


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

Markus Lohrey ✉ 

University of Siegen, Hölderlinstr. 3, D-57076 Siegen, Germany

Sebastian Maneth ✉ 

University of Bremen, Germany

Markus L. Schmid ✉ 

Humboldt-Universität zu Berlin, Unter den Linden 6, D-10099 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

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, 34, 40]. 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 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* [48, 54, 62, 63]. 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., [42].

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 [23].

SLPs are most famous for strings (see [5, 10, 23, 24] for some recent publications and [42] 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 [25, 44, 46, 47] and hypergraphs (i.e., general relational structures) [38, 43, 49, 50], 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 [61] (see also [27]) and to hyperedge replacement grammars [6, 29] (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, 30]. Moreover, FO model checking for *SLP-compressed* degree-bounded structures can be done efficiently [43]. 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 [43, 52].

► **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 [30] to the SLP-compressed setting. For this we have to overcome considerable technical barriers. The algorithm of [30] 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 [43], 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, 30], low degree [14], (locally) bounded expansion [31, 67], and structures that are tree-like [3, 32] or nowhere dense [64]; see [7, 66] 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 [30] is the most relevant one for our work.

Concerning other work on query enumeration on SLP-compressed data, we mention [54, 62, 63], which deals with constant delay enumeration for (a fragment of) MSO-queries on SLP-compressed strings, and [48], 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 [39] and have been further studied, e.g., in [9, 19, 20, 36, 37, 52, 51, 53]. Model

checking problems for SLP-compressed graphs have been studied in [43] for FO and MSO, [28] 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 [43] 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) [43]. 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 [60] following a “Sequitur scheme” [55] and in [50] following a “Repair scheme” [35] (see also [45]); 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 [33, 56, 57, 58, 59]). 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 $p = v_0 i_1 v_1 i_2 \dots v_{n-1} i_n v_n \in V(\mathbb{N}V)^*$ 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 Appendix A 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 the maximal nesting depth of quantifiers in ψ .

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 Appendix A 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 [30] 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 Appendix B.1), 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 Appendix B.2.

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 Appendix B.3):

► **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 $\bar{a} \in \mathcal{U}^m$ 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 $\text{extend}(s)$

```

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  $\text{extend}(sa)$ 
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 $\text{extend}(s)$ is called, $|s| < m$ holds. It is clear that the call $\text{extend}(\varepsilon)$ 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 $\text{extend}(\varepsilon)$, 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 concluded with moderate effort; see Appendix B.4.

► **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.

- 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 Appendix C. 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.

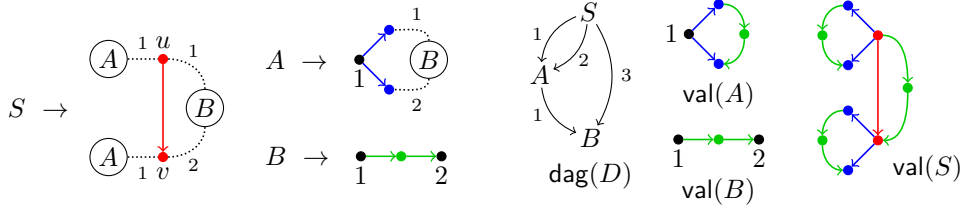
We define $\text{val}(D) = \text{val}(S)$. Since $\text{rank}(S) = 0$ it can be viewed as an ordinary (0-pointed) structure. It is not hard to see that $|\text{val}(D)| \leq 2^{\mathcal{O}(|D|)}$ and that this upper bound can be also reached. Thus, D can be seen as a compressed representation of the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

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 [52]. 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.



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

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 also Appendix D.3.1.

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

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.

the paths p and q ; see Section 2.1):

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

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 $\bar{a} \in \text{val}(D)^k$ (see Appendix B.1), 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 $|\mathcal{U}| \cdot f(d, |\phi|)$ [65]. 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 [43]. It is not hard to obtain a linear time algorithm from the NL-algorithm in [43], but there is an easier solution that bypasses model checking. We give all the details of how to perform the necessary reduction in the compressed setting in Appendix D.1.

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:

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  $(\text{lex}(p), q, v)$ 

```

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

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 proof of the following lemma can be found in Appendix D.2.

► **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 Appendix D.2.

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 details can be found in Appendix D.3.1. 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 $f(d, |\varphi|)$ for one nonterminal A and hence by $|D| \cdot f(d, |\phi|)$ in total. More details on this part can be found in Appendix D.3.2.

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 [46], 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 [46]. Details can be found in Appendix D.3.3 (which deals with the necessary preprocessing) and Appendix D.3.4 (which deals with the actual path enumeration).

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 $(\text{lex}(p_i), q_i, v_i)$, 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 (see Appendix D.3.3).

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. See Appendix D.3.5 for details.

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'); see Appendix D.3.6 for details.

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 the expansions in Appendix D.3.1. Details can be found in Appendix D.3.6. 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 [43] 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 Jez, 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 G. Rozenberg and A. 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 Haim Gaifman. On local and nonlocal properties. In *Proceedings of the Herbrand Symposium*, volume 107 of *Studies in Logic and the Foundations of Mathematics*, pages 105–135. Elsevier, 1982. doi:10.1016/S0049-237X(08)71879-2.
- 23 Moses Ganardi and Pawel 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.
- 24 Moses Ganardi, Artur Jez, and Markus Lohrey. Balancing straight-line programs. *Journal of the ACM*, 68(4):27:1–27:40, 2021. doi:10.1145/3457389.
- 25 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.
- 26 Leszek Gasieniec, Roman M. Kolpakov, Igor Potapov, and Paul Sant. Real-time traversal in grammar-based compressed files. In *Proceedings to the 2005 Data Compression Conference, DCC 2005*, page 458. IEEE Computer Society, 2005. doi:10.1109/DCC.2005.78.
- 27 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.
- 28 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.
- 29 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.
- 30 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.
- 31 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.
- 32 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.
- 33 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.

