cdot \underbrace{kn}_{c}$$

[6] There are at most $k+1$ spans of size n as well as $k+1$ edge label variables.

For the fixed recognition problem, we may assume a constant grammar size $|G|$. Furthermore, we also approximate k by $|G|$. Since the O-notation allows us to ignore constant factors, time complexity is in $O(n^{2(g+1)(|G|+1)} \cdot n)$. The exponent only depends on the constants g and $|G|$, therefore all grammars of the given theory T are recognisable in polynomial time with respect to the sentence length, which makes the fixed recognition problem polynomial. ∎

In the time complexity given in the proof, the exponent of the polynomial still depends on the size of the grammar. The result is not satisfying: Lexicalised context-free grammars, for example, can be encoded in LCG; their recognition complexity is known to be in $O(n^3)$, regardless of the grammar size. The next section contains a modified parsing schema in which only a constant number of premises is combined. Under certain conditions, the time complexity of that schema with respect to n is independent from the size of the grammar.

Polynomiality with respect to the size of the grammar

Interestingly, recognising the same type of grammars is also polynomial with respect to the size of the grammar, since another upper bound for k is n. Assuming a constant sentence length, we get a complexity of $O(|G|^{n+1})$. This even holds when dropping the gap restriction requirement.

In general, we cannot avoid the value k in the exponent, due to the unbounded number of premises in the COMBINE rule. Time complexity is thus always either exponential in n or $|G|$.

Polynomiality of the LCG theory without constraints

The proof above suggested that a restricted gap degree is required for the fixed recognition problem to be polynomial. This is, however, not the case: The result from Section 3.3 shows that even gap-unrestricted theories may be polynomial in n:

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

PROOF The spans of the parse items can be represented as *Parikh vectors*, which are mappings between node labels and integers. Such a vector tells how often each node label (word) appears in the set of words that are covered by an item. Given the alphabet $\Sigma = \{\text{loves}, \text{Mary}, \text{Peter}\}$, for example, an item with the span vector $(\text{loves} \mapsto 1, \text{Mary} \mapsto 1, \text{Peter} \mapsto 0)$ stands for all partial drawings covering the words 'loves' and 'Mary' each exactly once in the sentence. Note that the order of the words is not important, as the string languages of $(\mathcal{D}_*, \mathcal{L}_\emptyset)$ are closed under permutation. The union of spans is defined as the vector addition. A string is recognised if a parse item $\vec{s} : \langle \emptyset, \emptyset \rangle$ can be derived, where $\vec{s}$ is the Parikh vector of the input string. Since there are no lexical constraints in this theory, the only factor that depends on the length of the input string is

the number of spans. Each node label from Σ may be mapped to any number between 0 and n, as no label may occur more than n times. Hence we have $(n+1)^{|\Sigma|}$ different such Parikh vectors. Since there are no constraints, we can ignore c in our analysis and get a time complexity of

$$O(\|s\|^{k+1} \cdot \|I\| \cdot \|\Omega\|) = O(((n+1)^{|\Sigma|})^{k+1} \cdot |G| \cdot |G|^k)$$

An upper bound for $|\Sigma|$ is $|G|$, as we can ignore all labels from Σ for which there is no lexical entry. Assuming a constant grammar size and using the same argumentation as above, we get a time complexity that is polynomial in n. ∎

The interesting aspect is that the 'lazy' representation of spans as vectors is only possible because there are no constraints at all in this theory, neither global structural constraints nor lexical constraints.

4.6 Modified parsing schema

The last section gave a characterisation of grammars that have a polynomial fixed recognition problem. This section contains an exploration of various possibilities to restrict the factors even further in order to make the *general* recognition problem polynomial. However, this goal turns out to be quite difficult to achieve on an abstract level, hence the characterisation of theories that are polynomial in both n and $|G|$ will be rather informal.

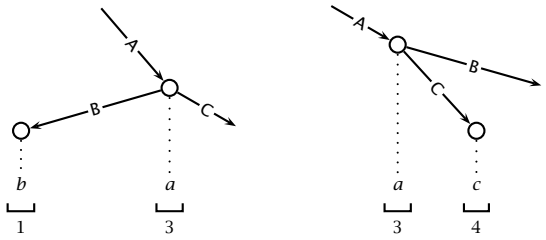
4.6.1 Saturations

The main idea is to remove the exponent k from the worst-case time complexity of the parsing schema above. As the exponent comes from the maximal number of premises in the combination rule, a shorter version of that rule is required which combines only a constant number of items at once. We will therefore introduce an updated schema, which combines only two items per step, filling open valencies in Ω one by one. This implies the derivation of intermediate, partly saturated items. The process of filling all open valencies of a particular initial item in m combination steps is called a *saturation*.[7]

The rules While the LOOKUP rule remains unchanged, the COMBINE rule is reduced to only two premises:

$$\frac{s_1 : \langle I_1, \Omega_1 \uplus I_2 \rangle \qquad s_2 : \langle I_2, \emptyset \rangle}{s_1 \oplus s_2 : \langle I_1, \Omega_1 \rangle} \text{ COMBINE}$$

[7] In the previous schema, a saturation was identical to one combination step.



$\{1,3\} : \langle\{A\},\{C\}\rangle$ $\{3,4\} : \langle\{A\},\{B\}\rangle$

Figure 4.2: Two intermediate items derived by COMBINE, and the partial drawings they stand for.

For each lexical entry, this rule is instantiated multiple times, such that I_1 is the type description, and $\Omega_1 \uplus I_2$ is a subset of the valency description, containing the valencies that still need to be saturated. Again, I_2 may not be empty.

The parsing schema is sound and complete, since it derives the same fully saturated parse items as before. An actual implementation works a bit differently: the parser takes an item from the agenda and searches for exactly one matching item in the chart. Thus it needs to traverse the chart only once per item.

Example We consider the parsing example from the last section again. Instead of filling all valencies at once, the modified parser now at first combines the item for *a* with the saturated *B*-item (shown on the left in Figure 4.2) and also combines the item for *a* with the saturated *C*-item (shown on the right). Both intermediate items are then saturated with the respective missing valency, resulting in the same fully saturated item as before (Figure 4.1). ⊣

Time complexity With Lemma 4.1, we can estimate the time complexity from the number of COMBINE rule instantiations. Variable parts are the two spans s_1 and s_2, as well as the sets I_1, I_2, and Ω_1, which together contain at most $k+1$ edge labels. Since the number of possible values for $\Omega_1 \uplus I_2$ is $\|\Omega\|$, an upper bound for the number of possible rule instantiations is $\|R\| \leq \|s\|^2 \cdot \|I\| \cdot \|\Omega\|$. Using the modified schema, parsing complexity for the most general LCG theory is thus in $O((2^n)^2 \cdot |G|^{k+1} \cdot c)$.

Computational factors Although the number of combined items is now constant, the value k still appears in the exponent due to the number of possible valency subsets. We can therefore identify three computational factors: a) the representation of valency subsets, b) the number of possible spans, and c) the constraint checking complexity. As we will see, the latter two cannot simply be restricted as before when using step-by-step saturation.

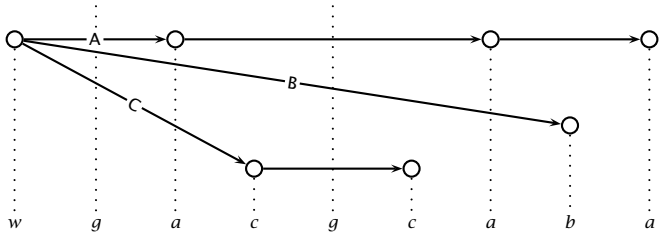


Figure 4.3: A gap 2 drawing for which span restriction results in incompleteness.

4.6.2 Restricting span representations

For theories that are restricted to a gap degree g, one could try to represent spans as tuples again. During a saturation process, however, the parser might derive intermediate items that have more than g gaps. The spans of these intermediate items therefore cannot be represented as tuples.

For example, we consider a theory over drawings with a gap degree restricted to 2, and a grammar that licenses the drawing in Figure 4.3. Let us further assume that the parse items for the subdrawings under A, B, and C have already been derived, and are about to be combined one by one with an unsaturated item for the node w. Regardless of the order in which the open valencies of w are filled, the parser always derives an intermediate item with more than two gaps. If spans were representated as pairs, the parsing schema would become incomplete.

Nevertheless, there are classes of drawings for which there is always an appropriate saturation order, such that the spans can be represented as tuples with $2(g+1)$ integers without losing completeness.

