


FO-Query Enumeration over SLP-Compressed Structures of Bounded Degree

Markus Lohrey 

University of Siegen, Germany

Sebastian Maneth 

University of Bremen, Germany

Markus L. Schmid 

Humboldt-Universität zu Berlin, Germany

Abstract

Enumerating the result set of a first-order query over a relational structure of bounded degree can be done with linear preprocessing and constant delay. In this work, we extend this result towards the compressed perspective where the structure is given in a potentially highly compressed form by a straight-line program (SLP). Our main result is an algorithm that enumerates the result set of a first-order query over a structure of bounded degree that is represented by an SLP satisfying the so-called apex condition. For a fixed formula, the enumeration algorithm has constant delay and needs a preprocessing time that is linear in the size of the SLP.

2012 ACM Subject Classification Theory of computation → Database query processing and optimization (theory); Theory of computation → Logic and databases

Keywords and phrases Enumeration algorithms, FO-logic, query evaluation over compressed data

Digital Object Identifier 10.4230/LIPIcs.MFCS.2025.69

Related Version *Full Version:* <https://arxiv.org/abs/2506.19421> [43]

Funding *Markus L. Schmid:* Supported by the German Research Foundation (Deutsche Forschungsgemeinschaft, DFG) – project number 522576760 (gefördert durch die Deutsche Forschungsgemeinschaft (DFG) – Projektnummer 522576760).

1 Introduction

First order model checking (i.e., deciding whether an FO-sentence ϕ holds in a relational structure \mathcal{U} , $\mathcal{U} \models \phi$ for short) is a classical problem in computer science and its complexity has been thoroughly investigated; see, e.g., [21, 32, 37]. In database theory, it is of importance due to its practical relevance for evaluating SQL-like query languages in relational databases. FO model checking is PSPACE-complete when ϕ and \mathcal{U} are both part of the input, but it becomes fixed-parameter tractable (even *linear* fixed-parameter tractable) with respect to the parameter $|\phi|$ when \mathcal{U} is restricted to a suitable class of relational structures (see the paragraph on related work below for details), while for the class of all structures it is not fixed-parameter tractable modulo certain complexity assumptions. This is relevant, since in practical scenarios queries are often small, especially in comparison to the data (represented by the relational structure) that is often huge.

FO model checking (i.e., checking a Boolean query that returns either *true* or *false*) reduces to practical query evaluation tasks and is therefore suitable to transfer lower bounds. However, from a practical point of view, *FO-query enumeration* is more relevant. It is the problem of enumerating without repetitions for an FO-formula $\phi(x_1, \dots, x_k)$ with free variables x_1, \dots, x_k the *result set* $\phi(\mathcal{U})$ of all tuples $(a_1, \dots, a_k) \in \mathcal{U}^k$ such that $\mathcal{U} \models \phi(a_1, \dots, a_k)$. Since $\phi(\mathcal{U})$ can be rather large (exponential in k in general), the total time for enumeration is not a good measure for the performance of an enumeration algorithm. More realistic



© Markus Lohrey, Sebastian Maneth, and Markus L. Schmid;
licensed under Creative Commons License CC-BY 4.0

50th International Symposium on Mathematical Foundations of Computer Science (MFCS 2025).

Editors: Paweł Gawrychowski, Filip Mazowiecki, and Michał Skrzypczak; Article No. 69; pp. 69:1–69:20

Leibniz International Proceedings in Informatics



LIPICs Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

measures are the *preprocessing time* (used for performing some preprocessing on the input) and the *delay*, which is the maximum time needed between the production of two consecutive output tuples from $\phi(\mathcal{U})$. In *data complexity* (where we consider $|\phi|$ to be constant), the best we can hope for is linear preprocessing time (i.e., $f(|\phi|) \cdot |\mathcal{U}|$ for a computable function f) and constant delay (i.e., the delay is $f(|\phi|)$ for some computable function f and therefore does not depend on $|\mathcal{U}|$). Over the last two decades, many of the linear time (with respect to data complexity) FO model checking algorithms for various subclasses of structures have been extended to FO-query enumeration algorithms with linear (or quasi-linear) time preprocessing and constant delay (see the paragraph on related work below for the relevant literature).

In this work, we extend FO-query enumeration towards the compressed perspective, i.e., we wish to enumerate the result set $\phi(\mathcal{U})$ in the scenario where \mathcal{U} is given in a potentially highly compressed form, and we want to work directly on this compressed form without decompressing it. In this regard, we contribute to a recent research effort in database theory that is concerned with *query evaluation over compressed data* [45, 51, 59, 60]. Let us now explain this framework in more detail.

Query evaluation over compressed data. Query evaluation over compressed data combines the classical task of query evaluation with the paradigm of *algorithmics on compressed data* (ACD), i.e., solving computational tasks directly on compressed data objects without prior decompression. ACD is an established algorithmic paradigm and it works very well in the context of *grammar-based compression* with so-called *straight-line programs* (SLPs). Such SLPs use grammar-like formalisms in order to specify how a data object is constructed from small building blocks. For example, if the data object is a finite string w , then an SLP is just a context-free grammar for the language $\{w\}$. For instance, the SLP $S \rightarrow AA$, $A \rightarrow BBC$, $B \rightarrow ba$, $C \rightarrow cb$ (where S, A, B, C are nonterminals and a, b, c are terminals) produces the string *babacbbabacb*. While SLPs achieve exponential compression in the best case, there are also fast heuristic compressors that yield decent compression in practical scenarios. Moreover, SLPs are very well suited for ACD; see, e.g., [38].

An important point is that the ACD perspective can lead to dramatic running time improvements: if the same problem can be solved in linear time both in the uncompressed and in the compressed setting (i.e., linear in the compressed size), then in the case that the input can be compressed from size n to size $\mathcal{O}(\log n)$ (which is possible with SLPs in the best case), the algorithm for the compressed data has a running time of $\mathcal{O}(\log n)$ (compared to $\mathcal{O}(n)$ for the algorithm working on uncompressed data). An important problem that shows this behavior is for instance pattern matching in compressed texts [22].

SLPs are most famous for strings (see [5, 10, 22, 23] for some recent publications and [38] for a survey). What makes them particularly appealing for query evaluation is that their general approach of compressing data objects by grammars extends from strings to more complex structures like trees [24, 40, 42, 44] and hypergraphs (i.e., general relational structures) [36, 39, 46, 47], while, at the same time, their good ACD-properties are maintained to some extent. This is due to the fact that context-free string grammars extend to context-free tree grammars [58] (see also [25]) and to hyperedge replacement grammars [6, 27] (see also [16]).

In this work, we are concerned with FO-query enumeration for relational structures that are compressed by SLPs based on hyperedge replacement grammars (also known as hierarchical graph definitions or SL HR grammars; see the paragraph on related work for references). An example of such an SLP is shown in Figure 1. It consists of productions (shown in Figure 1 on the left) that replace nonterminals (S , A , and B in Figure 1) by their unique right-hand sides. Each right-hand side is a relational structure (a directed

graph in Figure 1) together with occurrences of earlier defined nonterminals and certain distinguished contact nodes (labelled by 1 and 2 in Figure 1). In this way, every nonterminal $X \in \{S, A, B\}$ produces a relational structure $\text{val}(X)$ (the value of X) with distinguished contact nodes. These structures are shown in Figure 1 on the right. When replacing for instance the occurrence of B in the right hand side of S by $\text{val}(B)$, one identifies for every $i \in \{1, 2\}$ the i -labelled contact node in $\text{val}(B)$ with the node that is connected by the i -labelled dotted edge with the B -occurrence in the right-hand side of S (these are the nodes labelled with u and v in Figure 1).

Main result. It is known that FO-query enumeration for degree-bounded structures can be done with linear preprocessing and constant delay [13, 28]. Moreover, FO model checking for *SLP-compressed* degree-bounded structures can be done efficiently [39]. We combine these two results and therefore extend FO-query enumeration for bounded-degree structures towards the SLP-compressed setting, or, in other words, we extend FO model checking of SLP-compressed structures to the query-enumeration perspective. A preliminary version of our main result is stated below. It restricts to so-called apex SLPs. Roughly speaking, the apex property demands that each graph replacing a nonterminal must not contain other nonterminals at the “contact nodes” (the nodes the nonterminal was incident with). The apex property is well known from graph language theory [16, 17, 18] and has been used for SLPs in [39, 49].

► **Theorem 1.** *Let $d \in \mathbb{N}$ be a constant. For an FO-formula $\phi(x_1, \dots, x_k)$ and a relational structure \mathcal{U} , whose Gaifman graph has degree at most d , and that is given in compressed form by an apex SLP D , one can enumerate the result set $\phi(\mathcal{U})$ after preprocessing time $f(d, |\phi|) \cdot |D|$ and delay $f(d, |\phi|)$ for some computable function f .*

Note that the preprocessing is linear in the compressed size $|D|$ instead of the data size $|\mathcal{U}|$.

We prove this result by extending the FO-query enumeration algorithm for uncompressed structures from [28] to the SLP-compressed setting. For this we have to overcome considerable technical barriers. The algorithm of [28] exploits the Gaifman locality of FO-queries. In the preprocessing phase the algorithm computes for each element $a \in \mathcal{U}$ the r -sphere around a for a radius r that only depends on the formula ϕ . This leads to a preprocessing time of $|\mathcal{U}| \cdot f(d, \phi)$. For an SLP-compressed structure we cannot afford to iterate over all elements of the structure. Inspired by a technique from [39], we will expand every nonterminal of the SLP D locally up to a size that depends only on ϕ and d . This leads to at most $|D|$ substructures of size $f(d, |\phi|)$. Our enumeration algorithm then enumerates certain paths in the derivation tree defined by D and for each such path ending in a nonterminal A it searches in the precomputed local expansion of A for nodes with a certain sphere type.

Related work. In the uncompressed setting, there are several classes of relational structures for which FO-query enumeration can be solved with linear (or quasi-linear) preprocessing and constant delay, e.g., relational structures with bounded degree [8, 13, 28], low degree [14], (locally) bounded expansion [29, 64], and structures that are tree-like [3, 30] or nowhere dense [61]; see [7, 63] for surveys. Moreover, for conjunctive queries with certain acyclicity conditions, linear preprocessing and constant delay is also possible for the class of all relational structures [4, 7]. The algorithm from [28] is the most relevant one for our work.

Concerning other work on query enumeration on SLP-compressed data, we mention [51, 59, 60], which deals with constant delay enumeration for (a fragment of) MSO-queries on SLP-compressed strings, and [45], which presents a linear preprocessing and constant delay algorithm for MSO-queries on SLP-compressed unranked forests.

SLPs for (hyper)graphs were introduced as hierarchical graph descriptions by Lengauer and Wanke [36] and have been further studied, e.g., in [9, 19, 20, 34, 35, 49, 48, 50]. Model checking problems for SLP-compressed graphs have been studied in [39] for FO and MSO, [26] for fixpoint logics, and [1, 2] for the temporal logics LTL and CTL in the context of hierarchical state machines (which are a particular type of graph SLPs). Particularly relevant for this paper is a result from [39] stating that for every level Σ_i^P of the polynomial time hierarchy there is a fixed FO-formula for which the model checking problem for SLP-compressed input graphs is Σ_i^P -complete. In contrast, for apex SLPs the model checking problem for every fixed FO-formula belongs to NL (nondeterministic logspace) [39]. This (and the fact that FO-query enumeration reduces to FO model checking) partly explains the restriction to apex SLPs in Theorem 1.