- 34 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>.
- 35 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.
- 36 Thomas Lengauer. Hierarchical planarity testing algorithms. *Journal of the Association for Computing Machinery*, 36(3):474–509, 1989. doi:10.1145/65950.65952.
- 37 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.
- 38 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.
- 39 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.
- 40 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.
- 41 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.
- 42 Markus Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups Complexity Cryptology*, 4(2):241–299, 2012. doi:10.1515/gcc-2012-0016.
- 43 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.
- 44 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.
- 45 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.
- 46 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.
- 47 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.
- 48 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.
- 49 Sebastian Maneth and Fabian Peternek. Grammar-based graph compression. *Information Systems*, 76:19–45, 2018. doi:10.1016/J.IS.2018.03.002.
- 50 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.
- 51 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.
- 52 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.
- 53 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.
- 54 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.

- 55 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.
- 56 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>.
- 57 Dan Olteanu and Maximilian Schleich. F: regression models over factorized views. *Proc. VLDB Endow.*, 9(13):1573–1576, 2016. doi:10.14778/3007263.3007312.
- 58 Dan Olteanu and Maximilian Schleich. Factorized databases. *SIGMOD Rec.*, 45(2):5–16, 2016. doi:10.1145/3003665.3003667.
- 59 Dan Olteanu and Jakub Závodný. Size bounds for factorised representations of query results. *ACM Trans. Database Syst.*, 40(1):2:1–2:44, 2015. doi:10.1145/2656335.
- 60 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.
- 61 William C. Rounds. Mappings and grammars on trees. *Mathematical System Theory*, 4(3):257–287, 1970. doi:10.1007/BF01695769.
- 62 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.
- 63 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.
- 64 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.
- 65 Detlef Seese. Linear time computable problems and first-order descriptions. *Mathematical Structures in Computer Science*, 6(6):505–526, 1996. doi:10.1017/S0960129500070079.
- 66 Luc Segoufin. Constant delay enumeration for conjunctive queries. *ACM SIGMOD Record*, 44(1):10–17, 2015. doi:10.1145/2783888.2783894.
- 67 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.

A Omitted Details from Section 2.2

We give some more details for first order logic. Let us fix a relational signature $\mathcal{R} = \{R_i : i \in I\}$. Atomic FO-formulas over the signature \mathcal{R} are of the form $x = y$ and $R(x_1, \dots, x_n)$, where $R \in \mathcal{R}$ has arity n and x, y, x_1, \dots, x_n are first-order variables ranging over elements of the universe. From these atomic FO-formulas we construct arbitrary FO-formulas over the signature \mathcal{R} using Boolean connectives (\neg, \wedge, \vee) and the (first-order) quantifiers $\exists x$ and $\forall x$ (for a variable x ranging over elements of the universe).

As a general convention, we also write $\psi(x_1, \dots, x_k)$ to denote that ψ is an FO-formula with free variables x_1, \dots, x_k . A *sentence* is an FO-formula without free variables. 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 $\text{qr}(\forall x\psi) = \text{qr}(\exists x\psi) = 1 + \text{qr}(\psi)$.

The model checking problem (and also the enumeration problem) for first-order logic over arbitrary signatures can be reduced to the case, where all arities are at most two as follows. Let us consider a structure $\mathcal{U} = (U, (R_i)_{i \in I})$ over an arbitrary relational signature

$\mathcal{R} = \{R_i : i \in I\}$, where the arity of R_i is α_i . Let $n = \max\{\alpha_i : i \in I\}$. We then define a new signature \mathcal{R}' consisting of the following relation symbols:

- a unary symbol R_U ,
- all R_i with $\alpha_i = 1$,
- a unary symbol R'_i for every $i \in I$ with $\alpha_i > 1$, and
- binary symbols E_1, \dots, E_n .

From \mathcal{U} we construct the new structure \mathcal{U}' over the signature \mathcal{R}' by taking the universe² $U \uplus \{b_{i,\bar{a}} : i \in I, \bar{a} \in R_i\}$ (the $b_{i,\bar{a}}$ are new elements) and defining the relations as follows:

- $R_U = U$,
- R_i is the same relation as in \mathcal{U} if $\alpha_i = 1$.
- $R'_i = \{b_{i,\bar{a}} : \bar{a} \in R_i\}$ for all $i \in I$ with $\alpha_i > 1$, and
- $E_j = \{(b_{i,\bar{a}}, a_j) : i \in I, \alpha_i > 1, \bar{a} = (a_1, \dots, a_{\alpha_i}) \in R_i\}$ for all $j \in [n]$.

Given an FO-formula $\psi(x_1, \dots, x_k)$ we construct the new FO-formula $\psi'(x_1, \dots, x_k) = \bigwedge_{1 \leq j \leq k} R_U(x_j) \wedge \tilde{\psi}(x_1, \dots, x_k)$ where $\tilde{\psi}(x_1, \dots, x_k)$ is obtained from $\psi(x_1, \dots, x_k)$ by restricting all quantifiers in ψ to elements from U (using the unary predicate R_U) and replacing every atomic subformula $R_i(y_1, \dots, y_{\alpha_i})$ with $\alpha_i > 1$ by $\exists x : R'_i(x) \wedge \bigwedge_{j=1}^{\alpha_i} E_j(x, y_i)$. We then have $\psi(\mathcal{U}) = \psi'(\mathcal{U}')$. In addition the following facts are important:

- $\text{qr}(\psi') \leq \text{qr}(\psi) + 1$.
- The degree of \mathcal{U}' is bounded by the degree of \mathcal{U} .
- The structure \mathcal{U}' can be constructed in linear time from \mathcal{U} and the formula ψ' can be constructed in linear time from ψ .

B Omitted Details from Section 3

We explain in this section the reduction mentioned at the beginning of Section 3 and start with Gaifman's locality theorem.