Projective drawings In projective theories, the parser can be restricted to intermediate items with continuous spans: starting with the unsaturated lexical item containing only the word for a node, the parser is able to subsequently add the next adjacent parse item representing a projective child drawing. Since intermediate results remain projective, we can use integer pairs for the spans.

Gap-restricted, well-nested drawings As shown above, the present parsing schema becomes incomplete for theories over $\mathcal{D}_g$ if the spans are represented as $2(g+1)$ tuples. At least for well-nested, gap-restricted drawings, however, there is another parsing schema which allows gap restrictions on spans while maintaining completeness [9]. It extends the schema presented here by an additional rule for groupings of child items. A group is essentially a parse item that stands for a forest of saturated drawings, i.e. has more than one ingoing edge. With the COMBINE rule, a group item is 'plugged' into the parent item in one step. For example, the well-nested drawing in Figure 4.3 can be parsed by first grouping the A- and the B-item (resulting in a gap 1 item), then combin-

ing the group with the item for the word w (gap 2), and finally adding the C-item (gap 2). The schema shows that the well-nestedness property also gives computational advantages. In general, however, the complexity of that schema is more difficult to analyse, especially with respect to lexical constraints.

Arbitrary gap-restricted drawings For gap-restricted theories in general, gaps in intermediate items may not be limited by a constant, even if the plugging rule is used. As an example, consider a drawing with the projection $w\ a\ b\ c\ d\ e\ b\ d\ a\ e\ c$, where identical labels indicate nodes belonging to the same subdrawing, similar to the sample drawing above. The drawing is a gap 1 drawing that is not well-nested. There is no possible way to combine any two such subdrawings without exceeding the gap degree of 1.

Conclusion For theories over a gap-restricted, well-nested class of drawings $\mathcal{D}_{g,wn}$, we can use a slightly modified parsing schema to represent spans as tuples. The number of different spans is then polynomial in n: $\|s\| \leq n^{2(g+1)}$.

4.6.3 Constraint checking

In general, the lexical constraints cannot be checked before an item has been fully saturated. Moreover, a constraint checking algorithm might need to access all items that were involved during a saturation process. Even if only the yields of the contributing items are checked, the parser needs to save a combination history at each item during the process of parsing. As soon as an item is saturated, the constraint checker could then traverse the history of the item to retrieve all items that it was constructed of. The problem is that a saturated item may be derived from more than one set of parse items. Although the union of the item set is always the same, there may be different possibilities how the spans of each item contribute to that union.

To see this more clearly, consider the item $(7,7) : \langle \emptyset, \{A, B\} \rangle$ and the following two pairs of saturated items:

1. $(1,3) : \langle \{A\}, \emptyset \rangle \quad (4,6) : \langle \{B\}, \emptyset \rangle$
2. $(1,2) : \langle \{B\}, \emptyset \rangle \quad (3,6) : \langle \{A\}, \emptyset \rangle$

The initial item may be combined with the first two saturated items, as well as with the last two items. Both saturations derive the same item $(1,7) : \langle \emptyset, \emptyset \rangle$, though only one or neither saturation might be licensed by the constraints. The constraint verifier therefore has to review the entire history of the derived saturated item to check whether the item is valid at all.

In the worst case, there may occur all kinds of spans for the k valencies, resulting in a history with $(2^n)^k$ entries which have to be verified by the constraint checker. Therefore, we get $c \in O(\|s\|^k)$.[8]

[8] In the previous schema, the exponent k was hidden within the expensive search for items to combine.

Constraint complexity can be improved if constraint checking does not rely on a combination history. We will only give a rather informal characterisation of this type of constraints: each constraint must be decidable at the level of saturation steps, i.e. by simply looking at the one item that represents all already combined items, and one new item that fills a valency. If constraint verifying in this fashion is sound and complete, constraint complexity is linear in the size of the parse items in the premises, i.e. in $O(2n + k + 1) = O(n + k)$. For gap-restricted spans, we even improve to $c \in O(2 \cdot 2(g + 1) + k + 1) = O(k)$. The next chapter presents precedence and gap constraints that can be verified after each saturation step.

4.6.4 Improving the analysis

At this point, we can put the results from above together. As shown above, a theory T over a well-nested and gap-restricted class of drawings can be parsed using a tuple representation of spans. If the semantics of all the lexical constraints in T can be defined at the level of saturation steps, they can be verified efficiently. The resulting time complexity is thus in

$$O(\underbrace{(n^{2(g+1)})}_{\|s\|}^2 \cdot \underbrace{|G|}_{\|I\|} \cdot \underbrace{|G|^k}_{\|\Omega\|} \cdot \underbrace{k}_{c}) = O(n^{4(g+1)} \cdot |G|^{k+1} \cdot k)$$

Assuming $|G|$ and k to be constant, we get $O(n^{4(g+1)})$. For context-free grammars, where $g = 0$, parsing complexity is in $O(n^4)$. For TAG grammars, which have a gap degree of 1, we currently get $O(n^8)$. The next chapter shows how to make use of constraints directly within the combination rule to achieve the well-known $O(n^3)$ and $O(n^6)$ results.

4.6.5 Valency lists

The parsing schema above represents Ω as valency sets, which allows for saturations in any combination order. If we leave spans in their most general form (sets) and do not require special constraints, it can be easily seen that the parser performs each saturation in all possible combination orders. This is unnecessary, as the derived fully saturated item is always the same, and the intermediate items are not required for any other saturation.

We can therefore restrict saturation to one arbitrary, but fixed combination order. More precisely, we define for each lexical entry a list containing the outgoing valencies, which determines the order in which these valencies should be filled. (Note that this combination order in general does not tell anything about the precedence of the valencies.) This allows for a representation of open valencies as lists, with the rules changed accordingly.

For example, we define the valency list $[C, B]$ for the lexical entry for node a in Figure 4.1. The LOOKUP rule is instantiated for that entry as follows:

$$\frac{\mathsf{pos}(i) = a \qquad \langle \{A\}, \{B, C\}\,; \emptyset \rangle \in Lex(a)}{\{i\} : \langle [A], [C, B] \rangle}\ \text{LOOKUP}$$

The COMBINE rules are changed to combine always the first element in Ω with a matching saturated item. The instantiations for the sample entry are:

$$\frac{s_1 : \langle [A], [C, B] \rangle \qquad s_2 : \langle [C], [] \rangle}{s_1 \oplus s_2 : \langle [A], [B] \rangle}\ \text{COMBINE} \qquad \frac{s_1 : \langle [A], [B] \rangle \qquad s_2 : \langle [B], [] \rangle}{s_1 \oplus s_2 : \langle [A], [] \rangle}\ \text{COMBINE}$$

When using a fixed combination order, the value Ω is always a sublist of a valency list. It can thus be represented with two indices: one specifying the lexical entry the valency list is associated with, and one determining the start position in the list.[9] Since the number of possible values for both indices is limited by $|G|$, we get an upper bound of $\|\Omega\| \leq |G|^2$.

For particular cases, it might be necessary to allow additional combination orders for a lexical entry. In this case, the number of valency lists is increasing, and so is the number of possible sublists. If all permutations are allowed as possible orders, the number of lists is exponential in k, hence we get the same result as when using a set representation for Ω.

4.6.6 Polynomial general recognition problem

To give a condition for grammars with a polynomial general recognition problem, we need to find a way to achieve polynomiality with respect to both n and $|G|$, i.e. to combine the restrictions on factors above.

As shown above, one can use an efficient representation of spans for well-nested gap-restricted drawings, but in order to remain complete, all combination orders must be allowed. The case is even more complicated for constraints: to get linear time constraint checkers, constraints must be sufficiently decidable at each saturation step, and even then a possibly exponential number of combination orders is needed to maintain completeness. Moreover, it depends on each constraint whether its semantics can be transferred to the level of saturation steps in the first place. As a result, it is very difficult to give abstract conditions for an LCG grammar with a polynomial general recognition problem.

We therefore conclude exploration on this abstract level, and turn to concrete theories with precedence and gap constraints. The next chapter introduces these constraints, and shows how they can be built into the parsing schema so that additional constraint checking becomes obsolete. Furthermore, information from these constraints can be used to obtain exactly those combination orders for which only gap-restricted intermediate items are produced, so that spans are efficiently representable. Finally, there are

[9] This approach is similar to the 'dotted rules' in Earley-style parsers.

particular gap-restricted grammars where the number of combination orders is bounded by a constant. In these grammars, the number of valency representations is polynomial in $|G|$, hence the general recognition problem is polynomial.