Compression of graphs via graph SLPs has been considered in [57] following a “Sequitur scheme” [52] and in [47] following a “RePair scheme” [33] (see also [41]); note that both compressors produce graph SLPs that may *not* be apex.

Another recent concept in database theory that is concerned with compressed representations of relational data and query evaluation are *factorized databases* (see [31, 53, 54, 55, 56]). Intuitively speaking, in a factorized representation of a relational structure each relation R is represented as an expression over the relational operators union and product that evaluates to R . However, SLPs for relational structures and factorized representations cover completely different aspects of redundancy: A factorized representation is always at least as large as its active domain (i.e., all elements that occur in some tuple), while an SLP for a relational structure can be of logarithmic size in the size of the universe of the structure. On the other hand, small factorized representations do not seem to necessarily translate into small SLPs.

2 General Notations

Let $\mathbb{N} = \{0, 1, 2, \dots\}$. For every $k \in \mathbb{N}$, we set $[k] = \{1, 2, \dots, k\}$. For a finite alphabet A , we denote by A^* the set of all finite strings over A including the empty string ε . For a partial $f : A \rightarrow B$ let $\text{dom}(f) = \{a \in A : f(a) \neq \perp\} \subseteq A$ (where $\perp \notin B$ stands for undefined) and $\text{ran}(f) = \{f(a) : a \in \text{dom}(f)\} \subseteq B$. For functions $f : A \rightarrow B$ and $g : B \rightarrow C$ we define the composition $g \circ f : A \rightarrow C$ by $(g \circ f)(a) = g(f(a))$ for all $a \in A$.

A *partial k -tuple* over a set A is a partial function $t : [k] \rightarrow A$. If $\text{dom}(t) = [k]$, then we also say that t is a *complete k -tuple* or just a *k -tuple*; in this case we also write t in the conventional form $(t(1), t(2), \dots, t(k))$. Two partial k -tuples t_1 and t_2 are *disjoint* if $\text{dom}(t_1) \cap \text{dom}(t_2) = \emptyset$. In this case, their union $t_1 \sqcup t_2$ is the partial k -tuple defined by $(t_1 \sqcup t_2)(j) = t_i(j)$ if $j \in \text{dom}(t_i)$ for $i \in \{1, 2\}$ and $(t_1 \sqcup t_2)(j) = \perp$ if $j \notin \text{dom}(t_1) \cup \text{dom}(t_2)$.

2.1 Directed acyclic graphs

An *ordered dag* (directed acyclic graph) is a triple $G = (V, \gamma, \iota)$, where V is a finite set of nodes, $\gamma : V \rightarrow V^*$ is the child-function, the relation $E := \{(u, v) : u, v \in V, v \text{ occurs in } \gamma(u)\}$ is acyclic, and $\iota \in V$ is the *initial node*. The size of G is $|G| = \sum_{v \in V} (1 + |\gamma(v)|)$. A node $v \in V$ with $|\gamma(v)| = 0$ is called a *leaf*.

A path in G (from v_0 to v_n) is a sequence $p = v_0 i_1 v_1 i_2 \dots v_{n-1} i_n v_n \in V(NV)^*$ such that $1 \leq i_k \leq |\gamma(v_{k-1})|$ for all $k \in [n]$. The length of this path p is n (we may have $n = 0$ in which case $p = v_0$) and we also call p a *v_0 -to- v_n path* or *v_0 -path* if the end point v_n is not important. An ι -path is also called an *initial path*. We extend this notation to subsets of V in the obvious way, e.g., for $A, B \subseteq V$ and $v \in V$ we talk about *A -to- v paths*, *A -to- B paths*, *A -to-leaf paths* (where “leaf” refers to the set of all leaves of the dag), *initial-to-leaf paths*,

etc. For a v_0 -to- v_1 path $p = p'v_1$ and a v_1 -to- v_2 path $q = v_1q'$ we define the v_0 -to- v_2 path $pq = p'v_1q'$ (note that if we just concatenate p and q as words, then we have to replace the double occurrence v_1v_1 by v_1 to obtain pq). We say that the path p is a *prefix* of the path q if there is a path r such that $q = qr$.

Since we consider ordered dags, there is a natural lexicographical ordering on all v -paths (i.e., all paths that start in the same node v). More precisely, for two different v -paths p and q we write $p < q$ if either p is a proper prefix of q or we can write p and q as $p = rip'$, $q = rjq'$ for paths r, p', q' and $i, j \in \mathbb{N}$ with $i < j$.

2.2 Relational structures and first order logic

A *signature* \mathcal{R} is a finite set consisting of relation symbols r_i ($i \in I$) and constant symbols c_j ($j \in J$). Each relation symbol r_i has an associated arity α_i . A *structure over the signature* \mathcal{R} is a tuple $\mathcal{U} = (U, (R_i)_{i \in I}, (u_j)_{j \in J})$, where U is a finite non-empty set (the universe of \mathcal{U}), $R_i \subseteq U^{\alpha_i}$ is the relation associated with the relation symbol r_i , and $u_j \in U$ is the constant associated with the constant symbol c_j . Note that we restrict to finite structures. If the structure \mathcal{U} is clear from the context, we will identify R_i (u_j , respectively) with the relation symbol r_i (the constant symbol c_j , respectively). Sometimes, when we want to refer to the universe U , we will refer to \mathcal{U} itself. For instance, we write $a \in \mathcal{U}$ for $ua \in U$, or $f : [n] \rightarrow \mathcal{U}$ for a function $f : [n] \rightarrow U$. The size $|\mathcal{U}|$ of \mathcal{U} is $|U| + \sum_{i \in I} \alpha_i \cdot |R_i|$. As usual, a constant $a \in \mathcal{U}$ may be replaced by the unary relation $\{a\}$. Thus, in the following, we will only consider signatures without constant symbols, except when we explicitly introduce constants. Let $\mathcal{R} = \{r_i : i \in I\}$ be such a signature (we call it a *relational signature*) and let $\mathcal{U} = (U, (R_i)_{i \in I})$ be a structure over \mathcal{R} (we call it a *relational structure*). For relational structures \mathcal{U}_1 and \mathcal{U}_2 over the signature \mathcal{R} , we write $\mathcal{U}_1 \simeq \mathcal{U}_2$ to denote that \mathcal{U}_1 and \mathcal{U}_2 are isomorphic. A substructure of $\mathcal{U} = (U, (R_i)_{i \in I})$ is a structure $(V, (S_i)_{i \in I})$ such that $V \subseteq U$ and $S_i \subseteq R_i$ for all $i \in I$. The substructure of \mathcal{U} induced by $V \subseteq U$ is $(V, (R_i \cap V^{\alpha_i})_{i \in I})$. We define the undirected graph $\mathcal{G}(\mathcal{U}) = (U, E)$ (the so-called Gaifman graph of \mathcal{U}), where E contains an edge (a, b) if and only if there is a binary relation R_i ($i \in I$) and a tuple $(a_1, \dots, a_{\alpha_i}) \in R_i$ with $\{a, b\} \subseteq \{a_1, \dots, a_{\alpha_i}\}$. The degree of \mathcal{U} is the maximal degree of a node in $\mathcal{G}(\mathcal{U})$. If \mathcal{U} has degree at most d , we also say that \mathcal{U} is a *degree- d bounded structures*.

We use *first-order logic* (FO) over finite relational structures; see [15] for a detailed introduction and the full version [43] for some standard notations. For an FO-formula $\psi(x_1, \dots, x_k)$ over the signature \mathcal{R} with free variables x_1, \dots, x_k and a relational structure $\mathcal{U} = (U, (R_i)_{i \in I})$ over \mathcal{R} and $a_1, \dots, a_k \in U$, we write $\mathcal{U} \models \psi(a_1, \dots, a_k)$ if ψ is true in \mathcal{U} when the variable x_i is set to a_i for all $i \in [k]$. Hence, an FO-formula $\psi(x_1, \dots, x_k)$ can be interpreted as an *FO-query* that, for a given structure \mathcal{U} , defines a *result set*

$$\psi(\mathcal{U}) = \{(a_1, \dots, a_k) \in \mathcal{U}^k : \mathcal{U} \models \psi(a_1, \dots, a_k)\}.$$

The *quantifier rank* $\text{qr}(\psi)$ of an FO-formula ψ is inductively defined as follows: $\text{qr}(\psi) = 0$ if ψ contains no quantifiers, $\text{qr}(\neg\psi) = \text{qr}(\psi)$, $\text{qr}(\psi_1 \wedge \psi_2) = \text{qr}(\psi_1 \vee \psi_2) = \max\{\text{qr}(\psi_1), \text{qr}(\psi_2)\}$ and $\text{qr}(\forall x\psi) = \text{qr}(\exists x\psi) = 1 + \text{qr}(\psi)$.

In the rest of the paper, we assume that the signature only contains relation symbols of arity at most two. It is folklore that FO model checking and FO-query enumeration over arbitrary signatures can be reduced to this case; see the full version [43] for a possible construction. This construction can be carried out in linear time (in the size of the formula and the structure) and it increase the degree of the structure as well as the quantifier rank of the formula by at most one.

2.3 Distances, spheres and neighborhoods

Let us fix a relational signature \mathcal{R} (containing only relation symbols of arity at most two) and let $\mathcal{U} = (U, (R_i)_{i \in I})$ be a structure over this signature. We say that \mathcal{U} is *connected*, if its Gaifman graph $\mathcal{G}(\mathcal{U})$ is connected. The distance between elements $a, b \in U$ in the graph $\mathcal{G}(\mathcal{U})$ is denoted by $\text{dist}_{\mathcal{U}}(a, b)$ (it can be ∞). For subsets $A, B \subseteq U$ we define $\text{dist}_{\mathcal{U}}(A, B) = \min\{\text{dist}_{\mathcal{U}}(a, b) : a \in A, b \in B\}$. For two partial tuples (of any arity) t, t' over U let $\text{dist}_{\mathcal{U}}(t, t') = \text{dist}_{\mathcal{U}}(\text{ran}(t), \text{ran}(t'))$.

Fix a constant $d \geq 2$. We will only consider degree- d bounded structures in the following. Let us fix such a structure \mathcal{U} (over the relational signature \mathcal{R}). Take additional constant symbols c_1, c_2, \dots called *sphere center constants*. For an $r \geq 1$ and a partial k -tuple $t : [k] \rightarrow \mathcal{U}$ we define the r -sphere $\mathcal{S}_{\mathcal{U}, r}(t) = \{b \in \mathcal{U} : \text{dist}_{\mathcal{U}}(t, b) \leq r\}$. The r -neighborhood $\mathcal{N}_{\mathcal{U}, r}(t)$ of t is obtained by taking the substructure of \mathcal{U} induced by $\mathcal{S}_{\mathcal{U}, r}(t)$ and then adding every node $t(i)$ ($i \in \text{dom}(t)$) as the interpretation of the sphere center constant c_i . Hence, it is a structure over the signature $\mathcal{R} \cup \{c_i : i \in \text{dom}(t)\}$. The r -neighborhood of a k -tuple has at most $k \cdot \sum_{i=0}^r d^i \leq k \cdot d^{r+1}$ elements (here, the inequality holds since we assume $d \geq 2$).