B.1 Gaifman's theorem and a corollary for degree- d bounded structures

Clearly, for every $r \geq 0$, there is an FO-formula $\delta_r(x, y)$ such that for every relational structure \mathcal{U} and all $a, b \in \mathcal{U}$ we have: $\mathcal{U} \models \delta_r(a, b)$ if and only if $\text{dist}_{\mathcal{U}}(a, b) \leq r$. Moreover, define $\delta_{k,r}(x_1, \dots, x_k, y) = \bigvee_{i=1}^k \delta_r(x_i, y)$. For an FO-formula ψ , a tuple of variables $\bar{z} = (z_1, \dots, z_k)$ and $r \geq 1$ let $\psi^{\bar{z},r}$ be the (\bar{z}, r) -relativization of ψ , i.e., the FO-formula obtained from ψ by restricting all quantifiers to the r -sphere around \bar{z} in the graph $\mathcal{G}(\mathcal{U})$. Let us define this more formally. For an FO-formula ψ , a tuple of variables $\bar{z} = (z_1, \dots, z_k)$ and $r \geq 1$ we define the FO-formula $\psi^{\bar{z},r}$ inductively by

- $R(x_1, \dots, x_n)^{\bar{z},r} = R(x_1, \dots, x_n)$ for $R \in \mathcal{R}$ and $(x = y)^{\bar{z},r} = (x = y)$,
- $(\neg\psi)^{\bar{z},r} = \neg(\psi^{\bar{z},r})$, $(\psi_1 \circ \psi_2)^{\bar{z},r} = \psi_1^{\bar{z},r} \circ \psi_2^{\bar{z},r}$ for $\circ \in \{\wedge, \vee\}$, and
- $(\exists x : \psi)^{\bar{z},r} = \exists x : (\delta_{k,r}(\bar{z}, x) \wedge \psi^{\bar{z},r})$ and $(\forall x : \psi)^{\bar{z},r} = \forall x : (\delta_{k,r}(\bar{z}, x) \rightarrow \psi^{\bar{z},r})$.

Note that the z_i are free variables in $\psi^{\bar{z},r}$.

We can now state Gaifman's theorem [22]; see also [41, Theorem 4.22].

► **Theorem 9.** *From a given FO-formula $\phi(x_1, \dots, x_k)$ with $\text{qr}(\phi) = \nu$, one can compute a logically equivalent Boolean combination of FO-formulas of the following form, where $r \leq 7^\nu$, $q \leq k + \nu$ and $\bar{x} = (x_1, \dots, x_k)$:*

- (i) $\psi^{\bar{x},r}$ where only the variables x_1, \dots, x_k are allowed to occur freely in ψ ,

² For sets A_1 and A_2 we write their union as $A_1 \uplus A_2$ if A_1 and A_2 are disjoint.

(ii) $\exists z_1 \cdots \exists z_q : \bigwedge_{1 \leq i < j \leq q} \neg \delta_{2r}(z_i, z_j) \wedge \bigwedge_{1 \leq i \leq q} \theta^{z_i, r}$ where θ is a sentence.

Gaifman's theorem is particularly useful for degree- d bounded structures (for some fixed d). We therefore continue to derive a well-known corollary of Theorem 9 for degree- d bounded structures.

We first observe that for every formula $\psi^{\bar{x}, r}$ from (i) in Theorem 9, whether $\mathcal{U} \models \psi^{\bar{x}, r}(\bar{a})$ for some $\bar{a} \in \mathcal{U}^k$ does not depend on the whole structure \mathcal{U} , but is completely determined by the (k, r) -neighborhood type of \bar{a} . This is due to the fact that all quantifiers in $\psi^{\bar{x}, r}$ are restricted to the r -sphere around its free variables \bar{x} . Consequently, one can replace every formula $\psi^{\bar{x}, r}$ from (i) in Theorem 9 by the finite disjunction $\bigvee_{i \in I} \psi^{\mathcal{B}_i}(\bar{x})$ where the $\mathcal{B}_i \in \mathcal{T}_{k, r}$ for $i \in I$ are exactly those (k, r) -neighborhood types such that $\mathcal{U} \models \psi^{\bar{x}, r}(\bar{a})$ if and only if \bar{a} is a \mathcal{B}_i -tuple for some $i \in I$. By going to conjunctive normal form (and using the fact that a conjunction $\psi^{\mathcal{B}_1}(\bar{x}) \wedge \psi^{\mathcal{B}_2}(\bar{x})$ for different $\mathcal{B}_1, \mathcal{B}_2 \in \mathcal{T}_{k, r}$ is always false) we end up with a disjunction

$$\bigvee_{i \in [m]} (\psi^{\mathcal{B}_i}(\bar{x}) \wedge \psi_i),$$

where for all $i \in [m]$, $\mathcal{B}_i \in \mathcal{T}_{k, r}$ and ψ_i is a Boolean combination of sentences of the form (ii) in Theorem 9 for some $r \leq 7^\nu$. This directly yields the following corollary:

► **Corollary 10.** *From a given d and an FO-formula $\phi(x_1, \dots, x_k)$ with $\text{qr}(\phi) = \nu$, one can compute a sequence $(\mathcal{B}_1, \psi_1, \dots, \mathcal{B}_m, \psi_m)$ for some $m = m(d, |\phi|)$ with the following properties:*

- (i) *For every $i \in [m]$ there is an $r \leq 7^\nu$ such that $\mathcal{B}_i \in \mathcal{T}_{k, r}$ and ψ_i is a Boolean combination of sentences of the form (ii) in Theorem 9.*
- (ii) *For every degree- d bounded structure \mathcal{U} and all $\bar{a} \in \mathcal{U}^k$ we have: $\mathcal{U} \models \phi(\bar{a})$ if and only if there is an $i \in [m]$ such that $\mathcal{U} \models \psi_i$ and \bar{a} is a \mathcal{B}_i -tuple.*

Based on Gaifman's locality theorem, Seese [65] proved the following:

► **Theorem 11 (degree-bounded model checking).** *For a given FO-sentence ϕ and a degree- d bounded structure \mathcal{U} , one can check in time $|\mathcal{U}| \cdot f(d, |\phi|)$, whether $\mathcal{U} \models \phi$.*

Recall that we have fixed at the beginning of Section 3 a relational signature $\mathcal{R} = \{R_i : i \in I\}$, constants $d \geq 2$ and ν , a degree- d bounded structure $\mathcal{U} = (U, (R_i)_{1 \leq i \leq l})$ over the signature \mathcal{R} , and an FO-formula $\phi(x_1, \dots, x_k)$ over the signature \mathcal{R} with $\text{qr}(\phi) = \nu$. Moreover, our goal is to enumerate the set $\phi(\mathcal{U})$ after a linear time preprocessing in constant delay. By Corollary 10 we can compute a sequence $(\mathcal{B}_1, \psi_1, \dots, \mathcal{B}_m, \psi_m)$ for some $m = m(d, |\phi|)$ with the properties from point (i) in the corollary and then enumerate all k -tuples \bar{a} such that $\mathcal{U} \models \psi_i$ and \bar{a} is a \mathcal{B}_i -tuple for some $i \in [m]$. Using Theorem 11 one can check in time $|\mathcal{U}| \cdot f(d, |\phi|)$ which of the ψ_i are true in \mathcal{U} . We then keep only those $\mathcal{B}_i \in \mathcal{T}_{k, r}$ such that $\mathcal{U} \models \psi_i$. Moreover, w.l.o.g. the remaining \mathcal{B}_i are different and for each of them we enumerate all \mathcal{B}_i -tuples. This will not create duplicates (since each k -tuple has a unique (k, r) -neighborhood type).

B.2 Reduction to tuples with pairwise different components

By the previous discussion, it suffices to enumerate for a fixed $\mathcal{B} \in \mathcal{T}_{k, r}$ the set of all \mathcal{B} -tuples with delay $f(|\mathcal{B}|)$ after preprocessing time $|\mathcal{U}| \cdot f(|\mathcal{B}|)$. Recall that in Section 3 we assumed that the sphere center constant c_i is interpreted by the element $i \in [k]$ in \mathcal{B} . But this means that all \mathcal{B} -tuples have pairwise different components.

We next show that this is not a real restriction. More precisely, we claim that w.l.o.g. we can replace our initial FO-formula $\phi(x_1, \dots, x_k)$ by

$$\phi' = \phi \wedge \bigwedge_{i,j \in [k], i \neq j} x_i \neq x_j, \quad (4)$$

so that the entries a_i in each enumerated output tuple (a_1, \dots, a_k) are pairwise different. For this, the enumeration algorithm runs in an outermost loop through all equivalence relations \equiv on $[k]$. For each such equivalence relation \equiv and $i \in [k]$ let $\mu_{\equiv}(i) = \min([i]_{\equiv})$ be the minimal representative from the equivalence class $[i]_{\equiv} = \{j \in [k] : i \equiv j\}$ and let $I_{\equiv} = \mu_{\equiv}([k])$ be the image of μ_{\equiv} . Then we define the FO-formula ϕ_{\equiv} by replacing every free occurrence of a variable x_i in ϕ by $x_{\mu_{\equiv}(i)}$ and start the enumeration of the output tuples for the formula

$$\phi'_{\equiv} = \phi_{\equiv} \wedge \bigwedge_{i,j \in I_{\equiv}, i \neq j} x_i \neq x_j.$$

From every enumerated $|I_{\equiv}|$ -tuple we obtain a tuple in $\phi(\mathcal{U})$ by duplicating entries suitably (according to the equivalence relation \equiv). This shows that it suffices to consider a formula of the form ϕ' in (4).

We now apply Corollary 10 to the new formula ϕ' . By the additional disequality constraints $x_i \neq x_j$ in ϕ' we can assume that in the resulting (k, r) -neighborhood types $\mathcal{B}_1, \dots, \mathcal{B}_m$ the elements a_i ($i \in [k]$) that interpret the sphere center constants c_i are pairwise different. We can then w.l.o.g. assume that c_i is interpreted by the element $i \in [k]$ (recall that the universe of a (k, r) -neighborhood type is of the form $[\ell]$).

B.3 Proofs of Lemmas 2 and 3

As in Section 3 we fix a (k, r) -neighborhood type \mathcal{B} (with the sphere centers $1, \dots, k$) that splits into the connected components $\mathcal{C}_1^{\mathcal{B}}, \dots, \mathcal{C}_m^{\mathcal{B}}$. Moreover, for every \mathcal{B}' -node $a \in \mathcal{U}$ we fix an isomorphism $\pi_a : \mathcal{B}' \rightarrow \mathcal{N}_{\mathcal{U}, \rho}(a)$.

► **Lemma 12** (Lemma 2 restated). *If $\Lambda = (\mathcal{B}_1, \sigma_1, \dots, \mathcal{B}_m, \sigma_m)$ is a consistent factorization of \mathcal{B} and $\bar{b} \in \mathcal{U}^m$ is admissible for Λ then $\Lambda(\bar{b})$ is a \mathcal{B} -tuple.*