4.7 Comparison with other parsers

In this section, we analyse how parsers for specific formalisms relate to the parsing schemata for the respective LCG encodings, and how the other parsers make use of certain properties in their formalisms to improve efficiency.

Earley parser

The parser for context-free grammars developed by Earley [6] processes a sentence from left to right. In each parsing cycle, the parser performs the three operations of scanning, predicting and completing. For lexicalised grammars, the Earley parser can be changed so that it resembles an instantiation of the parsing schema above: the parsing process would start by predicting all rules with a terminal that occurs in the sentence, which makes later prediction obsolete. In the parsing schema, an initial item already stands for a word, whose valencies are to be filled. In constrast, an inital inactive edge in the Earley parser covers nothing. Its valencies, including the word, are then saturated one-by-one in a fixed combination order. ('Saturation' of the word happens in the scan step.) Since each inactive edge has a fixed combination order, the parser can make use of dotted rules, similar to our valency lists for Ω. The combination order followed by the parser is actually the precedence of valencies defined in the grammar rules. By ensuring that two combined edges are adjacent, with the inactive edge to the right, the parser automatically satisfies the linear order given in the grammar. As all intermediate inactive edges are projective, spans can be efficiently represented as pairs of integers, exactly as described above. Therefore, all conditions are fulfilled for the time complexity to be polynomial in both n and $|G|$.

LSL parser by Suhre

The parser for Linear Specification Grammars given by Suhre [18] is an extension of the Earley parser. Since parse trees in the formalism have no gap restriction, item spans are represented as bit vectors. Prediction takes place only in the beginning, where all rules are predicted at once. Suhre's parser also performs scanning from left to right, which helps verifying precedence constraints at an early stage. However, the parser has to store the spans of the items that have already been combinined so that a constraint checker can verify them when needed. Constraint complexity is thus exponential in k.[10] The

[10] Actually, it is sufficient for LSL constraint checking to store only the right-most span boundary of the already saturated valencies. Still, there may be k of these valencies.

parser scans the sentence from left to right and completes after each scanned word. In the end, however, each valid saturation has been performed in all possible combination orders, which is exponential in k. At the cost of possibly later constraint checking, one could define an arbitrary but fixed combination order for each grammar rule: the parser would not become incomplete, since spans are not restricted and the full combination histories are stored anyway.

Suhre points out that parsing becomes polynomial with respect to n if all nonterminals are *m-isolated*, i.e. if the spans of the constituents contain at most $m - 1$ gaps. Spans can then be represented efficiently with tuples. We will call the LSL formalism with that restriction m-LSL.

Suhre even shows shows that the gap restriction only needs to be applied to *recursive* nonterminals. However, he does not explain how to avoid intermediate items with more than $m - 1$ gaps. With the first parsing schema presented in this section, nevertheless, m-LSL can be parsed in time polynomial in n, though the exponent then depends on $|G|$.

LSL parser by Daniels and Meurers

Daniels and Meurers propose a different implementation of a parser for LSL grammars [4]. Instead of traversing a sentence from left to right, the entire sentence is scanned at first, and the chart is filled with initial items that correspond to the lexical entries (preterminal rules). This is followed by a prediction/completion cycle. Note that for lexicalised LSL grammars, prediction is unnecessary: since the occuring terminals in the sentence limit the set of applicable rules, these rules can be added to the chart altogether in the beginning.

As items may be non-projective, spans are again represented as bit vectors. The parser combines always two items at once, in an order specified in the grammar. In order to verify the constraints, the parser saves references to the combined origin items. There can be two items that only differ in their history, thus retrieving all matching items for completion is exponential in k.

The parser by Daniels and Meurers includes a lot of small optimisation techniques to avoid unnecessary completion steps; for example, the parser does not create an item if it detects that a precedence constraint for an open valency cannot be fulfilled because there is no space left for it in the span. Although these optimisations surely improve parsing efficiency in practical environments, they obstruct theoretical insights into complexity aspects of their parser.

4.8 Complexity matrix

As we have seen in Chapter 2, it is possible to encode a variety of formalisms into LCG. With the results from the complexity analyses, we can now position these formalisms in a matrix according to their recognition time complexity, as shown in Table 4.1.

	exponential in n	polynomial in n
exponential in $\|G\|$	$T_{\mathcal{X}}$	LUCFG, m-LSL, $T_{\emptyset}$
polynomial in $\|G\|$	LUCFG, LSL, m-LSL, $T_{\emptyset}$	LCFG

Table 4.1: Several formalisms positioned in a complexity matrix.

In the matrix, $T_{\mathcal{X}}$ stands for any theory for which a constraint checker needs time exponential in $|G|$ and n, e.g. because it needs to verify entire subdrawings. The theory $T_{\emptyset}$ is the theory without constraints: $(\mathcal{D}_*, \mathcal{L}_{\emptyset})$. As we have seen, it is either exponential in n or in $|G|$. Both m-LSL and LUCFG are gap-restricted and have constraints that only refer to the yields of subdrawings. They are therefore polynomial with respect to the length of the sentence. As shown above, these theories can also be recognised in time polynomial in $|G|$ and exponential in n, and can thus also be found in the lower left field. Since the gap-restriction is not required for theories that are polynomial in $|G|$, 'pure' LSL can also be found in that class. Finally, the fact that LCFG is polynomial with respect to both dimensions is shown in the next chapter using an efficient specialised parsing schema.

Note that the analysed (rough) complexity of an LCG encoding of a formalism corresponds to the complexity classes of that formalism. More precisely, the formalisms in the right column all have a polynomial fixed recognition problem, while it is NP-hard for the others. Furthermore, the formalisms in the lower right field have a polynomial general recognition problem; for the others, it is in NP.

Chapter 5

Precedence and Gap Constraints

The last chapter presented various possibilities to improve parsing efficiency. This chapter shows how to improve efficiency even further for concrete theories with precedence and gap constraints. These constraints can be seen as descriptions of local valency linearisations which can be built directly into the parsing rules, making external constraint checking obsolete.

The concept of linearisations is at first presented for projective theories, and later generalised to valency part linearisations for non-projective theories. In the end, it is shown how LSL and TAG grammars can be encoded into an LCG theory with valency part linearisations.

5.1 Projective valency linearisations

The *projective valency linearisation theory*, VLG_0 is a theory $(\mathcal{D}_0, \mathcal{L}_{\{\prec\}})$, where the semantics of the valency precedence constraint $\prec$ is defined as follows:

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

Section 2.3.1 already demonstrated that VLG_0 can be used to encode lexicalised context-free grammars (LCFGs). Figure 5.1 shows again the sample LCFG rule and its encoding as a lexical entry. The entry can be seen as a local description of a possible configuration for the grammar: The node must be marked with the terminal symbol, the nonterminal on the left hand side becomes the type of the node, and the nonterminals on the right hand side must appear on the outgoing edges. The lexical constraints are chosen so that they license only those drawings in which the order of the outgoing edges and the node itself exactly matches the order given by the original LCFG rule.

$$A \rightarrow BaCD$$

$$a \mapsto \langle \{A\}, \{B, C, D\} ; \{B \prec \iota, \iota \prec C, C \prec D\} \rangle$$

Figure 5.1: A sample encoding of an LCFG rule.

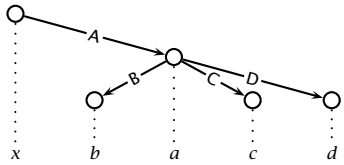


Figure 5.2: A sample drawing. Node a satisfies the linearisation $(BaCD)$.

Linearisations A *linearisation* is a sequence that gives a possible local order for the outgoing edges and the node itself. A linearisation is *well-typed* for a lexical entry if it contains the word and each valency declared in the entry exactly once. It is *licensed* by a lexical entry if the sequence satisfies the precedence constraints in the entry. For example, the constraints in Figure 5.1 license the linearisation $(BaCD)$. The theory does not require the constraints to describe unique linearisations: for example, if we took the constraints $\{B \prec \iota, \iota \prec C, \iota \prec D\}$ instead, they would license the two linearisations $(BaCD)$ and $(BaDC)$.

Since licensed linearisations for an entry are simply the solutions of possibly underspecified precedence descriptions, they allow us to take an extensional perspective on precedence constraints. More precisely, each lexical entry can be implicitly associated with the set of all linearisations that are licensed by the constraints. A node then satisfies the precedence constraints in a lexical entry if and only if it is licensed by one of the linearisations associated with the entry, i.e. its successors and the node itself are ordered accordingly. For example, node a in the drawing in Figure 5.2 satisfies the constraints in our sample entry because it is licensed by the linearisation $(BaCD)$.