We use the above definitions also for a single element $a \in \mathcal{U}$ in place of a tuple t ; formally a is identified with the 1-tuple t such that $t(1) = a$. We are mainly interested in r -spheres and r -neighborhoods of complete k -tuples, but the corresponding notions for partial k -tuples will be convenient later. We also drop the subscript \mathcal{U} from the above notations if this does not cause any confusion.

A (k, r) -neighborhood type is an isomorphism type for the r -neighborhood of a complete k -tuple in a degree- d bounded structure. More precisely, we can define a (k, r) -neighborhood type as a degree- d bounded structure \mathcal{B} over the signature $\mathcal{R} \cup \{c_1, \dots, c_k\}$ such that

- the universe of \mathcal{B} is of the form $[\ell]$ for some $\ell \leq k \cdot d^{r+1}$ and
- for every $j \in [\ell]$ there is $i \in [k]$ such that $\text{dist}_{\mathcal{B}}(a_i, j) \leq r$, where, for every $i \in [k]$, a_i is the interpretation of the sphere center constant c_i .

From each isomorphism class of (k, r) -neighborhood types we select a unique representative and write $\mathcal{T}_{k, r}$ for the set of all selected representatives. Then, for every k -tuple $\bar{a} \in \mathcal{U}^k$ there is a unique $\mathcal{B} \in \mathcal{T}_{k, r}$ such that $\mathcal{N}_{\mathcal{U}, r}(\bar{a}) \simeq \mathcal{B}$; we call it the (k, r) -neighborhood type of \bar{a} and say that \bar{a} is a \mathcal{B} -tuple. In case $k = 1$ we speak of \mathcal{B} -nodes instead of \mathcal{B} -tuples, write \mathcal{T}_r for $\mathcal{T}_{1, r}$ and call its elements r -neighborhood types instead of $(1, r)$ -neighborhood types.

For every (k, r) -neighborhood type $\mathcal{B} \in \mathcal{T}_{k, r}$ there is an FO-formula $\psi^{\mathcal{B}}(x_1, \dots, x_k)$ such that for every degree- d bounded structure \mathcal{U} and every k -tuple $\bar{a} \in \mathcal{U}^k$ we have $\mathcal{U} \models \psi^{\mathcal{B}}(\bar{a})$ if and only if \bar{a} is a \mathcal{B} -tuple.

2.4 Enumeration algorithms and the machine model

FO-query enumeration is the following problem: Given an FO-formula $\phi(x_1, \dots, x_k)$ over some signature \mathcal{R} and a relational structure \mathcal{U} over \mathcal{R} , we want to enumerate all tuples from $\phi(\mathcal{U})$ in some order and without repetitions, i.e., we want to produce a sequence $(t_1, \dots, t_n, t_{n+1})$ with $\{t_1, \dots, t_n\} = \phi(\mathcal{U})$, $|\phi(\mathcal{U})| = n$ and $t_{n+1} = \text{EOE}$ is the *end-of-enumeration* marker. An algorithm for FO-query enumeration starts with a *preprocessing phase* in which no output is produced, followed by an *enumeration phase*, where the elements $t_1, t_2, \dots, t_n, t_{n+1}$ are produced one after the other. The running time of the preprocessing phase is called the *preprocessing time*, and the *delay* measures the maximal time between the computation of two consecutive outputs t_i and t_{i+1} for every $i \in [n]$.

Usually, one restricts the input structure \mathcal{U} to some subclass \mathcal{C}_d of relational structures that is defined by some parameter d (in this paper, \mathcal{C}_d is the class of degree- d bounded structures). We say that an algorithm for FO-query enumeration for \mathcal{C}_d has *linear preprocessing* and

constant delay, if its preprocessing time is $\mathcal{O}(|\mathcal{U}| \cdot f(d, |\phi|))$ and its delay is $\mathcal{O}(f(d, |\phi|))$ for some computable function f . This complexity measure where the query ϕ is considered to be constant and the running time is only measured in terms of the data, i.e., the size of the relational structure, is also called *data complexity*. In data complexity, linear preprocessing and constant delay is considered to be optimal (since we assume that the relational structure has to be read at least once). As mentioned in the introduction, FO-query enumeration can be solved with linear preprocessing and constant delay for several classes \mathcal{C}_d .

For proving upper bounds in data complexity, we often have to argue that certain computational tasks can be performed in time $f(\cdot)$ (or $|\mathcal{U}| \cdot f(\cdot)$) for some function f . In these cases, without explicitly mentioning this in the remainder, f will always be a computable function (actually, f will be elementary, i.e., bounded by an exponent tower of fixed height). The arguments for f will only depend on the parameter d and the formula size $|\phi|$.

The special feature of this work is that we consider FO-query enumeration in the setting where the relational structure \mathcal{U} is not given explicitly, but in a potentially highly compressed form, and our enumeration algorithm must handle this compressed representation rather than decompressing it. Then the structure size $|\mathcal{U}|$ will be replaced by the size of the compressed representation of \mathcal{U} in all time bounds. This aspect shall be explained in detail in Section 4.

We use the standard RAM model with uniform cost measure as our model of computation. We will make some further restrictions for the register length tailored to the compressed setting in Section 4.2.

3 FO-Enumeration over Uncompressed Degree-Bounded Structures

In this section, we fix a relational signature $\mathcal{R} = \{R_i : i \in I\}$, constants $d \geq 2$ and ν , a degree- d bounded structure $\mathcal{U} = (U, (R_i)_{i \in I})$ over the signature \mathcal{R} , and an FO-formula $\phi(x_1, \dots, x_k)$ over the signature \mathcal{R} with $\text{qr}(\phi) = \nu$. Our goal is to enumerate the set $\phi(\mathcal{U})$ after a linear time preprocessing in constant delay. Before we consider the case where the structure \mathcal{U} is given in a compressed form, we will first outline the enumeration algorithm from [28] for the case where \mathcal{U} is given explicitly (with some modifications). In Section 5 we will extend this algorithm to the compressed setting.

By a standard application of the Gaifman locality of FO (see [43]), we first reduce the enumeration of $\phi(\mathcal{U})$ to the enumeration of all \mathcal{B} -tuples from \mathcal{U}^k for a fixed $\mathcal{B} \in \mathcal{T}_{k,r}$ (for some $r \leq 7^\nu$). Recall that \mathcal{B} contains at most $k \cdot d^{r+1}$ elements, and this upper bound only depends on d and the formula ϕ . To simplify notation, we assume that in \mathcal{B} the sphere center constant c_i is interpreted by $i \in [k]$. In particular, the sphere center constants are interpreted by different elements. This is not a real restriction; see [43].

In order to enumerate all \mathcal{B} -tuples, we will factorize \mathcal{B} into its connected components. In order to accomplish this, we need the following definitions. We first define the larger radius

$$\rho = 2rk - r + k - 1. \quad (1)$$

Every ρ -neighborhood of an element $a \in \mathcal{U}$ has at most $d^{\rho+1}$ elements. Recall that a ρ -neighborhood type is a degree- d bounded structure over the signature $\mathcal{R}_1 := \mathcal{R} \cup \{c_1\}$ with a universe $[\ell]$ for some $\ell \leq d^{\rho+1}$. W.l.o.g. we assume that the sphere center constant c_1 is interpreted by the element 1 in a ρ -neighborhood type. Hence, every $j \in [\ell]$ has distance at most ρ from 1. Moreover, the ρ -neighborhood types in \mathcal{T}_ρ are pairwise non-isomorphic.

Assume that our fixed (k, r) -neighborhood type \mathcal{B} splits into $m \geq 1$ connected components $\mathcal{C}_1^\mathcal{B}, \dots, \mathcal{C}_m^\mathcal{B}$. Thus, every $\mathcal{C}_i^\mathcal{B}$ is a connected induced substructure of \mathcal{B} , every node of \mathcal{B} belongs to exactly one $\mathcal{C}_i^\mathcal{B}$, and there is no edge in the undirected graph $\mathcal{G}(\mathcal{B})$ between two different

components $\mathcal{C}_i^{\mathcal{B}}$. Let $D_i = \mathcal{C}_i^{\mathcal{B}} \cap [k]$ be the set of sphere centers that belong to the connected component $\mathcal{C}_i^{\mathcal{B}}$. We must have $D_i \neq \emptyset$. Let $n_i = \min(D_i)$ (we could also fix any other element from D_i). Every node in $\mathcal{C}_i^{\mathcal{B}}$ has distance at most r from some $j \in D_i$. Since $\mathcal{C}_i^{\mathcal{B}}$ is connected, it follows that every node in $\mathcal{C}_i^{\mathcal{B}}$ has distance at most $r + (k-1)(2r+1) = 2rk - r + k - 1 = \rho$ from n_i (this is in fact true for every $j \in D_i$ instead of n_i). A *consistent factorization* of our (k, r) -neighborhood type \mathcal{B} is a tuple

$$\Lambda = (\mathcal{B}_1, \sigma_1, \mathcal{B}_2, \sigma_2, \dots, \mathcal{B}_m, \sigma_m)$$

with the following properties for all $i \in [m]$:

- $\mathcal{B}_i \in \mathcal{T}_\rho$ and $\sigma_i : [k] \rightarrow \mathcal{B}_i$ is a partial k -tuple with $\text{dom}(\sigma_i) = D_i$ and $\sigma_i(n_i) = 1$ (so, n_i is mapped by σ_i to the center of \mathcal{B}_i) and
- $\mathcal{N}_{\mathcal{B}_i, r}(\sigma_i) \simeq \mathcal{C}_i^{\mathcal{B}}$.

Clearly, the number of possible consistent factorizations of \mathcal{B} is bounded by $f(d, |\phi|)$.

For a ρ -neighborhood type \mathcal{B}' , a \mathcal{B}' -node $a \in \mathcal{U}$ and a partial k -tuple $\sigma : [k] \rightarrow \mathcal{B}'$ we moreover fix an isomorphism $\pi_a : \mathcal{B}' \rightarrow \mathcal{N}_{\mathcal{U}, \rho}(a)$ (this isomorphism is not necessarily unique) and define the partial k -tuple $t_{a, \sigma} : [k] \rightarrow \mathcal{U}$ by $t_{a, \sigma}(j) = \pi_a(\sigma(j))$ for all $j \in \text{dom}(\sigma)$. Note that, by definition, $\pi_a(1) = a$.

Take a consistent factorization $\Lambda = (\mathcal{B}_1, \sigma_1, \dots, \mathcal{B}_m, \sigma_m)$ of \mathcal{B} . We say that an m -tuple $(b_1, \dots, b_m) \in \mathcal{U}^m$ is *admissible* for Λ if the following conditions hold:

- for all $i \in [m]$, b_i is a \mathcal{B}_i -node, and
- for all $i, j \in [m]$ with $i \neq j$ we have

$$\text{dist}_{\mathcal{U}}(t_{b_i, \sigma_i}, t_{b_j, \sigma_j}) > 2r + 1. \quad (2)$$

Finally, with an m -tuple $\bar{b} = (b_1, \dots, b_m)$ we associate the k -tuple

$$\Lambda(\bar{b}) = t_{b_1, \sigma_1} \sqcup t_{b_2, \sigma_2} \sqcup \dots \sqcup t_{b_m, \sigma_m}.$$

Note that $t_{b_i, \sigma_i}(n_i) = \pi_{b_i}(\sigma_i(n_i)) = \pi_{b_i}(1) = b_i$.