Proof. Let $D_i = \mathcal{C}_i^{\mathcal{B}} \cap [k]$ and let $\bar{b} = (b_1, \dots, b_m)$. We have $\mathcal{N}_{\mathcal{U}, \rho}(b_i) \simeq \mathcal{B}_i$. Moreover, $\Lambda(\bar{b}) = t_{b_1, \sigma_1} \sqcup t_{b_2, \sigma_2} \sqcup \dots \sqcup t_{b_m, \sigma_m}$, where $t_{b_i, \sigma_i}(j) = \pi_{b_i}(\sigma_i(j))$ for all $j \in D_i$. Since $\mathcal{N}_{\mathcal{B}_i, r}(\sigma_i) \simeq \mathcal{C}_i^{\mathcal{B}}$ we obtain $\mathcal{N}_{\mathcal{U}, r}(t_{b_i, \sigma_i}) \simeq \mathcal{C}_i^{\mathcal{B}}$ for all $i \in [m]$. Moreover, from (2) it follows that the $\mathcal{N}_{\mathcal{U}, r}(t_{b_i, \sigma_i})$ are the connected components of $\mathcal{N}_{\mathcal{U}, r}(t_{b_1, \sigma_1} \sqcup \dots \sqcup t_{b_m, \sigma_m}) = \mathcal{N}_{\mathcal{U}, r}(\Lambda(\bar{b}))$. Hence, $\Lambda(\bar{b})$ is a \mathcal{B} -tuple. ◀

► **Lemma 13** (Lemma 3 restated). *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 $\bar{a} = \Lambda(\bar{b})$.*

Proof. Let $D_i = \mathcal{C}_i^{\mathcal{B}} \cap [k]$ and let $n_i \in \min(D_i)$. Moreover, let $\bar{a} = (a_1, \dots, a_k)$ and define the partial k -tuple $t_i : [k] \rightarrow \mathcal{U}$ with domain D_i by $t_i(j) = a_j$ for $j \in D_i$. Since \mathcal{B} is the (k, r) -neighborhood type of \bar{a} , we have $\mathcal{C}_i^{\mathcal{B}} \simeq \mathcal{N}_{\mathcal{U}, r}(t_i) \subseteq \mathcal{N}_{\mathcal{U}, \rho}(a_{n_i})$ and $\text{dist}_{\mathcal{U}}(t_i, t_j) > 2r + 1$ for all $i, j \in [m]$ with $i \neq j$.

We take $\bar{b} = (b_1, \dots, b_m)$ with $b_i := a_{n_i}$ for $i \in [m]$. Moreover, for every $i \in [m]$, let \mathcal{B}_i be the ρ -neighborhood type of b_i in \mathcal{U} and define the partial k -tuple $\sigma_i : [k] \rightarrow \mathcal{B}_i$ with domain D_i by $\sigma_i(j) = \pi_{b_i}^{-1}(a_j)$ for $j \in D_i$. Moreover, define $\Lambda = (\mathcal{B}_1, \sigma_1, \dots, \mathcal{B}_m, \sigma_m)$. The definition of σ_i implies $t_i = t_{b_i, \sigma_i}$ for all $i \in [m]$ and hence $\bar{a} = \Lambda(\bar{b})$. Note that $\sigma_i(n_i) = \pi_{b_i}^{-1}(a_{n_i}) =$

$\pi_{b_i}^{-1}(b_i) = 1$ and $\mathcal{N}_{\mathcal{B}_i, r}(\sigma_i) \simeq \mathcal{N}_{\mathcal{U}, r}(t_i) \simeq \mathcal{C}_i^{\mathcal{B}}$. Hence, Λ is a consistent factorization of \mathcal{B} . Moreover, (b_1, \dots, b_m) is admissible for Λ : b_i is a \mathcal{B}_i -node and $\text{dist}_{\mathcal{U}}(t_{b_i, \sigma_i}, t_{b_j, \sigma_j}) = \text{dist}_{\mathcal{U}}(t_i, t_j) > 2r + 1$ for all $i, j \in [m]$ with $i \neq j$.

To show uniqueness, assume that $\Lambda' = (\mathcal{B}'_1, \sigma'_1, \dots, \mathcal{B}'_m, \sigma'_m)$ is a consistent factorization of \mathcal{B} and $\bar{b}' \in \mathcal{U}^m$ is admissible for Λ' such that $\bar{a} = \Lambda(\bar{b}')$. First, notice that $\bar{b}' = (a_{n_1}, \dots, a_{n_m}) = \bar{b}$. Moreover, we must have $\mathcal{B}_i = \mathcal{B}'_i$ (both are the (k, r) -neighborhood type of a_{n_i}). Finally, for all $i \in [m]$ and $j \in D_j$ we have $\sigma_i(j) = \pi_{b_i}^{-1}(a_j) = \pi_{b'_i}^{-1}(a_j) = \sigma'_i(j)$ for all $j \in D_i$. \blacktriangleleft

B.4 Proof of Lemma 4

We now formally prove that the enumeration algorithm described in Section 3.1 has a delay of $f(d, |\phi|)$.

Recall from Section 3.1 that we do this under the assumption that we can check whether a stack s is admissible in time $f(d, |\phi|)$. This will be explained in detail for the more general SLP-compressed setting in Section 5. Hence, in the following, we only count the number of steps in the DFLR-traversal, i.e., the total number of iterations of the for-loop in Algorithm 1 over all recursion levels. In the following, we use all notations introduced in Section 3.1.

We first make the following general observation: Let $s = a_1 \cdots a_\ell$ with $\ell < m$ be an admissible stack. If, for some element $a_{\ell+1} \in L_{\ell+1}$, the stack $sa_{\ell+1}$ is not admissible, then

$$\text{dist}_{\mathcal{U}}(t_{a_i, \sigma_i}, t_{a_{\ell+1}, \sigma_{\ell+1}}) \leq 2r + 1$$

for some $i \in [\ell]$. This implies $\text{dist}(a_i, a_{\ell+1}) \leq 2\rho + 2r + 1$. The number of elements $a \in \mathcal{U}$ such that $\text{dist}(a_i, a) \leq 2\rho + 2r + 1$ for some $i \in [\ell]$ is bounded by $\ell \cdot d^{2\rho+2r+2} \leq k \cdot d^{2\rho+2r+2}$. Hence, we obtain the following observation:

► **Observation 14.** *For a fixed admissible stack s of length $\ell < m$, the list $L_{\ell+1}$ contains at most $\ell \cdot d^{2\rho+2r+2} \leq k \cdot d^{2\rho+2r+2}$ elements a such that sa is not admissible.*

In order to bound the delay, let us first consider the time that elapses from the call $\text{extend}(\varepsilon)$ to the first output (or the termination of the algorithm). Recall that the lists L_1, \dots, L_q are short. Hence, there are at most $c := (k \cdot d^{2\rho+2r+2})^{q+1} \leq k^{k+1} d^{(2\rho+2r+2)(k+1)}$ many stacks s with $|s| \leq q$. Hence, either the algorithm terminates after at most c many steps in the DFLR-traversal or it reaches after at most c steps a stack sa where $|s| = q$ and a is the first element of the list L_{q+1} . Then, by Observation 14, the algorithm reaches after $(m - q) \cdot k \cdot d^{2\rho+2r+2} \leq k^2 \cdot d^{2\rho+2r+2}$ further steps an admissible stack of length m (which is then written to the output).