We call a grammar in which each entry is associated with exactly one linearisation an *ordered context-free grammar*. In constrast, an *unordered context-free grammar* is encoded by not using any constraints in the entries. In this case, the set of linearisations at each lexical entry are all permutations of the word and the valencies of the respective lexical entry. Between both grammar classes, there is a wide range of grammars with partly specified linearisations.

Strict locality The global projectivity constraint from the class of drawings can be moved into the definition of linearisation satisfaction: Rather than ordering successor nodes only, a linearisation can be seen as a description of the order of entire subtrees. Moreover, we define that two labels in a linearisation appearing next to each other represent adjacent subtrees. Since this definition enforces projectivity, we can safely extend the theory to the class of arbitrary drawings, $\mathcal{D}_*$. As a result, the global structural properties of licensed drawings are described in a purely local fashion by the lexical entries: the type and valency descriptions determine the tree structure, while the linearisations at each node determine the total order of the nodes. It is not possible to have two nodes in a drawing whose linearisations contradict each other. Hence if the precedence constraints are locally consistent, they can always be satisfied.

$$\frac{\begin{array}{ccc} & (j,j) : \langle \{A\}, \{B,C,D\} \rangle & \\ (i,j-1) : \langle \{B\}, \emptyset \rangle & (j+1,k) : \langle \{C\}, \emptyset \rangle & (k+1,l) : \langle \{D\}, \emptyset \rangle \end{array}}{(i,l) : \langle A, \emptyset \rangle} \text{ COMBINE}$$

Figure 5.3: A parsing schema instantiation with built-in linearisation $(BaCD)$.

Parsing schema Instead of verifying after a completed saturation whether the item yields satisfy one of the linearisations in the set, one can incorporate each linearisation into the COMBINE rule by constraining the span indices. Figure 5.3 shows the instantiation of the combination rule for the sample entry with linearisation $(BaCD)$. It exploits the projectivity property: if two valencies are adjacent in the linearisation, the corresponding items for the subtrees must have adjacent spans.

Complexity analysis From left to right, each premise introduces one new span variable, while the other one is determined. Moreover, the first premise contains $k+1$ edge labels that reappear in the other premises. The maximal number of COMBINE rule instantiations is thus $|G|^{k+1} \cdot n^{k+1}$. Since there are no other constraints to check, the upper bound for the time complexity is $O(n^{k+1})$ if we assume a constant grammar size. Hence the fixed word problem for VLG_0 is polynomial.

However, the exponent in the polynomial still depends on k. VLG_0, nevertheless, allows for the encoding of ordered context-free grammars, which are known to be recognisable in $O(n^3)$. We will therefore transfer the concept of linearisations to the 'narrow' parsing schema, in which each saturation is performed by combining only two items at once. The possible order for the saturation steps can be retrieved from the linearisations. As it turns out, one combination order per linearisation is sufficient to recognise all drawings satisfying that linearisation.

Combination orders For each linearisation, there is a possible order for filling open valencies such that intermediate items are projective, and all valency adjacency conditions hold. Unfortunately, we cannot directly use the linearisation as the order in which the items are combined, as each saturation must start with the item for the single word.

Therefore, the order is retrieved from a linearisation by beginning with the item representing the word and combining it with the items to the right of it one by one, and then with the items to the left of it. For the linearisation $(BaCD)$, for example, the parser first combines the a-item with a C-item, ensuring that the C-item is immediately to the right. The combination results in a projective item, which we will write as $(a + C)$. This item is combined with a D-item which must be to the right, resulting in $((a + C) + D)$. As there are no more items to the right, the parser continues with the B-item to the left, which is also the last item. A combination order for the linearisation $(BaCD)$ is thus

$$\dfrac{\dfrac{\dfrac{\dfrac{\mathsf{pos}(j) = a \qquad \langle \{A\}, \{B, C, D\}\,; \Phi \rangle \in \mathit{Lex}(a)}{(j, j) : \langle [A], [C, D, B] \rangle}\ \textsc{lookup} \quad \vdots \quad (j+1, k) : \langle [C], [] \rangle}{(j, k) : \langle [A], [D, B] \rangle}\ \textsc{combine} \quad \vdots \quad (k+1, l) : \langle [D], [] \rangle}{(j, l) : \langle [A], [B] \rangle}\ \textsc{combine} \quad \vdots \quad (i, j-1) : \langle [B], [] \rangle}{(i, l) : \langle [A], [] \rangle}\ \textsc{combine}$$

Figure 5.4: Step-by-step saturation of open valencies.

$((a + C) + D) + B$. Actually, there are other valid combination orders for that linearisation, such as $((a + C) + B) + D$). However, the result is the same for each suitable order, therefore only one of them needs to be considered.

Figure 5.4 shows how to compile the order into a LOOKUP rule and three 'narrow' COMBINE rules. The values of Ω are represented as sublists of the combination order list $[C, D, B]$. The entire list is introduced by LOOKUP. The three COMBINE rules then process the list by removing always the first valency and matching it with an appropriate item, at the same time ensuring the span adjacency conditions. The inference tree shows that using these rules, the same item can be derived from the same premises as in the previous schema.

Improved time complexity As mentioned above, the precedence constraints in a lexical entry describe a set of linearisations. Each linearisation can be translated into a combination order, which in turn can be seen as a description of a set of combination rules. All of these rules deal with projective items only, thus each span can be represented as a pair of two integers between 1 and n. Moreover, each rule combines two adjacent spans, hence at least one of the four span variables in the premises is always determined.

The value of Ω of the left premise can be represented by two values: one containing the combination order, and one containing the current step of combination. As each order corresponds to a linearisation, the number of combination orders is the number of linearisations. The maximal number of steps during a saturation is k. Since each linearisation is associated with a lexical entry, the incoming edge label I of the left premise is determined. The first label in Ω also determines the incoming edge of the

right premise. In conclusion, the maximal number of rule instantiations is

$$\underbrace{n^3}_{\text{span variables}} \cdot \underbrace{k \cdot \|L\|}_{\text{valency variables}}$$

where $\|L\|$ is the number of linearisations in the grammar. As there are no additional constraints to check, this is also the maximal time complexity for a parser of these types of grammars.

The formula can be used to obtain familiar complexity results. In ordered context-free grammars, there is one linearisation per entry, thus an upper bound for $\|L\|$ is $|G|$. Approximating k with $|G|$, the time complexity is therefore in $O(|G|^2 \cdot n^3)$. For unordered context-free grammars, the number of linearisations is the number of valency permutations of each entry, resulting in an overall complexity of $O(2^{|G|} \cdot n^3)$.

5.2 Non-projective valency linearisations

We will now generalise the results of the previous section to non-projective, gap-restricted LCGs, where constraints describe *linearisations of possibly non-projective valencies*.

5.2.1 Valency linearisation theory

In non-projective theories, drawings may have discontinuous yields with gaps. These gaps partition the set of nodes belonging to the drawings. As defined in Chapter 2, parts are maximal intervals that are numbered from the left, starting with zero.

Instead of ordering entire subdrawings by positioning their entire yields, the precedence constraints in the following theory order the parts of the yields of the local subdrawings. Furthermore, the theory contains gap constraints which determine the number of parts of a subdrawing. In the following, we will simply refer to *valency parts* by identifying a valency with the yield of the corresponding subdrawing.

For a node u, the notion *local drawing* refers to the entire drawing that is rooted at the node u. The local drawing itself might consist of several parts with gaps in between them. The precedence constraints therefore do not only need to order the valency parts and the node itself, but also its gaps. To simplify the definition of the constraints, we assume a special label $\sqcup$, the *gap valency*, as already introduced in Section 2.3.4. It can be used to access the set of nodes in the gaps of u:

$$\diamond_{\sqcup} u := C(\triangleleft^* u) - \triangleleft^* u$$

This set may again have several parts: the first part $[\diamond_{\sqcup} u]^0$ contains the nodes in the left-most gap of u, the second part contains the nodes in the second gap of u, and so on. We can thereby position the gaps of node u simply by ordering the parts of the gap valency.

$$\begin{aligned}
\mathfrak{D}, u &\vDash \pi_1^i \prec \pi_2^j && \text{iff} \quad [\diamond_{\pi_1} u]^i \times [\diamond_{\pi_2} u]^j \subseteq \prec \\
\mathfrak{D}, u &\vDash \|\pi\|^i && \text{iff} \quad i = \mathsf{gd}(\diamond_\pi u) \\
\mathfrak{D}, u &\vDash \|\bullet\|^i && \text{iff} \quad i = \mathsf{gd}(\lhd^* u)
\end{aligned}$$

Figure 5.5: The three valency part constraints.