We claim that in order to enumerate all \mathcal{B} -tuples $\bar{a} \in \mathcal{U}^k$, it suffices to enumerate for every consistent factorization $\Lambda = (\mathcal{B}_1, \sigma_1, \dots, \mathcal{B}_m, \sigma_m)$ of \mathcal{B} the set of all m -tuples $\bar{b} \in \mathcal{U}^m$ that are admissible for Λ . If we can do this, then we replace every output tuple $\bar{b} \in \mathcal{U}^m$ by $\Lambda(\bar{b}) \in \mathcal{U}^k$. Note that $\Lambda(\bar{b})$ can be easily computed in time $\mathcal{O}(k)$ from the tuple \bar{b} , the isomorphisms π_{b_i} , and the partial k -tuples $\sigma_i : [k] \rightarrow \mathcal{B}_i$. The correctness of this algorithm follows from the following two lemmas (with full proofs in [43]):

► **Lemma 2.** *If Λ is a consistent factorization of \mathcal{B} and $\bar{b} \in \mathcal{U}^m$ is admissible for Λ then $\Lambda(\bar{b}) \in \mathcal{U}^k$ is a \mathcal{B} -tuple.*

► **Lemma 3.** *If $\bar{a} \in \mathcal{U}^k$ is a \mathcal{B} -tuple then there are a unique consistent factorization Λ of \mathcal{B} and a unique m -tuple $\bar{b} \in \mathcal{U}^m$ that is admissible for Λ and such that $\bar{a} = \Lambda(\bar{b})$.*

3.1 Enumeration algorithm for uncompressed structures

Let us fix a (k, r) -neighborhood type \mathcal{B} and a consistent factorization $\Lambda = (\mathcal{B}_1, \sigma_1, \dots, \mathcal{B}_m, \sigma_m)$ of \mathcal{B} . By Lemmas 2 and 3, it suffices to enumerate (with linear preprocessing and constant delay) the set of all $\bar{a} \in \mathcal{U}^k$ that are admissible for Λ . In the preprocessing phase we compute

- for every $i \in [m]$ a list L_i containing all \mathcal{B}_i -nodes from \mathcal{U} and
- for every $a \in L_i$ an isomorphism $\pi_a : \mathcal{B}_i \rightarrow \mathcal{N}_\rho(a)$.

Algorithm 1 `extend(s)`.

```

1  $\ell := |s| + 1;$ 
2 for all  $a \in L_\ell$  such that  $sa$  is admissible do
3   if  $\ell = m$  then output  $sa$  else extend(sa)
4 return

```

It is straightforward to compute these data in time $|\mathcal{U}| \cdot f(d, |\phi|)$ (in Section 5, where we deal with the more general SLP-compressed case, this is more subtle). We classify each list L_i as being *short* if $|L_i| \leq k \cdot d^{2\rho+2r+2}$ and as being *long* otherwise. Without loss of generality, we assume that, for some $0 \leq q \leq m$ the lists L_1, \dots, L_q are short and the lists L_{q+1}, \dots, L_m are long (note that this includes the cases that all lists are short or all lists are long).

Our enumeration procedure maintains a stack of the form $a_1 a_2 \dots a_\ell$ with $0 \leq \ell \leq m$ and $a_i \in L_i$ for all $i \in [\ell]$. Note that if $\ell = 0$, then we have the empty stack ε . Such a stack is called *admissible* for Λ (or just *admissible*), if for all $i, i' \in [\ell]$ with $i \neq i'$ and all $j \in \text{dom}(\sigma_i)$ and $j' \in \text{dom}(\sigma_{i'})$ we have $\text{dist}_{\mathcal{U}}(\pi_{a_i}(\sigma_i(j)), \pi_{a_{i'}}(\sigma_{i'}(j'))) > 2r + 1$. Note that the empty stack as well as every stack a_1 with $a_1 \in L_1$ are admissible.

The general structure of our enumeration algorithm is a depth-first-left-to-right (DFLR) traversal over all admissible stacks s . For this, it calls the recursive procedure `extend` (shown as Algorithm 1) with the initial admissible stack $s = \varepsilon$. Note that whenever `extend(s)` is called, $|s| < m$ holds. It is clear that the call `extend(ε)` triggers the enumeration of all admissible stacks $a_1 a_2 \dots a_m$. In an implementation one would store s as a global variable.

Let us assume that we can check whether a stack s is admissible in time $f(d, |\phi|)$ (it is not hard to see that this is possible, and this aspect will anyway be discussed in detail for the compressed setting in Section 5). After the initial call `extend(ε)`, the algorithm constructs an admissible stack s with $|s| = q$ (or terminates) after time bounded in d, k, r and ρ (since the lists L_1, \dots, L_q are short). If some $a \in L_{q+1}$ is *non-admissible*, i.e., the stack sa is not admissible, then $\text{dist}_{\mathcal{U}}(t_{a_i, \sigma_i}, t_{a, \sigma_{q+1}}) \leq 2r + 1$ and therefore $\text{dist}(a_i, a) \leq 2\rho + 2r + 1$ for some $i \in [q]$. Thus, the total number of non-admissible elements from L_{q+1} can be bounded by a function of d, k, r and ρ . Consequently, since L_{q+1} is long, the algorithm necessarily finds some admissible $a \in L_{q+1}$ (or terminates) after time bounded in d, k, r and ρ . From this observation, the following lemma can be obtained with moderate effort; see [43].

► **Lemma 4.** *The delay of the above enumeration procedure is bounded by $f(d, |\phi|)$.*

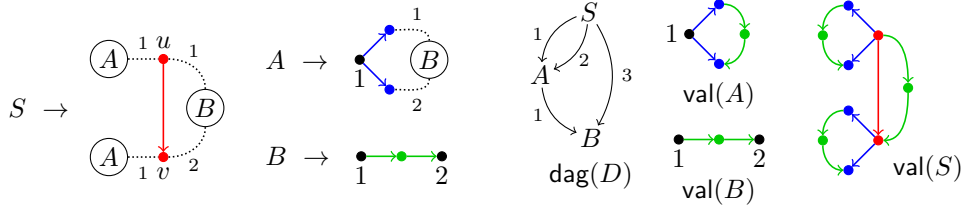
4 Straight-Line Programs for Relational Structures

In this section, we introduce the compression scheme that shall be used to compress relational structures. We first need the definition of pointed structures.

For $n \geq 0$, an n -*pointed structure* is a pair (\mathcal{U}, τ) , where \mathcal{U} is a structure and $\tau : [n] \rightarrow \mathcal{U}$ is injective. The elements in $\text{ran}(\tau)$ ($\mathcal{U} \setminus \text{ran}(\tau)$, respectively) are called *contact nodes* (*internal nodes*, respectively). The node $\tau(i)$ is called the i -*th contact node*.

A *relational straight-line program* (r -SLP or just SLP) is a tuple $D = (\mathcal{R}, N, S, P)$, where

- \mathcal{R} is a relational signature,
- N is a finite set of *nonterminals*, where every $A \in N$ has a *rank* $\text{rank}(A) \in \mathbb{N}$,
- $S \in N$ is the *initial nonterminal*, where $\text{rank}(S) = 0$, and
- P is a set of *productions* that contains for every $A \in N$ a unique production $A \rightarrow (\mathcal{U}_A, \tau_A, E_A)$ with (\mathcal{U}_A, τ_A) a $\text{rank}(A)$ -pointed structure over the signature \mathcal{R} and E_A a multiset of *references* of the form (B, σ) , where $B \in N$ and $\sigma : [\text{rank}(B)] \rightarrow \mathcal{U}_A$ is injective.



■ **Figure 1** The SLP D of Example 5 together with $\text{dag}(D)$ and $\text{val}(X)$ for $X \in \{S, A, B\}$.

- Define the binary relation \rightarrow_D on N as follows: $A \rightarrow_D B$ if and only if E_A contains a reference of the form (B, σ) . Then we require that \rightarrow_D is acyclic. Its transitive closure \succ_D is a partial order that we call the *hierarchical order* of D .

Let $|D| = \sum_{A \in N} (|\mathcal{U}_A| + \sum_{(B, \sigma) \in E_A} (1 + \text{rank}(B)))$ be the size of D . We define the ordered dag $\text{dag}(D) = (N, \gamma, S)$, where the child-function γ is defined as follows: Let $B \in N$ and let $(B_1, \sigma_1), \dots, (B_m, \sigma_m)$ be an enumeration of the references in E_B , where every reference appears in the enumeration as many times as in the multiset E_B . The specific order of the references is not important and assumed to be somehow given by the input encoding of D . We then define $\gamma(B) = B_1 \cdots B_m$.

We now define for every nonterminal $A \in N$ a $\text{rank}(A)$ -pointed relational structure $\text{val}(A)$ (the value of A). We do this on an intuitive level, a formal definition can be found in the full version [43]. If $E_A = \emptyset$, then we define $\text{val}(A) = (\mathcal{U}_A, \tau_A)$. If, on the other hand, $E_A \neq \emptyset$, then $\text{val}(A)$ is obtained from (\mathcal{U}_A, τ_A) by expanding all references in E_A . A reference $(B, \sigma) \in E_A$ is expanded by the following steps: (i) create the disjoint union of \mathcal{U}_A and \mathcal{U}_B , (ii) merge node $\tau_B(i) \in \mathcal{U}_B$ with node $\sigma(i) \in \mathcal{U}_A$ for every $i \in [\text{rank}(B)]$, (iii) remove (B, σ) from E_A , and (iv) add all references from E_B to E_A . Due to the fact that \rightarrow_D is acyclic, we can keep on expanding references (the original ones from E_A and the new ones added by the expansion operation) in any order until there are no references left. The resulting relational structure is $\text{val}(A)$; see Example 5 and Figure 1 for an illustration.

We define $\text{val}(D) = \text{val}(S)$. Since $\text{rank}(S) = 0$ it can be viewed as an ordinary (0-pointed) structure. It is not hard to see that $|\text{val}(D)| \leq 2^{\mathcal{O}(|D|)}$ and that this upper bound can be also reached. Thus, D can be seen as a compressed representation of the structure $\text{val}(D)$.

In Section 2.2 we claimed that FO-query enumeration can be reduced to the case where \mathcal{R} only contains relation symbols of arity at most two (with the details given in the full version [43]). It is easy to see that this reduction can be also done in the SLP-compressed setting simply by applying the reduction to all structures \mathcal{U}_A ; details can be again found in [43].

We say that the SLP $D = (\mathcal{R}, N, S, P)$ is *apex*, if for every $A \in N$ and every reference $(B, \sigma) \in E_A$ we have $\text{ran}(\sigma) \cap \text{ran}(\tau_A) = \emptyset$. Thus, contact nodes of a right-hand side cannot be accessed by references. Apex SLPs are called 1-level restricted in [49]. It is easy to compute the maximal degree of nodes in $\mathcal{G}(\text{val}(D))$ for an apex SLP D : for every node v in a structure \mathcal{U}_A compute d_v as the sum of (i) the degree of v in $\mathcal{G}(\mathcal{U}_A)$ and (ii) for every reference $(B, \sigma) \in E_A$ and every $i \in [\text{rank}(B)]$ with $v = \sigma(i)$, the degree of $\tau_B(i)$ in $\mathcal{G}(\mathcal{U}_B)$. Then the maximum of all these numbers d_v is the maximal degree of nodes in $\mathcal{G}(\text{val}(D))$. The apex property implies a certain locality property for $\text{val}(D)$ that will be explained in Section 4.1. In the rest of the paper we will mainly consider apex SLPs.