Now, let us assume that we output an admissible stack s of length m . We have to bound the number of steps of the algorithm until the next output is generated (or until the termination of the algorithm if no further output is produced). The algorithm will first reduce the length of the stack s (by doing return-statements in line 4). First assume that the length of the stack does not go below $q + 1$. Then, Observation 14 shows that the next output happens after at most $2(m - q) \cdot k \cdot d^{2\rho+2r+2}$ DFLR-traversal steps, i.e., $(m - q) \cdot k \cdot d^{2\rho+2r+2}$ steps in the first phase, where the stack shrinks, followed by $(m - q) \cdot k \cdot d^{2\rho+2r+2}$ steps where the stack grows. On the other hand, if the stack length reaches q (this can only happen after at most $(m - q) \cdot k \cdot d^{2\rho+2r+2}$ steps) then either the algorithm terminates after c more steps or after $c + (m - q) \cdot k \cdot d^{2\rho+2r+2}$ more steps the next output occurs. Hence, the delay can be bounded by $f(d, |\phi|)$.

C Omitted details from Section 4

Consider an SLP $D = (\mathcal{R}, N, S, P)$ and a nonterminal $A \in N$. In Section 4, we have explained on an intuitive level how the structure $\text{val}(A)$ is defined. Let us now define this more formally. We start with a few general definitions.

Let \equiv be an equivalence relation on a set U . Then, for $a \in U$, $[a]_{\equiv} = \{b \in U : a \equiv b\}$ denotes the equivalence class containing a . With $[U]_{\equiv}$ we denote the set of all equivalence classes. With $\pi_{\equiv} : U \rightarrow [U]_{\equiv}$ we denote the function with $\pi_{\equiv}(a) = [a]_{\equiv}$ for all $a \in U$.

For a relational structure $\mathcal{U} = (U, (R_i)_{i \in I})$ and an equivalence relation \equiv on U we define the quotient $\mathcal{U}/_{\equiv} = ([U]_{\equiv}, (R_i/_{\equiv})_{i \in I})$, where

$$R_i/_{\equiv} = \{(\pi_{\equiv}(v_1), \dots, \pi_{\equiv}(v_{\alpha_i})) : (v_1, \dots, v_{\alpha_i}) \in R_i\}.$$

For two structures $\mathcal{U}_1 = (U_1, (R_{i,1})_{i \in I})$ and $\mathcal{U}_2 = (U_2, (R_{i,2})_{i \in I})$ over the same signature \mathcal{R} and with disjoint universes U_1 and U_2 , respectively, we define the disjoint union

$$\mathcal{U}_1 \oplus \mathcal{U}_2 = (U_1 \uplus U_2, (R_{i,1} \uplus R_{i,2})_{i \in I}).$$

With these definitions we can now define the $\text{rank}(A)$ -pointed structure $\text{val}(A)$ for $A \in N$ as follows. Assume that $E_A = \{(A_i, \sigma_i) : i \in [n]\}$. Recall that this is a multiset, i.e., we may have $(A_i, \sigma_i) = (A_j, \sigma_j)$ for $i \neq j$. Assume now that $\text{val}(A_i) = (\mathcal{U}_i, \tau_i)$ is already defined for every $i \in [n]$. Then

$$\text{val}(A) = ((\mathcal{U}_A \oplus \mathcal{U}_1 \oplus \dots \oplus \mathcal{U}_n)/_{\equiv}, \pi_{\equiv} \circ \tau_A),$$

where \equiv is the smallest equivalence relation on the universe of $\mathcal{U}_A \oplus \mathcal{U}_1 \oplus \dots \oplus \mathcal{U}_n$, which contains $\{(\sigma_i(j), \tau_i(j)) : i \in [n], j \in [\text{rank}(A_i)]\}$. Note that $\text{val}(A) = (\mathcal{U}_A, \tau_A)$ if E_A is empty.

Next, let us take a closer look at our general assumption that the signature \mathcal{R} only contains relation symbols of arity at most two (see the end of Section 2.2). We have seen in Appendix A how we can transform an FO-formula ψ and a relational structure \mathcal{U} over a signature \mathcal{R} (with relation symbols of arbitrary arity) into an FO-formula ψ' and a relational structure \mathcal{U}' over a signature \mathcal{R}' with all relation symbols of arity at most two, such that $\psi(\mathcal{U}) = \psi'(\mathcal{U}')$ and, furthermore, $\text{qr}(\psi') = \text{qr}(\psi) + 1$.

For the compressed setting, we must argue that this reduction can be performed in linear time also in the case where \mathcal{U} is compressed by an SLP, which is rather simple: From an SLP D (whose signature \mathcal{R} contains relation symbols of arbitrary arity) one can construct a new SLP D' (whose signature contains only relation symbols of arity at most two) such that $\text{val}(D') = \text{val}(D)'$. For this it suffices to replace every structure \mathcal{U}_A (for A a nonterminal of D) by \mathcal{U}'_A (where \mathcal{U}'_A is obtained from \mathcal{U}_A by the construction described in Appendix A). Therefore, it is justified also in the compressed setting to consider only SLPs that produce structures with relations of arity at most two.