A *valency linearisation theory*, VLG in short, is a theory $(\mathcal{D}_g, \mathcal{L}_\mathcal{R})$ with a gap-restricted class of drawings and a constraint language as presented in Figure 5.5. Note that i and j are metavariables; the language actually consists of the constraints with i and j instantiated to all numbers between 0 and g.

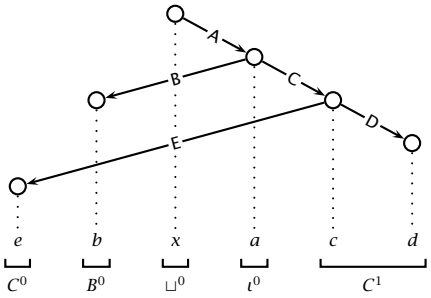
The constraints order the parts of the valencies and the gaps, and determine their numbers:

- $\pi_1^i \prec \pi_2^j$ means part i of π_1 must precede part j of π_2. If π_1 and/or π_2 are the gap valency ($\sqcup$), they refer to the respective gaps instead.
- $\|\pi\|^i$ expresses that the π-successor of u must have i gaps, i.e. it must consist of exactly $i + 1$ parts.
- $\|\bullet\|^i$ finally ensures that the local drawing of u must have a gap degree of i.[1]

Note that if a constraint refers to a part that does not exist, the constraint definition deals with empty node sets, which may lead to unwanted results. In VLG, it is in the responsibility of the writer of the grammar to ensure that the constraints 'do what they mean'. For example, it is syntactically possible to use the part ι^2 in a precedence constraint. However, the constraint is always satisfied, since $[\diamond_\iota u]^2$ is empty, as $\diamond_\iota u = \{u\}$ is projective.

Example As an example, consider the sample entry of a VLG grammar and the drawing in Figure 5.6. The node a in the drawing satisfies the given entry: $\|\bullet\|^1$ holds because the yield of a has one gap. $\sqcup^0$ refers to the nodes in that gap, namely x. Similarly $\|B\|^0$ holds, since the subtree below the edge marked with B is projective. C^0 refers to the left-most part of the valency C of node a; it consists of the single node e. The second part, C^1, consists of the nodes c and d. The valency B has only one part, consisting of the node b. The order of the parts of the subdrawing, of the node u and its gap satisfy the precedence constraints. ⊣

[1] We need this variant to express e.g. projectivity of the local drawing. This cannot be achieved with the gap constraint on the gap valency: something like $\|\sqcup\|^0$ means the *gap valency* has no gap, i.e. one part, which means that u has one gap.



$$a \mapsto \langle \{A\}, \{B, C\} ; \{\|\bullet\|^1, \|B\|^0, C^0 \prec B^0 \prec \sqcup^0 \prec \iota^0 \prec C^1\} \rangle$$

Figure 5.6: A sample VLG entry, and a drawing with a node a satisfying the entry.

Valency part linearisations

Similarly to precedence constraints in projective grammars above, valency part constraints can be seen as descriptions of local linearisations of possibly non-projective valencies, also called *valency part linearisations*. They contain the valency parts in the form π^i, as well as the word symbolising the node itself. In constrast to linearisations before, a valency part linearisation is not merely a sequence, but a tuple with a global order, which effectively partitions the sequence. The first component of the tuple specifies the contents of the first part of the local drawing, the second component contains the second part of the local drawing, and so on. The position of the component boundaries (the 'commas') in the tuple thus indicate the positions of the gaps, i.e. the parts of the gap valency.

More formally, a valency part linearisation is *well-typed* for a lexical entry if it has at most $g + 1$ components, if all components are non-empty, and if for each valency π, there is a part number $g(\pi)$ such that following holds:

- the linearisation contains the parts $\pi^0, \pi^1, \cdots, \pi^{g(\pi)}$ exactly once and no other part of π,
- no two parts are adjacent in the linearisation, and
- for all i with $0 < i \leq g(\pi)$, π^{i-1} precedes π^i.

A valency part linearisation *satisfies* a lexical entry

- if its number of components and the number of the valency parts are licensed by the respective gap constraints ($\|\pi\|^i$ and $\|\bullet\|^i$),
- if the order of the valency parts and the word satisfy the precedence constraints ($\pi_1^i \prec \pi_2^j$), and

$$\frac{(l,l) : \langle \{A\}, \{B,C,D\} \rangle \qquad (j,k) : \langle \{B\}, \emptyset \rangle \qquad (i, j-1, l+1, m) : \langle \{C\}, \emptyset \rangle}{(i,k,l,m) : \langle A, \emptyset \rangle} \text{ COMBINE}$$

Figure 5.7: A parsing schema instantiation with built-in linearisation (C^0B^0, aC^1).

- if the position of the component boundaries satisfy the precedence constraints on gaps ($\pi^i \prec \sqcup^j$ and $\sqcup^i \prec \pi^j$).

For example, a valency part linearisation that satisfies the constraints in the sample entry is (C^0B^0, aC^1): the order of the valency parts and the word as well as the position of the gap is directly specified by the constraints in the entry. That linearisation is also the only one that is well-typed and licensed by the constraints: There must be two components ($\|\bullet\|^1$), the valency B has only one part ($\|B\|^0$), and there cannot be more parts of C: C^2 would have to appear after C^1, but in the same component, thus C^1 and C^2 would be adjacent.

Relation to projective valency linearisations It is easy to see that VLG_0 is equivalent to the VLG theory with $g = 0$: Each constraint is instantiated only once, as the metavariables i and j can only be 0. Since the maximal part number of a valency is 0, part 0 of a yield corresponds to the entire yield, i.e. $[\diamond_\pi u]^0 = \diamond_\pi u$. Moreover, $\|\pi\|^0$ as well as $\|\bullet\|^0$ always hold, which makes them obsolete. The gap valency is also useless, as there are no gap nodes to position. Finally, the remaining precedence constraint can be weakened to immediate successor nodes, because precedence of the entire subdrawings is enforced by projectivity.

Parsing schema Again, a linearisation can be built into the parsing schema by constraining the span variables, as shown in Figure 5.7. The required number of gaps for the valencies is reflected by the number of integers in the respective spans. Note that the notion of a 'valid' span ensures that the spans of the result and the C valency actually contain a gap, i.e. that $k < l + 1$ and $j - 1 < l + 1 + 1$.

Complexity analysis

In VLG, the gap degree is restricted, hence we can represent spans as tuples of integers. The constraints can be used to find all possible valency part linearisations, which in turn can be built into the parsing schema, making external constraint checking obsolete. As we have shown in the previous chapter, the time complexity for such a theory is in $O((n^{2(g+1)})^{k+1})$, ignoring the size of the grammar.

The formula corresponds to the number of integer variables in the spans in the premises of a COMBINE rule. However, some of these variables depend on the others,

as some spans are adjacent. Therefore, we can improve the estimate.

In each linearisation, there are $\|p\|$ valency parts. Each part p_i is represented by two integers in the schema, $p_{i,l}$ and $p_{i,r}$, which indicate the left and the right position of the sentence range that the part covers. In general, the right position is determined by the left position of the following, adjacent part in the linearisation: $p_{i,r} = p_{i+1,l} - 1$. The word itself can be seen as a part of length 1, though its both integers are determined: $p_{i,l} = p_{i,r} = p_{i+1,1} - 1$. Up to this point, we have counted $\|p\|$ independent variables, one per valency part. The only case in which the right boundary is not determined is when there is no following part, i.e. when the part is positioned at the end of one of the $g + 1$ components of the linearisation tuple. Including these special cases, there are $\|p\| + g + 1$ independent variables.

The number of parts $\|p\|$ in the premises can be approximated by the maximal number of valencies and their maximal part count: $\|p\| \leq k \cdot (g + 1)$. The upper bound for the number of independent span variables in the premises is therefore

$$\|p\| + g + 1 \leq k \cdot (g + 1) + g + 1 = (k + 1)(g + 1)$$

The number of COMBINE rule instantiations is thus in $O(n^{(k+1)(g+1)})$. As we have built all constraints into the parser, that is also the time complexity.

For well-nested drawings, this figure can still be improved. As we have seen, it is possible to use the 'narrow' parsing schema for these drawings, with only two premises per rule, each premise consisting of at most $g + 1$ parts. We can therefore give a better upper bound for the number of parts: $\|p\| \leq 2(g + 1)$. Using the same argumentation as above, we get a time complexity of $O(n^{3g+3})$. For context-free grammars, which are projective, this gives the familiar complexity of $O(n^3)$. TAG grammars have a gap degree of 1, hence we get the well-known result of $O(n^6)$.