A simple example of a class of graphs that are exponentially compressible with apex SLPs is the class of perfect binary trees. The perfect binary tree of height n (with 2^n leaves) can be produced by an apex SLP of size $\mathcal{O}(n)$. Here is an explicit example for an apex SLP:

► **Example 5.** Consider the SLP $D = (\mathcal{R}, N, S, P)$ where \mathcal{R} only contains a binary relation symbol r_1 and $N = \{S, A, B\}$ with $\text{rank}(S) = 0$, $\text{rank}(A) = 1$ and $\text{rank}(B) = 2$. The productions of these nonterminals are depicted on the left of Figure 1. For instance, the production $S \rightarrow (\mathcal{U}_S, \tau_S, E_S)$ consists of the 0-pointed structure (\mathcal{U}_S, τ_S) , where the universe of \mathcal{U}_S consists of the two red nodes u and v , and the reference set $E_S = \{(A, \sigma_1), (A, \sigma_2), (B, \sigma_3)\}$ with $\sigma_1(1) = u$, $\sigma_2(1) = v$, $\sigma_3(1) = u$ and $\sigma_3(2) = v$ (in Figure 1 each $\sigma_i(j)$ is connected by a j -labeled dotted line with the nonterminal). The production for nonterminal B consists of a 2-pointed structure (and no references), the contact nodes of which are labeled by 1 and 2. The structure $\text{val}(D) = \text{val}(S)$ is shown on the right of Figure 1. It can be obtained by first constructing $\text{val}(A)$ by replacing the single B -reference in \mathcal{U}_A by $\mathcal{U}_B = \text{val}(B)$. Note that 1- and 2-labeled dotted lines identify the two nodes to be merged with the two contact nodes of \mathcal{U}_B , and that $\text{val}(A)$ has exactly one contact node. Then we replace the B -reference in \mathcal{U}_S by $\text{val}(B)$ and both A -references in \mathcal{U}_S by $\text{val}(A)$. This merges u (and v) with the contact node of the first (and the second) occurrence of $\text{val}(A)$. Red (resp., blue, green) edges and nodes are produced from S (resp., A , B).

Since no contact node is adjacent to any reference, this SLP is apex. The size of $\text{val}(D)$ is 31. The size of D is 26: 9 (for the S -production) + 10 (for the A -production) + 7 (for the B -production).

4.1 Representation of nodes of an SLP-compressed structure

Let $A \in N$. A node $a \in \text{val}(A)$ can be uniquely represented by a pair (p, v) such that p is an A -path in $\text{dag}(D)$ and one of the following two cases holds:

- p ends in $B \in N \setminus \{A\}$ and $v \in \mathcal{U}_B \setminus \text{ran}(\tau_B)$ is an internal node.¹
- $p = A$ and $v \in \mathcal{U}_A$.

We call this the *A-representation of a* . The S -representations of the nodes of $\text{val}(S) = \text{val}(D)$ are also called *D-representations*. Note that if (p, v) is the D -representation of a node then $v \in \mathcal{U}_A \setminus \text{ran}(\tau_A)$ for some $A \in N$ (since $\text{rank}(S) = 0$). We will often identify a node of $\text{val}(A)$ with its A -representation; in particular when $A = S$. One may view a D -representation (p, v) as a stack pv . In order to construct outgoing (or incoming) edges of (p, v) in the structure $\text{val}(D)$, one only has to modify this stack at its end; see [43] for more details.

The apex condition implies a kind of locality in $\text{val}(D)$ that can be nicely formulated in terms of D -representations: If two nodes $a = (p, u)$ and $b = (q, v)$ have distance ζ in the graph $\mathcal{G}(\text{val}(D))$ then the prefix distance between p and q (which is the number of edges in p and q that do not belong to the longest common prefix of p and q) is also at most ζ . This property is exploited several times in the paper.

Based on A -representations, we can define a natural embedding of $\text{val}(B)$ into $\text{val}(A)$ in case $A \succ_D B$. Assume that p is a non-empty A -to- B path in $\text{dag}(D)$ with $A \neq B$. Let us write $p = p'CiB$ for some nonterminal C (we may have $C = A$). Let $(B, \sigma) \in E_C$ be the unique reference that corresponds to the edge (C, i, B) in $\text{dag}(D)$. We then define the embedding $\eta_p : \text{val}(B) \rightarrow \text{val}(A)$ as follows, where (q, v) is a node in $\text{val}(B)$ given by its B -representation so that q is a B -path (recall that the path pq is obtained by concatenating the paths p and q ; see Section 2.1):

$$\eta_p(q, v) = \begin{cases} (p'C, \sigma(i)) & \text{if } q = B \text{ and } v = \tau_B(j) \text{ for some } j \in [\text{rank}(B)], \\ (pq, v) & \text{otherwise.} \end{cases}$$

¹ The nodes in $\text{ran}(\tau_B)$, i.e., the contact nodes of \mathcal{U}_B , are excluded here, because they were already generated by some larger (with respect to the hierarchical order \succ_D) nonterminal.

We can extend this definition to the case $A = B$ (where $p = A$) by defining η_p as the identity map on $\text{val}(A) = \text{val}(B)$. If \mathcal{U} is the substructure of $\text{val}(B)$ induced by the set $U \subseteq \text{val}(B)$ then we write $\eta_p(\mathcal{U})$ for the substructure of $\text{val}(A)$ induced by the set $\eta_p(U)$. Note that in general we do not have $\eta_p(\mathcal{U}) \simeq \mathcal{U}$. For instance, if $\mathcal{U} = \text{val}(B)$ then in $\text{val}(A)$ there can be edges between contact nodes of $\text{val}(B)$ that are generated by a nonterminal C with $C \rightarrow_D B$.

Recall the definition of the lexicographic order on the set of all A -paths of $\text{dag}(D)$ for $A \in N$ (see Section 2.1). We define $\text{lex}_A(p)$ as the position of p in the lexicographically sorted list of all A -paths of $\text{dag}(D)$, where we start with 0 (i.e., $\text{lex}_A(A) = 0$; note that A is the empty path starting in A and hence the lexicographically smallest path among all A -paths). For $\text{lex}_S(p)$ we just write $\text{lex}(p)$. Later it will be convenient to represent the initial path component p of a D -representation (p, v) by the number $\text{lex}(p)$ and call $(\text{lex}(p), v)$ be the *lex-representation* of the node $a = (p, v) \in \text{val}(D)$. The number of initial paths in $\text{dag}(D)$ can be bounded by $2^{\mathcal{O}(|D|)}$: the number of initial-to-leaf paths in $\text{dag}(D)$ is bounded by $3^{|\text{dag}(D)|/3} \leq 3^{|D|/3}$ (this is implicitly shown in the proof of [11, Lemma 1]) and the number of all initial paths in D is bounded by twice the number of initial-to-leaf paths in D . Hence, the numbers $\text{lex}(p)$ have bit length $\mathcal{O}(|D|)$.

► **Example 6.** Recall the SLP D from Example 5 and $\text{dag}(D)$ shown to the right of D 's productions in Figure 1. Then the pairs (S, u) and (S, v) (recall that u and v are the two nodes of \mathcal{U}_S) represent the two red nodes of $\text{val}(D) = \text{val}(S)$, and $(S3B, w)$, where w is the green node in \mathcal{U}_B , represents the rightmost green node of $\text{val}(D)$. Its lex-representation is $(5, w)$ (there are six initial paths in $\text{dag}(D)$). As another example, the two leftmost (green) nodes of $\text{val}(D)$ are represented by the pairs $(S1A1B, w)$ and $(S2A1B, w)$ with the lex-representations $(2, w)$ and $(4, w)$, respectively. For the S -to- B path $p = S2A1B$ in $\text{dag}(D)$ we have $\eta_p(B, w) = (S2A1B, w)$ and $\eta_p(B, \tau_B(1)) = (S2A, \sigma(1))$, where (B, σ) is the only reference in E_A .

4.2 Register length in the compressed setting

In the following sections we will develop an enumeration algorithm for the set of all tuples in $\phi(\text{val}(D))$, where the SLP D is part of the input. Recall that $\text{val}(D)$ may contain $2^{\mathcal{O}(|D|)}$ many elements. In order to achieve constant delay, we therefore should set the register length in our algorithm to $\Theta(|D|)$ so that we can store elements of $\text{val}(D)$. This is in fact a standard assumption for algorithms on SLP-compressed objects. For instance, when dealing with SLP-compressed strings, one usually assumes that registers can store positions in the decompressed string. We only allow additions, subtractions and comparisons on these $\Theta(|D|)$ -bit registers and these operations take constant time (since we assume the uniform cost measure). For registers of length $\mathcal{O}(\log |D|)$ we will also allow pointer operations.

Note that a D -representation (p, v) needs $\mathcal{O}(|D|)$ many $\mathcal{O}(\log |D|)$ -bit registers, whereas its lex-representation $(\text{lex}(p), v)$ fits into two registers (one of length $\mathcal{O}(\log |D|)$).

5 FO-Enumeration over SLP-Compressed Degree-Bounded Structures

We now have all definitions available in order to state a more precise version of Theorem 1:

► **Theorem 7.** *Given an apex SLP D such that $\text{val}(D)$ is degree- d bounded and an FO-formula $\phi(x_1, \dots, x_k)$, we can enumerate the result set $\phi(\mathcal{U})$ with preprocessing time $f(d, |\phi|) \cdot |D|$ and delay $f(d, |\phi|)$ for some computable function f . All nodes of $\phi(\mathcal{U})$ are output in their lex-representation.*

Throughout Section 5 we fix $D = (\mathcal{R}, N, S, P)$ and $\phi(x_1, \dots, x_n)$ as in Theorem 7. Let $\text{qr}(\phi) = \nu$. W.l.o.g. we can assume that $d \geq 2$.

The general structure of our enumeration algorithm is the same as for the uncompressed setting. In particular, we also use Gaifman-locality to reduce to the problem of enumerating for a fixed $\mathcal{B} \in \mathcal{T}_{k,r}$ the set of all \mathcal{B} -tuples $\bar{a} \in \text{val}(D)^k$, which then reduces to the problem of enumerating for all consistent factorizations $\Lambda = (\mathcal{B}_1, \sigma_1, \dots, \mathcal{B}_m, \sigma_m)$ of \mathcal{B} the set of all m -tuples $\bar{b} \in \text{val}(D)^m$ that are admissible for Λ (see the beginning of Section 3).

Here, a first complication occurs: one important component of the above reduction for the uncompressed setting is that FO model checking on degree- d bounded structures can be done in time $|\mathcal{U}| \cdot f(d, |\phi|)$ [62]. For the SLP-compressed setting we do not have a linear time (i. e., in time $|D| \cdot f(d, |\phi|)$) model checking algorithm. Only an NL-algorithm for apex SLPs is known [39]. It is not hard to obtain a linear time algorithm from the NL-algorithm in [39]. Alternatively, one can also bypasses model checking; see the full version [43].