D Omitted details from Section 5

D.1 The Gaifman locality reduction in the compressed setting

In this section, we explain why the reduction described in Appendix B.1 is also applicable in the compressed setting. As in the uncompressed setting we want to reduce the enumeration of the result set $\phi(\text{val}(D))$ for a given FO-formula $\phi(x_1, \dots, x_k)$ and an SLP D to the enumeration of all \mathcal{B} -tuples in $\text{val}(D)^k$ for some (k, r) -neighborhood type \mathcal{B} with $r \leq 7^\nu$ (where $\nu = \text{qr}(\phi)$).

In the uncompressed setting, we apply Corollary 10 to the formula $\phi(x_1, \dots, x_k)$ and obtain the list $(\mathcal{B}_1, \psi_1, \dots, \mathcal{B}_m, \psi_m)$ with the properties described in Corollary 10. We then check for each sentence ψ_i , whether it is true in \mathcal{U} (using Theorem 11). We keep only those neighborhood types \mathcal{B}_i such that $\mathcal{U} \models \psi_i$ holds and then run the enumeration algorithm for those \mathcal{B}_i .

For the SLP-compressed setting we have no linear time model checking algorithm (for a fixed FO-sentence) in the spirit of Theorem 11 available; only an NL-algorithm for apex SLPs is known [43]. With some effort, one can adapt this NL-algorithm to obtain a linear time algorithm, but there is an easier solution that we explain below.

Assume that we have already an algorithm that enumerates after preprocessing time $|D| \cdot f(d, \phi)$ in delay $f(d, \phi)$ all \mathcal{B} -tuples from $\text{val}(D)^k$ for a fixed (k, r) -neighborhood type \mathcal{B} (for some $r \leq 7^\nu$). We then use this algorithm for

- (i) testing whether $\text{val}(D) \models \psi_i$ for a ψ_i from Corollary 10 and, in case this holds,
- (ii) for enumerating all \mathcal{B}_i -tuples from $\text{val}(D)^k$.

Let us show how we one achieve (i). Consider one of the sentences ψ_i from Corollary 10. It is a Boolean combination of sentences of the form

$$\exists z_1 \dots \exists z_q : \bigwedge_{1 \leq i < j \leq q} \neg \delta_{2r}(z_i, z_j) \wedge \bigwedge_{1 \leq i \leq q} \theta^{z_i, r} \quad (5)$$

with $r \leq 7^\nu$, $q \leq k + \nu$ and a sentence θ . So, we have to check whether there exist at least q many disjoint r -neighborhoods (of single nodes) for which the FO-property $\theta^{z, r}$ holds. From $\theta^{z, r}$ one can compute a finite list $\mathcal{B}'_1, \dots, \mathcal{B}'_s$ (for some $s = f(d, |\phi|)$) of r -neighborhood types such that $\theta^{z, r}(z)$ is equivalent to the fact that z has one of the r -neighborhood types $\mathcal{B}'_1, \dots, \mathcal{B}'_s$. Then, by using the assumed enumeration algorithm, we enumerate the set L'_1 of all \mathcal{B}'_1 -nodes in $\text{val}(D)$, then the set L'_2 of all \mathcal{B}'_2 -nodes in $\text{val}(D)$ and so on, until we either have enumerated $q \cdot d^{2r+1}$ many nodes in total, or the enumeration of L'_s terminates. Hence, in total, we enumerate at most $q \cdot d^{2r+1}$ many nodes. The preprocessing for these enumerations takes time $|D| \cdot f(d, \phi)$ and all enumerations take time $f(d, |\phi|)$ in total. If we actually enumerate $q \cdot d^{2r+1}$ many nodes, then, since $\text{val}(D)$ is degree- d bounded, there must exist q disjoint r -neighborhoods in $\text{val}(D)$ with a type from $\{\mathcal{B}'_1, \dots, \mathcal{B}'_s\}$. Indeed, if we have a list of $q \cdot d^{2r+1}$ enumerated nodes, then we pick the first element a_1 and remove from the list all nodes from the sphere $\mathcal{S}_{\text{val}(D), 2r}(a_1)$. Then we pick the first element a_2 from the remaining list and remove all nodes from the sphere $\mathcal{S}_{\text{val}(D), 2r}(a_2)$. We proceed like this and pick elements a_3, a_4, \dots until the list is empty. Obviously, the picked nodes have pairwise distance of at least $2r + 1$ and have all an r -neighborhood type \mathcal{B}'_i for some $i \in [s]$. Moreover, since we remove in each step at most d^{2r+1} nodes (namely the sphere $\mathcal{S}_{\text{val}(D), 2r}(a_i)$), it is guaranteed that we select at least q such nodes (recall that we start with a list of $q \cdot d^{2r+1}$ nodes). Hence, the sentence (5) holds in $\text{val}(D)$.

If, on the other hand, we enumerate strictly fewer than $q \cdot d^{2r+1}$ nodes, then we can check for all pairwise distinct enumerated nodes a, b whether the distance in $\mathcal{G}(\text{val}(D))$ between a and b is at most $2r + 1$. It is shown in Appendix D.3.6 that this can be done in time $f(d, r)$; see also Remark 25.

D.2 Omitted details from Section 5.1

D.2.1 Top-level nodes and subsets

In this section we introduce some concepts that are useful for the proof of Lemma 8 in the next section.

For a nonterminal A , we say that a node $a = (p, v)$ of $\text{val}(D)$ is *A-produced*, if the path p contains A . The unique S -to- A prefix of p is then called the *A-origin* of a . Intuitively speaking, every A -produced node of $\text{val}(D)$ is produced by an occurrence of the nonterminal A in the derivation tree, and if two such nodes have the same A -origin, then they are produced by the same occurrence of A . We say that a is *top-level A-produced* if p ends in A . By the definition of D -representations (see Section 4.1), this means that $v \in \mathcal{U}_A \setminus \text{ran}(\tau_A)$. Moreover, for every node a of $\text{val}(D)$ there is a unique nonterminal A such that a is a top-level A -produced.