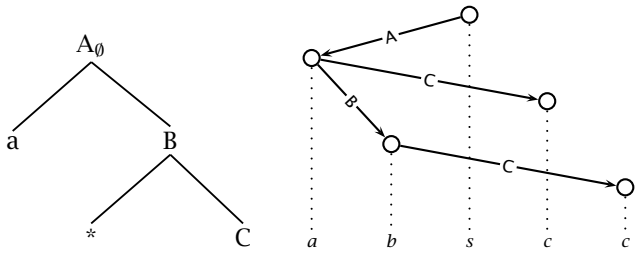
5.2.2 Sample encodings with linearisations

This section shows how two formalisms, m-LSL and TAG, can be encoded as a VLG theory, and how the grammar rules can be directly translated into linearisations.

Linear Specification Language

Let $G = (N, T, P, L, S)$ be a gap-restricted m-LSL grammar with a maximal gap degree of $m - 1$. Since G is lexicalised, we may assume that in each rule in P, exactly one of the preterminals is replaced by the terminal itself. We define a VLG theory with a gap degree of $m - 1$ and a VLG grammar $G' = (\Sigma, \Pi, Lex)$, where $\Sigma = T$ and $\Pi = N$.

For every grammar rule in $P \cup L$, we create all lexical entries that have the following properties: the word that is associated with the entry is the terminal on the right hand side, the valency description contains exactly the nonterminals on the right hand side, and the type description contains the nonterminal on the left hand side. If the left hand



(a) auxiliary tree (b) TAG drawing

Figure 5.8: A sample TAG tree and a TAG drawing. The node a in the drawing represents the tree.

side is the start symbol S, we also need to create a duplicate entry for root nodes in which the type description is empty. The set of linearisations contains all linearisations that fulfil the LSL constraints:

- If there is an isolation constraint for a valency, $\langle\pi\rangle$, there may only be one part of π in the linearisation, namely π^0.
- If the isolation constraint is applied on the left hand side, the linearisation may only consist of one component.
- The precedence constraint $\pi_1 < \pi_2$ requires that the last part of the π_1 valency must precede the first part of the π_2 valency in the linearisation.
- Immediate precedence $\pi_1 \ll \pi_2$ additionally requires these both parts to be adjacent.

For example, consider the following rule of an LSL grammar that is restricted to gap degree 1:

$$A \quad \rightarrow \quad a\ B\ C\ ; \quad \langle A\rangle, \langle C\rangle, a < C, B \ll C$$

It is translated into an entry that contains the two possible linearisations:

$$a \quad \mapsto \quad \langle \{A\}, \{B, C\}\ ; \{(aB^0C^0), (B^0aB^1C^0)\}\rangle$$

An encoding of Tree-Adjoining Grammars

For Tree Adjoining Grammars, we consider again the example from Section 2.3.4. Figure 5.8 shows the auxiliary tree for the valency A as well as a TAG drawing in which the node a stands for the tree. We can derive a lexical entry and the linearisations for that node directly from the tree:

- The node must be labelled with the anchor, a.

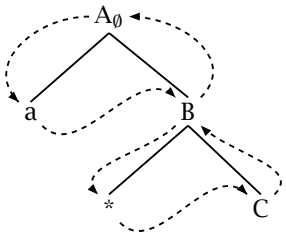


Figure 5.9: A path through the TAG tree that yields the possible linearisations.

- Since it stands for an auxiliary tree, the gap degree of the node must be 1. (Again, we exclude left and right adjunction from the encoding.)
- Its incoming edge is labelled with the root node, A.
- The outgoing edges represent the performed operations. While no adjunction is permitted at node A, there may be an outgoing edge labelled with B for the possible adjunction at node B. The B-successor in the drawing must have one gap, as it stands for an auxiliary tree. Also, there must be a projective C-successor for the required substitution at node C.
- The node itself precedes the gap, while the drawing below the C-edge must appear directly to the right of the gap.
- If adjunction takes place at B, the auxiliary tree for B is 'wrapped' around $*$ and C. In the TAG drawing, this means that the first part of B-valency must appear immediately before the gap, while the second part must appear right after the C-subdrawing.

As in the samples section, we have to create two entries to reflect the optionality of the adjunction at node B:

$$\begin{aligned} a &\mapsto \langle \{A\}, \{B, C\} ; \{(aB^0, C^0B^1)\}\rangle \\ a &\mapsto \langle \{A\}, \{C\} ; \{(a, C^0)\}\rangle \end{aligned}$$

In general, one needs one entry for each possible set points where adjunctions take place; each such choice determines the linearisation uniquely. Again, if the encoded tree has a start symbol as its root, we need to add for each entry another entry with an empty type description.

Figure 5.9 shows that the linearisations can be directly read off the tree: starting at the root, one walks over the tree using left-to-right depth-first traversal. Inner nodes are visited twice; if it is an adjunction point, we place the left part on the first visit and the right part on the second visit.

5.2.3 Globally consistent constraints

Unfortunately, VLG lacks an important property of VLG_0: globally consistent constraints. In VLG_0, it is possible to write down lexical entries with precedence constraints that are locally inconsistent and can thus never be fulfilled. These entries can be easily identified, as the set of possible linearisations is empty. Moreover, precedence constraints at two different nodes can never contradict each other, since they order entire subtrees, without constraining their internal order. In other words, assume there is a tree structure, marked with edge and node labels but without an order, and lexical entries for each node such that the type and valency descriptions are satisfied. Then there is always a (projective) drawing with that tree structure satisfying the constraints, as long as the constraints are locally consistent. This is because the constraints do not influence the tree structure, but merely restrict the possible total orders. In general, constraints for which this property holds are called *globally consistent*.

In VLG, locally inconsist constraints also lead to empty linearisation sets, and can thus be detected easily. However, there are locally consistent constraints which are not globally consistent. Consider, for example, the following two lexical entries[2]:

$$\begin{aligned} a &\mapsto \langle \emptyset, \{B\}\,; \{(B^0 a B^1)\}\rangle \\ b &\mapsto \langle \{B\}, \emptyset\,; \{(b)\}\rangle \end{aligned}$$

There is a possible labelled tree structure that satisfies the type and valency descriptions of both entries. That structure consists of a node a and a child node b, connected by an edge labelled with B. However, there is no drawing with that structure that satisfies the linearisations, because the first one requires its child to have a gap, while the second one states the gap degree of b is zero.

To avoid possibly globally inconsistencies, we 'compile' the gap requirements into the edge labels. More precisely, we translate the VLG grammar into another grammar in which the edge labels are extended to pairs, consisting of the original label and of a number that corresponds to the gap degree as required in the linearisation. The new lexicon for the example above looks like this:

$$\begin{aligned} a &\mapsto \langle \emptyset, \{(B,1)\}\,; \{(B^0 a B^1)\}\rangle \\ b &\mapsto \langle \{(B,0)\}, \emptyset\,; \{(b)\}\rangle \end{aligned}$$

Additionally, the edge label set Π is replaced by the extended set $\Pi' = \{\,(\pi, i) \mid \pi \in \Pi \text{ and } 0 \le i \le g\,\}$.

The situation as described above cannot occur anymore, since there is no tree structure that satisfies the entries in the first place: $(B,1)$ does not match $(B,0)$. Inversely, if a tree structure exists for a lexicon, then the respective linearisations can always be satisfied.

[2]For reasons of readability, we give the linearisations instead of the corresponding constraints here.

In general, every VLG grammar G can be translated into an extended VLG grammar G' that describes the same drawings as G, modulo the gap information on the edges. In this extended grammar, constraints are globally consistent, i.e. the description of the tree structure (type and valency descriptions) is separated from the description of the total order (linearisations), as it is the case in VLG_0. This property is important for the next chapter, which shows that VLG theories are mildly context-sensitive.

Chapter 6

Mildly Context-Sensitive Dependency Languages

Midly context-sensitive languages are a class of languages that are linguisticly relevant and at the same time efficiently recognisable. With the results and observations made in the previous chapters, it is easy to transfer that notion to sets of dependency structures, and to give a characterisation of mildly context-sensitive dependency grammars. The chapter concludes with a short proof that VLG, the class of valency part linearisation theories from the previous chapter, is a theory for mildly context-sensitive LCG grammars.

6.1 Mildly context-sensitive languages

Mildly context-sensitive languages form an important proper set between context-free and context-sensitive languages in the Chomsky hierarchy. They are introduced to allow for descriptions of natural languages in a linguistically significant manner, while being only 'slightly' more powerful than context-free languages.