Consequently, as in the uncompressed setting, it suffices to consider a fixed consistent factorization $\Lambda = (\mathcal{B}_1, \sigma_1, \dots, \mathcal{B}_m, \sigma_m)$ of \mathcal{B} and to enumerate the set of all m -tuples in $\text{val}(D)$ that are admissible for Λ . As before we define the larger radius $\rho = 2rk - r + k - 1$; see (1).

5.1 Expansions of nonterminals

In this section we introduce the concept of ζ -expansions for a constant $\zeta \geq 1$ (later, ζ will be a constant of the form $f(d, |\phi|)$), which will be needed to transfer the enumeration algorithm for the uncompressed setting (Section 3.1) to the SLP-compressed setting. The idea is to apply the productions from D , starting with a nonterminal $A \in N$, until all nodes of $\text{val}(A)$ that have distance at most ζ from the nodes in the right-hand side of A (except for the contact nodes of A) are produced. For a nonterminal $A \in N$ we define

$$\text{In}_A = \{(A, v) : v \in \mathcal{U}_A \setminus \text{ran}(\tau_A)\} \subseteq \text{val}(A).$$

These are the internal nodes of $\text{val}(A)$ (written in A -representation) that are directly produced with the production $A \rightarrow (\mathcal{U}_A, \tau_A, E_A)$. Let a_1, \dots, a_m be a list of all nodes from In_A . We then define the ζ -expansion as the following induced substructure of $\text{val}(A)$:

$$\mathcal{E}_\zeta(A) = \mathcal{N}_{\text{val}(A), \zeta}(a_1, \dots, a_m).$$

We always assume that the nodes of $\mathcal{E}_\zeta(A)$ are represented by their A -representations. Let

$$\text{Bd}_{A, \zeta} = \{(A, v) : v \in \text{ran}(\tau_A)\} \cup \{a \in \text{val}(A) : \text{dist}_{\text{val}(A)}(\text{In}_A, a) = \zeta\} \subseteq \text{val}(A)$$

be the *boundary* of $\mathcal{E}_\zeta(A)$. A *valid substructure* of $\mathcal{E}_\zeta(A)$ is an induced substructure \mathcal{A} of $\mathcal{E}_\zeta(A)$ with $\mathcal{A} \cap \text{Bd}_{A, \zeta} = \emptyset \neq \mathcal{A} \cap \text{In}_A$. If \mathcal{A} is a valid substructure of $\mathcal{E}_\zeta(A)$ and p is an S -to- A path in $\text{dag}(D)$, then any neighbor of $\eta_p(\mathcal{A})$ in the graph $\mathcal{G}(\text{val}(D))$ belongs to $\eta_p(\mathcal{E}_\zeta(A))$. Moreover, $\eta_p(\mathcal{A}) \simeq \mathcal{A}$, since all contact nodes $(A, \tau_A(i))$ are excluded from a valid substructure of $\mathcal{E}_\zeta(A)$. In the following, we consider the radius $\zeta = 2\rho + 1$. For a nonterminal $A \in N$ we write $\mathcal{E}(A)$ for the expansion $\mathcal{E}_{2\rho+1}(A)$ in the rest of the paper.

Fix a ρ -neighborhood type \mathcal{B} . A node $a \in \mathcal{E}(A) \subseteq \text{val}(A)$ is called a *valid \mathcal{B} -node* in $\mathcal{E}(A)$ if (i) $\mathcal{N}_{\mathcal{E}(A), \rho}(a) \simeq \mathcal{B}$ and (ii) $\mathcal{N}_{\mathcal{E}(A), \rho}(a)$ is a valid substructure of $\mathcal{E}(A)$. We say that A is *\mathcal{B} -useful* if there is a valid \mathcal{B} -node in $\mathcal{E}(A)$. We consider now the following two sets:

- $S_1^\mathcal{B} = \{(p, a) : \exists A \in N : p \text{ is an } S\text{-to-}A \text{ path in } \text{dag}(D), a \text{ is a valid } \mathcal{B}\text{-node in } \mathcal{E}(A)\}$
- $S_2^\mathcal{B} = \{b \in \text{val}(D) : b \text{ is a } \mathcal{B}\text{-node}\}$

■ **Algorithm 2** enumeration of all \mathcal{B}_i -nodes.

```

1 for all initial paths  $p$  in  $\text{dag}(D)$  that end in a  $\mathcal{B}_i$ -useful nonterminal  $A$  do
2   for all  $(q, v) \in \mathcal{E}(A)$  that are valid  $\mathcal{B}_i$ -nodes in  $\mathcal{E}(A)$  do
3     return  $(\text{lex}(p), q, v)$ 

```

We define a mapping $h : S_1^{\mathcal{B}} \rightarrow \text{val}(D)$ as follows. Let $(p, a) \in S_1^{\mathcal{B}}$, where p is an S -to- A path in $\text{dag}(D)$ and let (q, v) be the A -representation of $a \in \mathcal{E}(A)$. We then define $h(p, a) = \eta_p(a) = (pq, v)$ (where the latter is a D -representation that we identify as usual with a node from $\text{val}(D)$). The following lemma is proved in the full version [43].

► **Lemma 8.** *The mapping h is a bijection from $S_1^{\mathcal{B}}$ to $S_2^{\mathcal{B}}$.*

5.2 Overview of the enumeration algorithm

Our goal is to carry out the algorithm described in Section 3.1, but in the compressed setting, i.e., by only using the apex SLP $D = (\mathcal{R}, N, S, P)$ instead of the explicit structure $\text{val}(D)$. As in the uncompressed setting, it suffices to consider a fixed (k, r) -neighborhood type $\mathcal{B} \in \mathcal{T}_{k,r}$ together with a fixed consistent factorization

$$\Lambda = (\mathcal{B}_1, \sigma_1, \dots, \mathcal{B}_m, \sigma_m) \quad (3)$$

of \mathcal{B} and to enumerate the set of all m -tuples in $\text{val}(D)$ that are admissible for Λ . In the following we sketch the algorithm; details can be found in the full version [43].

Enumeration of all \mathcal{B}_i -nodes. The algorithm for the uncompressed setting (Section 3) precomputes for every \mathcal{B}_i a list L_i of all \mathcal{B}_i -nodes of the structure \mathcal{U} . This is no longer possible in the compressed setting since the structure $\text{val}(D)$ is too big. However, as shown in Section 5.1, there is a bijection between the set of \mathcal{B}_i -nodes in $\text{val}(D)$ and the set of all pairs (p, a) , where p is an S -to- A path in $\text{dag}(D)$ for a \mathcal{B}_i -useful nonterminal A and a is a valid \mathcal{B}_i -node in $\mathcal{E}(A)$ that is written in its A -representation (q, v) . Hence, on a high level, instead of explicitly precomputing the lists L_i of all \mathcal{B}_i -nodes, we enumerate them with Algorithm 2.

To execute this algorithm we first have to compute in the preprocessing all expansions $\mathcal{E}(A)$ for a nonterminal A . This is easy: using a breath-first-search (BFS), we locally generate $\text{val}(A)$ starting with the nodes in In_A until all nodes $a \in \text{val}(A)$ with $\text{dist}_{\text{val}(A)}(\text{In}_A, a) \leq 2\rho + 1$ are generated. The size of $\mathcal{E}(A)$ is bounded by $|\mathcal{U}_A| \cdot f(d, |\phi|)$ (the size of a $(2\rho + 1)$ -sphere around a tuple of length at most $|\mathcal{U}_A|$ in a degree- d bounded structure) and can be constructed in time $|\mathcal{U}_A| \cdot f(d, |\phi|)$. Summing over all $A \in N$ shows that all $(2\rho + 1)$ -expansions can be precomputed in time $|D| \cdot f(d, |\phi|)$.

With the $\mathcal{E}(A)$ available, we can easily precompute brute-force the set of all valid \mathcal{B}_i -nodes in $\mathcal{E}(A)$ (needed in Line 2 of Algorithm 2) and then the set of all \mathcal{B}_i -useful nonterminals (needed in Line 1 of Algorithm 2). Recall that A is \mathcal{B}_i -useful iff there is a valid \mathcal{B}_i -node in $\mathcal{E}(A)$. Moreover, for every valid \mathcal{B}_i -node $c = (q, v) \in \mathcal{E}(A)$ we compute also an isomorphism $\pi_c : \mathcal{B}_i \rightarrow \mathcal{N}_{\mathcal{E}(A), \rho}(c_i)$. The time for this is bounded by $f(d, |\phi|)$ for one nonterminal A and hence by $|D| \cdot f(d, |\phi|)$ in total.

The most challenging part of Algorithm 2 is the enumeration of all initial paths p in $\text{dag}(D)$ that end in a \mathcal{B}_i -useful nonterminal (Line 1). Let \mathcal{P}_i be the set of these paths. In constant delay, we cannot afford to output a path $p \in \mathcal{P}_i$ as a list of edges (it does not fit into a constant number of registers in our machine model, see Section 4.2). That is why we return

the number $\text{lex}(p)$ (which fits into a single register in our machine model) in Line 3. The idea for constant-delay path enumeration is to run over all paths $p \in \mathcal{P}_i$ in lexicographical order and thereby maintain the number $\text{lex}(p)$. The path p is internally stored in a contracted form. If $\text{dag}(D)$ would be a binary dag, then we could use an enumeration algorithm from [42], where maximal subpaths of left (right, respectively) outgoing edges are contracted to single edges. In our setting, $\text{dag}(D)$ is not a binary dag, therefore we have to adapt the technique from [42] slightly; see [43].

In order to see how Algorithm 2 can be used to replace the precomputed lists L_i in Algorithm 1 for the uncompressed setting, a few additional points have to be clarified.

Producing the final output tuples. Note that for each enumerated \mathcal{B}_i -node $b_i \in \text{val}(D)$ we have to produce the partial k -tuple t_{b_i, σ_i} (then the final output tuple is $t_{b_1, \sigma_1} \sqcup t_{b_2, \sigma_2} \sqcup \dots \sqcup t_{b_m, \sigma_m}$). Let us first recall that in the uncompressed setting each partial k -tuple t_{b_i, σ_i} is defined by $t_{b_i, \sigma_i}(j) = \pi_{b_i}(\sigma_i(j))$ for all $j \in \text{dom}(\sigma_i)$, where $\pi_{b_i} : \mathcal{B}_i \rightarrow \mathcal{N}_{\mathcal{U}, \rho}(b_i)$ is a precomputed isomorphism. In the compressed setting, Algorithm 2 outputs every \mathcal{B}_i -node $b_i \in \text{val}(D)$ as a triple $(\text{lex}(p_i), q_i, v)$, where the initial path $p_i \in \mathcal{P}_i$ ends in some \mathcal{B}_i -useful nonterminal $A_i \in N$ and $c_i := (q_i, v_i)$ is a valid \mathcal{B}_i -node in $\mathcal{E}(A_i)$. Moreover, we have a precomputed isomorphism $\pi_{c_i} : \mathcal{B}_i \rightarrow \mathcal{N}_{\mathcal{E}(A_i), \rho}(c_i)$, which yields the isomorphism $\pi_{b_i} = \eta_{p_i} \circ \pi_{c_i} : \mathcal{B}_i \rightarrow \mathcal{N}_{\text{val}(D), \rho}(b_i)$. Then, for every $j \in \text{dom}(\sigma_i)$ we can easily compute the lex-representation of $\pi_{b_i}(\sigma_i(j))$. We first compute $\pi_{c_i}(\sigma_i(j))$ in its A_i -representation $(q_{i,j}, v_{i,j})$ using the precomputed mapping π_{c_i} . Then the lex-representation of $t_{b_i, \sigma_i}(j) = \pi_{b_i}(\sigma_i(j))$ is $(\text{lex}(p_i q_{i,j}), v_{i,j})$, where $\text{lex}(p_i q_{i,j}) = \text{lex}(p_i) + \text{lex}_{A_i}(q_{i,j})$. Here, $\text{lex}(p_i)$ is produced by Algorithm 2. The path $q_{i,j}$ has length at most $2\rho + 1$ (this is a consequence of the apex condition for D). Its lex-number $\text{lex}_{A_i}(q_{i,j})$ can be computed by summing at most $2\rho + 1$ many edge weights that were computed in the preprocessing phase.

Count total number of ρ -neighborhoods. In Section 3.1 we distinguish between short and long lists L_i . Since in our compressed setting, Algorithm 2 replaces the precomputed list L_i we have to count the number of triples produced by Algorithm 2 (of course, before we run the algorithm) in the preprocessing phase. This is easy: the number of output triples can be computed by summing over all \mathcal{B}_i -useful nonterminals A the product of (i) the number of S -to- A paths in $\text{dag}(D)$ and (ii) the number of valid \mathcal{B}_i -nodes in $\mathcal{E}(A)$. The latter can be computed in the preprocessing phase. Computing the number of S -to- A paths (for all $A \in N$) involves a top-down pass (starting in S) over $\text{dag}(D)$ with $|\text{dag}(D)| \leq |D|$ many additions on $\mathcal{O}(|D|)$ -bit numbers in total.

Checking distance constraints. Recall that we fixed the consistent factorization Λ from (3) of the fixed (k, r) -neighborhood type \mathcal{B} and want to enumerate all tuples $(b_1, \dots, b_m) \in \text{val}(D)^m$ that are admissible for Λ . The definition of an admissible tuple also requires to check whether $\text{dist}_{\text{val}(D)}(t_{b_i, \sigma_i}, t_{b_j, \sigma_j}) > 2r + 1$ for all $i \neq j$ (see (2)). The nodes b_i are enumerated with Algorithm 2, hence the following assumptions hold for all $i \in [m]$:

- b_i is given by a triple $(\text{lex}(p_i), q_i, v_i)$,
- p_i is an initial-to- A_i path in $\text{dag}(D)$ (for some \mathcal{B}_i -useful nonterminal A_i), and
- $c_i := (q_i, v_i)$ is a node (written in A_i -representation) from $\mathcal{E}(A_i)$ such that c_i has ρ -neighborhood type \mathcal{B}_i in $\mathcal{E}(A_i)$ and $\mathcal{N}_{\mathcal{E}(A_i), \rho}(c_i)$ is a valid substructure of $\mathcal{E}(A_i)$.

In a first step, we show that if $\text{dist}_{\text{val}(D)}(t_{b_i, \sigma_i}, t_{b_j, \sigma_j}) \leq 2r + 1$ then there is a path q of length at most $3\rho - r$ such that $p_i = p_j q$ or $p_j = p_i q$. For this, the apex property for D is important, since it lower bounds the distance between two nodes $a = (p, u)$ and $a' = (p', v')$ of $\text{val}(D)$ by the prefix distance between the paths p and p' (i.e., the total number of edges that do not belong to the longest common prefix of p and p').

We then proceed in two steps: We first check in time $f(d, |\phi|)$ whether $p_j = p_i q$ or $p_i = p_j q$ for some path q of length at most $3\rho - r$. For checking $p_j = p_i q$ (the case $p_i = p_j q$ is analogous) we check whether $p_j = p_i$ (by checking $\text{lex}(p_j) = \text{lex}(p_i)$) and if this is not the case, we repeatedly remove the last edge of p_j (for at most $3\rho - r$ times) and check whether the resulting path equals p_i . However, the whole procedure is complicated by the fact that p_i and p_j are given in a contracted form, where some subpaths are contracted to single edges (see the above paragraph on the path enumeration algorithm for $\text{dag}(D)$).

In the second step we have to check in time $f(d, |\phi|)$ whether $\text{dist}_{\text{val}(D)}(t_{b_i, \sigma_i}, t_{b_j, \sigma_j}) \leq 2r+1$, assuming that $p_j = p_i q$ for some path q of length at most $3\rho - r$. This boils down to checking, for every $b \in \text{ran}(t_{b_i, \sigma_i})$ and $b' \in \text{ran}(t_{b_j, \sigma_j})$, whether $\text{dist}_{\text{val}(D)}(b, b') \leq 2r+1$, which is the case iff $\text{dist}_{\text{val}(A_i)}(c, \eta_q(c')) \leq 2r+1$, where $c, c' \in \text{val}(A_i)$ correspond to b, b' in the sense that $\eta_{p_i}(c) = b$ and $\eta_{p_j}(c') = b'$. For this we locally construct $\mathcal{N}_{\text{val}(A_i), 2r+1}(c)$ by starting a BFS in c and then computing all elements of $\text{val}(A_i)$ with distance at most $2r+1$ from c just like we constructed all expansions $\mathcal{E}(A)$ (as explained above). This concludes our proof sketch for Theorem 7.

6 Conclusions and Outlook

We presented an enumeration algorithm for FO-queries on structures that are represented succinctly by apex SLPs. Assuming that the formula is fixed and the degree of the structure is bounded by a constant, the preprocessing time of our algorithm is linear and the delay is constant.

There are several possible directions into which our result can be extended. One option is to use more general formalisms for graph compression. Our SLPs are based on Courcelle's HR (hyperedge replacement) algebra, which is tightly related to tree width [12, Section 2.3]. Our SLPs can be viewed as dag-compressed expressions in the HR algebra, where the leaves can be arbitrary pointed structures; see [39] for more details. Another (and in some sense more general) graph algebra is the VR algebra, which is tightly related to clique width [12, Section 2.5]. It is straightforward to define a notion of SLPs based on the VR algebra and this leads to the question whether our result also holds for the resulting VR-algebra-SLPs.

Another interesting question is to what extent the results on enumeration for conjunctive queries [4, 7] can be extended to the compressed setting. In this context, it is interesting to note that model checking for a fixed existential FO-formula on SLP-compressed structures (without the apex restriction) belongs to NL. It would be interesting to see, whether the constant delay enumeration algorithm from [4] for free-connex acyclic conjunctive queries can be extended to SLP-compressed structures.

Finally, one may ask whether in our main result (Theorem 7) the apex restriction is really needed. More precisely, consider an SLP D such that $\text{val}(D)$ has degree d . Is it possible to construct from D in time $|D| \cdot f(d)$ an equivalent apex SLP D' of size $|D| \cdot f(d)$ for a computable function f ? If this is true then one could enforce the apex property in the preprocessing. In [17] it is shown that a set of graphs of bounded degree d that can be produced by a hyperedge replacement grammar (HRG) H can be also produced by an apex HRG, but the size blow-up is not analyzed with respect to the parameter d and the size of H .

References

- 1 Rajeev Alur, Michael Benedikt, Kousha Etessami, Patrice Godefroid, Thomas W. Reps, and Mihalis Yannakakis. Analysis of recursive state machines. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 27(4):86–818, 2005. doi:10.1145/1075382.1075387.

- 2 Rajeev Alur and Mihalis Yannakakis. Model checking of hierarchical state machines. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 23(3):273–303, 2001. doi:10.1145/503502.503503.
- 3 Guillaume Bagan. MSO queries on tree decomposable structures are computable with linear delay. In *Proceedings of the 20th International Workshop on Computer Science Logic, CSL 2006*, volume 4207 of *Lecture Notes in Computer Science*, pages 167–181. Springer, 2006. doi:10.1007/11874683_11.
- 4 Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *Proceedings of the 21st International Workshop on Computer Science Logic, CSL 2007*, volume 4646 of *Lecture Notes in Computer Science*, pages 208–222. Springer, 2007. doi:10.1007/978-3-540-74915-8_18.
- 5 Hideo Bannai, Momoko Hirayama, Danny Huc, Shunsuke Inenaga, Artur Jež, Markus Lohrey, and Carl Philipp Reh. The smallest grammar problem revisited. *IEEE Transaction on Information Theory*, 67(1):317–328, 2021. doi:10.1109/TIT.2020.3038147.
- 6 Michel Bauderon and Bruno Courcelle. Graph expressions and graph rewritings. *Mathematical System Theory*, 20(2-3):83–127, 1987. doi:10.1007/BF01692060.
- 7 Christoph Berkholz, Fabian Gerhardt, and Nicole Schweikardt. Constant delay enumeration for conjunctive queries: a tutorial. *ACM SIGLOG News*, 7(1):4–33, 2020. doi:10.1145/3385634.3385636.
- 8 Christoph Berkholz, Jens Keppeler, and Nicole Schweikardt. Answering FO+MOD queries under updates on bounded degree databases. *ACM Transactions on Database Systems*, 43(2):7:1–7:32, 2018. doi:10.1145/3232056.
- 9 Romain Brenguier, Stefan Göller, and Ocan Sankur. A comparison of succinctly represented finite-state systems. In *Proceedings of the 23rd International Conference on Concurrency Theory, CONCUR 2012*, volume 7454 of *Lecture Notes in Computer Science*, pages 147–161. Springer, 2012. doi:10.1007/978-3-642-32940-1_12.
- 10 Katrin Casel, Henning Fernau, Serge Gaspers, Benjamin Gras, and Markus L. Schmid. On the complexity of the smallest grammar problem over fixed alphabets. *Theory of Computing Systems*, 65(2):344–409, 2021. doi:10.1007/s00224-020-10013-w.
- 11 Moses Charikar, Eric Lehman, Ding Liu, Rina Panigrahy, Manoj Prabhakaran, Amit Sahai, and Abhi Shelat. The smallest grammar problem. *IEEE Transactions on Information Theory*, 51(7):2554–2576, 2005. doi:10.1109/TIT.2005.850116.
- 12 Bruno Courcelle and Joost Engelfriet. *Graph Structure and Monadic Second-Order Logic - A Language-Theoretic Approach*, volume 138 of *Encyclopedia of Mathematics and its Applications*. Cambridge University Press, 2012. doi:10.1017/CB09780511977619.
- 13 Arnaud Durand and Etienne Grandjean. First-order queries on structures of bounded degree are computable with constant delay. *ACM Transactions on Computational Logic*, 8(4):21, 2007. doi:10.1145/1276920.1276923.
- 14 Arnaud Durand, Nicole Schweikardt, and Luc Segoufin. Enumerating answers to first-order queries over databases of low degree. *Logical Methods in Computer Science*, 18(2), 2022. doi:10.46298/LMCS-18(2:7)2022.
- 15 Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite model theory*. Perspectives in Mathematical Logic. Springer, 1995. doi:10.1007/3-540-28788-4.
- 16 Joost Engelfriet. Context-free graph grammars. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages, Volume 3: Beyond Words*, pages 125–213. Springer, 1997. doi:10.1007/978-3-642-59126-6_3.
- 17 Joost Engelfriet, Linda Heyker, and George Leih. Context-free graph languages of bounded degree are generated by apex graph grammars. *Acta Informatica*, 31(4):341–378, 1994. doi:10.1007/BF01178511.
- 18 Joost Engelfriet and Grzegorz Rozenberg. A comparison of boundary graph grammars and context-free hypergraph grammars. *Information and Computation*, 84(2):163–206, 1990. doi:10.1016/0890-5401(90)90038-J.