Currently, there exists no exact characterisation of mild context-sensitivity. The most important, though somewhat informal conditions are those given by Joshi [11]. He proposes that a language L is mildly context-sensitive if it has the following three properties:

1. the membership problem for L can be decided in polynomial time,
2. L has the constant growth property, and
3. the number of cross-serial dependencies is limited by a constant.

Polynomiality Each mildly context-sensitive language L is a PTIME language. As mentioned in Section 3.1, this is the case if L is generated by a grammar that has a polynomial fixed recognition problem.

Constant growth property A language L has the constant growth property if it is either finite or if there is a constant $c \in \mathbb{N}$ such that for each sentence in L with a length of n, there is a longer sentence in L with a length of at most $n + c$.

Cross-serial dependencies A cross-serial dependency arises if there are overlapping dependencies between parts of a sentence. It is not quite clear what the formal aspects of a cross-serial dependency are. In general, linguists agree on the idea that information is shared between at least two non-adjacent parts of the sentence, with independent material in between. Kracht [13] suggests that the condition thus requires a limited maximal number of non-adjacent parts that are dependent among each other, i.e. that the number of discontinuities of any constituent is limited by a constant.

Examples Grammar formalisms that generate mildly context-sensitive languages are for example Tree Adjoining Grammars (TAG), Combinatory Categorial Grammars (CCG), Linear Indexed Grammars (LIG) and Head Grammars (HG), which are all weakly equivalent. All these languages are expressible as Linear Context-Free Rewriting Systems (LCFRS) [19]. LCFRS system are in general, under certain restrictions, mildly context-sensitive.

6.2 Semilinear projections

Before we analyse how the conditions for mild context-sensitivity can be transferred to sets of dependency structures, we examine a common specialisation of constant growth, known as semilinearity, and show how it applies to dependency languages.

Semilinearity Originally, semilinearity is a property of vector sets. By abstracting away the word order, languages can also be seen as sets of vectors.

Definition 6.1 Let L be a language over some alphabet Σ. The *Parikh image* of L is a function that associates each sentence in L with a vector in $\mathbb{N}^{|\Sigma|}$, such that each position in the vector contains the number of occurrences of a particular word in the sentence. The elements of a Parikh image are called *Parikh vectors*. ⊣

An example for Parikh vectors was already given in Section 4.5: Given the alphabet $\Sigma = \{\text{loves, Mary, Peter}\}$, each sentence that contains two times the word 'loves', one time 'Mary' and nothing more has the Parikh vector $(2, 1, 0)$.

Definition 6.2 Let M be a set of n-vectors of natural numbers, i.e. $M \subseteq \mathbb{N}^n$. We call M *linear* if there are vectors $\vec{u}^{(0)}, \vec{u}^{(1)}, \ldots, \vec{u}^{(k)} \in \mathbb{N}^n$ such that each vector in M is a linear combination of them:

$$M = \{\, \vec{u}^{(0)} + \sum_{i=1}^{k} n_i \cdot \vec{u}^{(i)} \mid n_1, \ldots, n_k \in \mathbb{N} \,\}$$

A set is called *semilinear* if it is a finite union of linear sets. A language L is called *semilinear* if and only if its Parikh image is semilinear. ⊣

The following important result is also known as the *Parikh theorem*. A proof is given by Kracht [13].

Theorem 6.3 Every context-free language is semilinear. □

Since this means that the Parikh image of each context-free language is semilinear, we get the following corollary:

Corollary 6.4 Every language that has the same Parikh image as a context-free language is semilinear.[1] □

It is easy to show that semilinearity is a specialisation of constant growth.

Lemma 6.5 A semilinear language has the constant growth property. □

PROOF We define the size of a vector $\vec{v}$ of natural numbers, $|\vec{v}|$, as the sum of all component numbers in the vector. Let L be a semilinear language, and let $\vec{u}$ be a base vector of the (semilinear) Parikh image of L with $|\vec{u}| > 0$.[2] Given a sentence in L, its vector $\vec{v}$ is in the Parikh image of L. Then the vector $\vec{w} = \vec{v} + \vec{u}$ is also in the Parikh image, which means that there is a sentence in L that has $\vec{w}$ as its image and is exactly $|\vec{u}|$ words larger than the original sentence. Thus for every sentence in L, there is another sentence that is exactly $|\vec{u}|$ words larger. Hence the constant growth property holds. ∎

Regular tree languages A *regular tree language* is a set of trees that can be derived from a regular tree grammar. A *regular tree grammar* is a 4-tuple (N, T, P, S) such that

- N is a set of nonterminal symbols,
- T is a set of terminal symbols,
- P is a set of production rules and
- $S \subseteq N$ is a set of start symbols.

A production rule has the form

$$A \to f(R_1, R_2, \ldots, R_n)$$

where $A, R_1, \ldots, R_n \in N$ and $f \in T$. The trees produced by a such a rule have a root node labelled f, an incoming edge labelled A, and outgoing edges labelled $R_1, \ldots, R_n$, under which there are subtrees produced by the appropriate rules. The trees generated by a regular tree grammar are those trees generated by rules with a start symbol on the left hand side. Note that in the non-standard definition given here, the order of the nonterminals $R_1, \ldots, R_n$ is irrelevant, hence the resulting trees are unordered.

[1] Note that two languages with the same Parikh image are also called *letter-equivalent*. In order to avoid confusion between the notions of symbols, letters, words, and sentences, we will not use that term at all.

[2] If all base vectors in any linear subset are null vectors, the Parikh image is finite and so is the language. In this case, L has the constant growth property.

Regular tree projection languages are semilinear Let G be a regular tree grammar. A *projection language* of G, written π_G, is a mapping from each tree that can be derived from G to one or more projection strings, i.e. strings that are obtained by putting the nodes of the tree in an arbitrary, but fixed order and then concatenating the node labels in that order. As projection languages do not "forget" any trees or nodes, it is clear that two projection languages of the same tree grammar have the same Parikh image.

Lemma 6.6 Let G be a regular tree grammar, and π_G be a projection language of G. Then π_G is semilinear. □

PROOF We construct a lexicalised context-free grammar G' from the regular tree grammar G: for each rule in G, there is a corresponding ordered rule in G' such that the right hand side contains at first the node label, and then the outgoing edge labels in an arbitrary but fixed order. By construction of the grammar, its (string) language $L(G')$ is a projection language of G.[3] Hence $L(G')$ and π_G have the same Parikh image. As $L(G')$ is a context-free language, it follows from Corollary 6.4 that π_G is semilinear. ∎

Semilinear dependency languages and grammars We are now able to characterise semilinear dependency languages: a dependency language is semilinear if the trees that underlie the dependency structures form a regular tree language.

A dependency *grammar* is therefore semilinear if it describes a set of regular trees and a projection language for them. This is the case if the grammar defines the tree structure and their possible projections separately such that there is at least one projection for each tree.

In LCG, this separation is reflected by the type and valency descriptions on the one hand, which define the tree structure of the drawings, and the lexical and global constraints on the other hand, which influence the possible total orders for each drawing. If the constraints are globally consistent, they do not 'break' the tree structure; in this case, the LCG grammar is semilinear. Note that grammars of the most general LCG theory $(\mathcal{D}_*, \mathcal{L}_\emptyset)$ contain no constraints at all, neither lexical nor model class constraints, thus they are semilinear.

6.3 Mildly context-sensitive dependency languages and grammars

In this last section, we combine the results from above and from the previous chapters to characterise a class of mildly context-sensitive dependency languages, and give the corresponding conditions for the design of LCG grammars. Afterwards, it can be easily shown that VLG grammars, the valency linearisation grammars from the last chapter, are mildly context-sensitive.

[3] More precisely, it contains the pre-order projections of the trees.

6.3.1 Dependency languages

The characterisation of mildly context-sensitive dependency languages, i.e. sets of dependency structures whose projections form a mildly context-sensitive language, consists of three conditions, which imply the respective original conditions.

1. **The membership problem is polynomial**
   By deciding in polynomial time whether a drawing is in the dependency language, we also decide in polynomial time the membership of the projection of the drawing in the string language.

2. **The underlying tree language is regular**
   In this case, the structures of the dependency language can be seen as trees of a regular tree language with an added global order. As proven in the previous section, the dependency language is then semilinear and thereby has the constant growth property.

3. **The gap degree is restricted**
   If the gap degree is restricted, the number of discontinuities within connected dependents, i.e. within a substructure, is limited, and so is the number of cross-serial dependencies in the projections.