- 19 Rachel Faran and Orna Kupferman. LTL with arithmetic and its applications in reasoning about hierarchical systems. In *LPAR-22. 22nd International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 57 of *EPiC Series in Computing*, pages 343–362. EasyChair, 2018. doi:10.29007/WPG3.
- 20 Rachel Faran and Orna Kupferman. A parametrized analysis of algorithms on hierarchical graphs. *International Journal on Foundations of Computer Science*, 30(6-7):979–1003, 2019. doi:10.1142/S0129054119400252.
- 21 Jörg Flum and Martin Grohe. *Parameterized Complexity Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2006. doi:10.1007/3-540-29953-X.
- 22 Moses Ganardi and Paweł Gawrychowski. Pattern matching on grammar-compressed strings in linear time. In *Proceedings of the 2022 ACM-SIAM Symposium on Discrete Algorithms, SODA 2022*, pages 2833–2846. SIAM, 2022. doi:10.1137/1.9781611977073.110.
- 23 Moses Ganardi, Artur Jeż, and Markus Lohrey. Balancing straight-line programs. *Journal of the ACM*, 68(4):27:1–27:40, 2021. doi:10.1145/3457389.
- 24 Adrià Gascón, Markus Lohrey, Sebastian Maneth, Carl Philipp Reh, and Kurt Sieber. Grammar-based compression of unranked trees. *Theory of Computing Systems*, 64(1):141–176, 2020. doi:10.1007/s00224-019-09942-y.
- 25 Ferenc Gécseg and Magnus Steinby. Tree languages. In Grzegorz Rozenberg and Arto Salomaa, editors, *Handbook of Formal Languages, Volume 3: Beyond Words*, pages 1–68. Springer, 1997. doi:10.1007/978-3-642-59126-6_1.
- 26 Stefan Göller and Markus Lohrey. Fixpoint logics on hierarchical structures. *Theory of Computing Systems*, 48(1):93–131, 2009. doi:10.1007/s00224-009-9227-1.
- 27 Annegret Habel and Hans-Jörg Kreowski. Some structural aspects of hypergraph languages generated by hyperedge replacement. In *Proceedings of the 4th Annual Symposium on Theoretical Aspects of Computer Science, STACS 1987*, volume 247 of *Lecture Notes in Computer Science*, pages 207–219. Springer, 1987. doi:10.1007/BFB0039608.
- 28 Wojciech Kazana and Luc Segoufin. First-order query evaluation on structures of bounded degree. *Logical Methods in Computer Science*, 7(2), 2011. doi:10.2168/LMCS-7(2:20)2011.
- 29 Wojciech Kazana and Luc Segoufin. Enumeration of first-order queries on classes of structures with bounded expansion. In *Proceedings of the 32nd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2013*, pages 297–308. ACM, 2013. doi:10.1145/2463664.2463667.
- 30 Wojciech Kazana and Luc Segoufin. Enumeration of monadic second-order queries on trees. *ACM Transactions on Computational Logic*, 14(4):25:1–25:12, 2013. doi:10.1145/2528928.
- 31 Benny Kimelfeld, Wim Martens, and Matthias Niewerth. A formal language perspective on factorized representations. In *Proceedings of the 28th International Conference on Database Theory, ICDT 2025*, volume 328 of *LIPICs*, pages 20:1–20:20. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2025. doi:10.4230/LIPICs.ICDT.2025.20.
- 32 Stephan Kreutzer and Anuj Dawar. Parameterized complexity of first-order logic. *Electronic Colloquium on Computational Complexity*, TR09-131, 2009. URL: <https://eccc.weizmann.ac.il/report/2009/131>.
- 33 N. Jesper Larsson and Alistair Moffat. Off-line dictionary-based compression. *Proceedings of the IEEE*, 88(11):1722–1732, 2000. doi:10.1109/5.892708.
- 34 Thomas Lengauer. Hierarchical planarity testing algorithms. *Journal of the ACM*, 36(3):474–509, 1989. doi:10.1145/65950.65952.
- 35 Thomas Lengauer and Klaus W. Wagner. The correlation between the complexities of the nonhierarchical and hierarchical versions of graph problems. *Journal of Computer and System Sciences*, 44:63–93, 1992. doi:10.1016/0022-0000(92)90004-3.
- 36 Thomas Lengauer and Egon Wanke. Efficient solution of connectivity problems on hierarchically defined graphs. *SIAM Journal on Computing*, 17(6):1063–1080, 1988. doi:10.1137/0217068.
- 37 Leonid Libkin. *Elements of Finite Model Theory*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2004. doi:10.1007/978-3-662-07003-1.

- 38 Markus Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012. doi:10.1515/gcc-2012-0016.
- 39 Markus Lohrey. Model-checking hierarchical structures. *Journal of Computer and System Sciences*, 78(2):461–490, 2012. doi:10.1016/J.JCSS.2011.05.006.
- 40 Markus Lohrey. Grammar-based tree compression. In *Proceedings of the 19th International Conference on Developments in Language Theory, DLT 2015*, volume 9168 of *Lecture Notes in Computer Science*, pages 46–57. Springer, 2015. doi:10.1007/978-3-319-21500-6_3.
- 41 Markus Lohrey, Sebastian Maneth, and Roy Mennicke. XML tree structure compression using RePair. *Information Systems*, 38(8):1150–1167, 2013. doi:10.1016/J.IS.2013.06.006.
- 42 Markus Lohrey, Sebastian Maneth, and Carl Philipp Reh. Constant-time tree traversal and subtree equality check for grammar-compressed trees. *Algorithmica*, 80(7):2082–2105, 2018. doi:10.1007/s00453-017-0331-3.
- 43 Markus Lohrey, Sebastian Maneth, and Markus L. Schmid. FO-query enumeration over SLP-compressed structures of bounded degree, 2025. arXiv:2506.19421.
- 44 Markus Lohrey, Sebastian Maneth, and Manfred Schmidt-Schauß. Parameter reduction and automata evaluation for grammar-compressed trees. *Journal of Computer and System Sciences*, 78(5):1651–1669, 2012. doi:10.1016/j.jcss.2012.03.003.
- 45 Markus Lohrey and Markus L. Schmid. Enumeration for MSO-queries on compressed trees. *Proceedings of the ACM on Management of Data*, 2(2):78, 2024. doi:10.1145/3651141.
- 46 Sebastian Maneth and Fabian Peternek. Grammar-based graph compression. *Information Systems*, 76:19–45, 2018. doi:10.1016/J.IS.2018.03.002.
- 47 Sebastian Maneth and Fabian Peternek. Constant delay traversal of grammar-compressed graphs with bounded rank. *Information and Computation*, 273:104520, 2020. doi:10.1016/J.IC.2020.104520.
- 48 Madhav V. Marathe, Harry B. Hunt III, Richard Edwin Stearns, and Venkatesh Radhakrishnan. Approximation algorithms for PSPACE-hard hierarchically and periodically specified problems. *SIAM Journal on Computing*, 27(5):1237–1261, 1998. doi:10.1137/S0097539795285254.
- 49 Madhav V. Marathe, Harry B. Hunt III, and S. S. Ravi. The complexity of approximation PSPACE-complete problems for hierarchical specifications. *Nordic Journal of Computing*, 1(3):275–316, 1994. URL: https://www.cs.helsinki.fi/njc/njc1_papers/number3/paper1.pdf.
- 50 Madhav V. Marathe, Venkatesh Radhakrishnan, Harry B. Hunt III, and S. S. Ravi. Hierarchically specified unit disk graphs. *Theoretical Computer Science*, 174(1–2):23–65, 1997. doi:10.1016/S0304-3975(96)00008-4.
- 51 Martin Muñoz and Cristian Riveros. Constant-delay enumeration for SLP-compressed documents. *Logical Methods in Computer Science*, 21(1), 2025. doi:10.46298/LMCS-21(1:17)2025.
- 52 Craig G. Nevill-Manning and Ian H. Witten. Identifying hierarchical structure in sequences: A linear-time algorithm. *Journal of Artificial Intelligence Research*, 7:67–82, 1997. doi:10.1613/JAIR.374.
- 53 Dan Olteanu. Factorized databases: A knowledge compilation perspective. In *Beyond NP, Papers from the 2016 AAAI Workshop, Phoenix, Arizona, USA, February 12, 2016*, 2016. URL: <http://www.aaai.org/ocs/index.php/WS/AAAIW16/paper/view/12638>.
- 54 Dan Olteanu and Maximilian Schleich. F: regression models over factorized views. *Proceedings of the VLDB Endowment*, 9(13):1573–1576, 2016. doi:10.14778/3007263.3007312.
- 55 Dan Olteanu and Maximilian Schleich. Factorized databases. *ACM SIGMOD Record*, 45(2):5–16, 2016. doi:10.1145/3003665.3003667.
- 56 Dan Olteanu and Jakub Závodný. Size bounds for factorised representations of query results. *ACM Transactions on Database Systems*, 40(1):2:1–2:44, 2015. doi:10.1145/2656335.
- 57 Leonid Peshkin. Structure induction by lossless graph compression. In *Proceedings of the 2007 Data Compression Conference, DCC 2007*, pages 53–62. IEEE Computer Society, 2007. doi:10.1109/DCC.2007.73.

- 58 William C. Rounds. Mappings and grammars on trees. *Mathematical System Theory*, 4(3):257–287, 1970. doi:10.1007/BF01695769.
- 59 Markus L. Schmid and Nicole Schweikardt. Spanner evaluation over SLP-compressed documents. In *Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2021*, pages 153–165. ACM, 2021. doi:10.1145/3452021.3458325.
- 60 Markus L. Schmid and Nicole Schweikardt. Query evaluation over SLP-represented document databases with complex document editing. In *Proceedings of the International Conference on Management of Data, PODS 2022*, pages 79–89. ACM, 2022. doi:10.1145/3517804.3524158.
- 61 Nicole Schweikardt, Luc Segoufin, and Alexandre Vigny. Enumeration for FO queries over nowhere dense graphs. *Journal of the ACM*, 69(3):22:1–22:37, 2022. doi:10.1145/3517035.
- 62 Detlef Seese. Linear time computable problems and first-order descriptions. *Mathematical Structures in Computer Science*, 6(6):505–526, 1996. doi:10.1017/S0960129500070079.
- 63 Luc Segoufin. Constant delay enumeration for conjunctive queries. *ACM SIGMOD Record*, 44(1):10–17, 2015. doi:10.1145/2783888.2783894.
- 64 Luc Segoufin and Alexandre Vigny. Constant delay enumeration for FO queries over databases with local bounded expansion. In *Proceedings of the 20th International Conference on Database Theory, ICDT 2017*, volume 68 of *LIPICs*, pages 20:1–20:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPICs.ICDT.2017.20.