Note that this characterisation is stricter than the original one, and probably excludes languages that actually should be considered mildly context-sensitive. On the other hand, these conditions establish a more formal basis for the notion of mild context-sensitivity, at least in the field of dependency languages.

6.3.2 LCG grammars

The conditions above can be transferred in a straight-forward manner to characterise a class of LCG grammars that describe mildly context-sensitive dependency languages.

Let T be an LCG theory, and G be an LCG grammar of type T.

- The membership problem is polynomial if the fixed recognition problem is polynomial. Chapter 4 gave restrictions on T to achieve polynomiality of the fixed recognition problem.

- The gap degree can be directly restricted in the theory by choosing an appropriate model class.

- If all constraints in the theory are globally consistent, then, as shown above, the tree structures of the drawings in the dependency languages form a regular tree language.

6.3.3 Valency linearisation grammars are mildly context-sensitive

It is easy to show that all grammars of the VLG theories presented in the last chapter fulfil the characterisation of mildly context-sensitive grammars above.

We have seen that VLG grammars are recognisable in $O(n^{(k+1)(g+1)})$, i.e. the fixed recognition problem is polynomial. The gap degree is restricted by definition – each VLG theory is based on a gap-restricted class of drawings. Finally, it is possible to translate each VLG grammar into an equivalent grammar with extended edge labels containing gap information; constraints in this grammar are globally consistent.

VLG and Linear Context-Free Rewriting Systems The linearisations described by constraints in a VLG grammar are actually a special case of the rules of a Linear Context-Free Rewriting System (LCFRS). We can construct an LCFRS from a VLG grammar by creating one rule per linearisation and entry: The left hand side contains the incoming edge label, while the right hand side contains the linearisation. All other information in the lexical entry, namely the word and the outgoing valency set, can be retrieved from the linearisation. From this perspective, valency part linearisation grammars can be seen as a declarative description language of gap-restricted Linear Context-Free Rewriting Systems.

It can be shown that a gap-restricted LCFRS generates a mildly context-sensitive language [19]. This provides an alternative proof that VLG grammars are mildly context-sensitive.

Chapter 7

Summary

We have formally introduced Lexicalised Configuration Grammars (LCG), a framework for declarative specifications of dependency grammars. While LCG is powerful enough to encode a variety of existing formalisms, it contains several design decisions and deliberate restrictions that simplify the analysis of computational aspects.

As a first step to make complexity tractable, we analysed the complexity classes of various LCG fragments and found large computational differences between them. For example, the fixed recognition problem for the Linear Specification Language (LSL) was shown to be NP-complete, while the same problem for the same formalism without lexical constraints is polynomial.

By applying chart parsing techniques from the well-explored field of generative grammars, we were able to isolate the computational factors that contribute to parsing complexity. Some of these factors can be easily restricted for specific theories, yielding a notable improvement of efficiency. An important result is that an LCG theory over drawings with a restricted gap degree g, with constraints that are decidable by considering the yields of the local subdrawings only, is recognisable in $O(n^{2(g+1)(|G|+1)+1})$, which is polynomial in the length of the input sentence.

To remove the factor $|G|$ from the exponent, we examined a modified parser that performs saturations in several steps. We found that a theory over gap-restricted, well-nested drawings, in which the constraints are decidable on the level of saturation steps, is recognisable in $O(n^{4(g+1)})$. In general, however, the constraints have been proven to be quite difficult to restrict, as they are subtly interwoven and influence each other. A main observation is that improving efficiency of our parser is generally focussed around the question to what extent the constraints can be incorporated into the parsing algorithm. The more parse results are produced that automatically satisfy the constraints, the less there is the need for time-consuming generate-and-test mechanisms using external constraint checking algorithms.

This implies that efficiency highly depends on the types of constraints used in the formalism. We therefore left the abstract perspective and presented a concrete theory, VLG, with precedence and gap constraints. We have seen that these constraints are descriptions of local valency part linearisations, which can be built efficiently into the parser. Moreover, the theory is expressive enough to allow for an encoding of TAG and gap-restricted LSL grammars.

VLG has a strong connection to the class of mildly context-senstive languages, and can be seen as a special case of linear context-free rewriting systems. Using the results in this

thesis, we have given a formal characterisation of mildly context-sensitive dependency languages: a set of gap-restricted dependency structures with a regular underlying tree language and a polynomial membership problem is mildly context-sensitive.

Future work

Due to the explorative nature of this thesis, there are various open topics that are worth a more detailed examination.

An interesting task could be to encode other grammar formalisms into LCG, such as Multi-Component TAG (MCTAG) and Combinatory Categorial Grammar (CCG) in order to gain new insights into their complexity properties.

LCG misses several common features of other dependency frameworks which would be convenient for linguistic applications. Examples are optional and 'star' valencies, which allow a variable number of edges marked with a specific label. Another question is then how these features can be integrated elegantly into LCG, and how they affect the parsing complexity.

Concerning the computational factors, one could try to find more formal characterisations for theories with a polynomial general membership problem. This probably implies a change of the parsing schema and the factors; maybe even an extended framework is needed.

In the field of mildly context-sensitive languages, we have subsumed constant growth by semilinearity, which is in turn subsumed by regular tree grammars. The resulting characterisation may be too strong a restriction; there are probably find weaker conditions for mildly context-sensitive dependency languages.

Another interesting idea is to reformulate VLG as some sort of underspecified, gap-restricted Linear Context-Free Rewriting System. Moreover, one could extend VLG to a formalism that is equivalent to Simple Literal Movement Grammars (SLMG) [10], which is known to generate exactly the set of polynomial-time recognisable languages.

Last but not least, one could try to make use of the complexity results in a practical parsing environment, such as the XDG framework [5], and measure the actual impact of efficient representations and built-in constraints on the time complexity.

Bibliography

[1] Yehoshua Bar-Hillel. A quasi-arithmetical notation for syntactic description. *Language*, 29(1):47–58, 1953.

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

[3] Manuel Bodirsky, Marco Kuhlmann, and Mathias Möhl. 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.

[4] Mike Daniels and W. Detmar Meurers. Improving the efficiency of parsing with discontinuous constituents. In Shuly Wintner, editor, *7th International Workshop on Natural Language Understanding and Logic Programming*, Copenhagen, 2002.

[5] Ralph Debusmann, Denys Duchier, and Geert-Jan M. Kruijff. Extensible dependency grammar: A new methodology. In *Proceedings of the COLING 2004 Workshop on Recent Advances in Dependency Grammar*, Geneva/SUI, 2004.

[6] Jay C. Earley. An efficient context-free parsing algorithm. *Commun. ACM*, 13(2):94–102, 1970.

[7] Haim Gaifman. Dependency systems and phrase-structure systems. *Information and Control*, 8(3):304–337, 1965.

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

[9] Robert Grabowski, Marco Kuhlmann, and Mathias Möhl. Lexicalised configuration grammars. In *2nd International Workshop on Constraint Solving and Processing*, Sitges, Spain, 2005.

[10] Annius V. Groenink. An elegant grammatical formalism for the class of polynomial-time recognisable languages. In *4th Meeting on Mathematics of Language*, Philadelphia, PA, USA, 1995.

[11] Aravind K. Joshi. How much context-sensitivity is necessary for characterizing structural descriptions: Tree adjoining grammars. In D. Dowty, L. Karttunen, and A. Zwicky, editors, *Natural Language Processing: Theoretical, Computational, and Psychological Perspectives*. Cambridge University Press, New York, 1985.

[12] Aravind K. Joshi and Yves Schabes. Tree-adjoining grammars. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 3, pages 96–124. Springer, Berlin, Heidelberg, New York, 1997.

[13] Marcus Kracht. *The Mathematics of Language*, volume 63 of *Studies in Generative Grammars*. Mouton de Gruyter, Berlin, 2003.

[14] Mathias Möhl. Drawings as models of syntactic structure: Theory and algorithms. Diploma thesis, Saarland University, 2006.

[15] Stuart M. Shieber. Evidence against the context-freeness of natural language. *Linguistics and Philosophy*, 8:333–343, 1985.

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

[17] Klaas Sikkel. *Parsing Schemata: A Framework for Specification and Analysis of Parsing Algorithms*. Springer, Berlin, Heidelberg, New York, 1997.

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

[19] K. Vijay-Shanker, David J. Weir, and Aravind K. Joshi. Characterizing structural descriptions produced by various grammar formalisms. In *25th annual meeting on Association for Computational Linguistics*, Stanford, 1987.