A subset U of $\text{val}(D)$ is an *A-subset*, if all its nodes are A -produced with the same A -origin, which is then also called the *A-origin* of U . If additionally there is at least one node of U that is top-level A -produced, then we say that U is a *top-level A-set*. Note that this is equivalent to the existence of at least one node in U with a D -representation (p, v) , where p ends in A and v is an internal node of \mathcal{U}_A .

► **Lemma 15.** *If U is a top-level A -subset, then there is no other nonterminal B such that U is a top-level B -subset.*

Proof. Assume that for some nonterminals $A, B \in N$ with $A \neq B$, U is both a top-level A -subset with A -origin p and a top-level B -subset with B -origin q . This means that p is a proper prefix of q or the other way around; without loss of generality, we assume that $p = qp'$, where p' is a B -to- A path. This means that every node in U has the A -origin qp' , which implies that there is no top-level B -produced node in U ; a contradiction to the assumption that U is a top-level B -structure. ◀

Note that even though for every subset U of $\text{val}(D)$ there is a nonterminal A such that U is an A -subset (for example, every subset is an S -subset), U is not necessarily a top-level A -subset for some nonterminal A . If, however, U is a *connected* subset (in the sense that the substructure induced by U is a connected set in the graph $\text{val}(D)$), then we have the following:

► **Lemma 16.** *Let D be an apex SLP and U be a connected subset of $\text{val}(D)$. Then there is a unique $A \in N$ such that U is a top-level A -subset.*

Proof. Let $(p_1, v_1), \dots, (p_n, v_n)$ be the D -representations of all the elements of U and let p be the longest common prefix of the initial paths p_1, \dots, p_n . Assume that the initial path p is an S -to- A path. Hence, all nodes of U are A -produced with the same A -origin p , which means that U is an A -subset. We next show that it is also a top-level A -subset, i.e., we show that U contains a node with D -representation (p, v) , where v is an internal node of \mathcal{U}_A .

In order to get a contradiction, assume that such a node does not exist in U . By the choice of p as a longest common prefix, there must exist $i, j \in [n]$ with $i \neq j$ such that $p_i = p k_i B_i q_i$ and $p_j = p k_j B_j q_j$ with $k_i \neq k_j$. Let (B_i, σ_i) be the k_i^{th} reference in E_A and (B_j, σ_j) be the k_j^{th} reference in E_A . Since U is connected there is a path from $(p k_i B_i q_i, v_i)$ to $(p k_j B_j q_j, v_j)$ in the undirected graph $\mathcal{G}(\text{val}(D))$. This path must contain a node $(p, \sigma_i(\ell))$ from $\text{val}(A)$ (as well as a node $(p, \sigma_j(\ell))$ from $\text{val}(A)$). Since D is an apex SLP, $\sigma_i(\ell)$ is internal in \mathcal{U}_A , which gives the desired contradiction.

This means that U is a top-level A -subset and by Lemma 15, U cannot be a top-level B -subset for some $B \in N$ with $A \neq B$. ◀

Note that if U is the universe of a valid substructure (see Section 5.1) of $\mathcal{E}_\zeta(A)$, then for every S -to- A path p , the set of $\eta_p(U)$ is a top-level A -subset.

Also note that if U is a subset of $\text{val}(A)$, p is an S -to- A path in $\text{val}(D)$, and $\eta_p(U)$ is a top-level A -subset, then the substructure of $\text{val}(A)$ induced by U is isomorphic to the

substructure of $\text{val}(D)$ induced by $\eta_p(U)$. The reason is that U contains none of the contact nodes of $\text{val}(A)$. In particular, if \mathcal{A} is a valid substructure of the expansion $\mathcal{E}_\zeta(A)$ then, by definition, it is an induced substructure, and we obtain $\mathcal{A} \simeq \eta_p(\mathcal{A})$.

D.2.2 Proof of Lemma 8

Let us recall that

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

and, for all $(p, a) \in S_1^B$, $h(p, a) = \eta_p(a) = (pq, v)$ if p is an S -to- A path in $\text{dag}(D)$ and (q, v) is the A -representation of the valid \mathcal{B} -node $a \in \mathcal{E}(A)$.

We prove that h is a bijection from S_1^B to S_2^B (i.e., Lemma 8) by proving the following three lemmas.

► **Lemma 17.** *For every $(p, a) \in S_1^B$ we have $h(p, a) \in S_2^B$.*

Proof. Assume that p is an S -to- A path in $\text{dag}(D)$. Hence, a is a node of $\mathcal{E}(A)$ such that $\mathcal{N}_{\mathcal{E}(A), \rho}(a) \simeq \mathcal{B}$ and $\mathcal{N}_{\mathcal{E}(A), \rho}(a)$ is a valid substructure of $\mathcal{E}(A)$. To prove the lemma, we show that

$$\mathcal{N}_{\text{val}(D), \rho}(\eta_p(a)) = \eta_p(\mathcal{N}_{\mathcal{E}(A), \rho}(a)). \quad (6)$$

Since $\eta_p(\mathcal{N}_{\mathcal{E}(A), \rho}(a)) \simeq \mathcal{N}_{\mathcal{E}(A), \rho}(a) \simeq \mathcal{B}$ this yields $\mathcal{N}_{\text{val}(D), \rho}(h(p, a)) = \mathcal{N}_{\text{val}(D), \rho}(\eta_p(a)) \simeq \mathcal{B}$.

Let (q, v) be the A -representation of a . Thus the path q starts with A . The node $a = (q, v)$ is not of the form $(A, \tau_A(i))$ for some $i \in [\text{rank}(A)]$ (otherwise we would have $a \in \text{Bd}_{A, 2\rho+1}$ and $\mathcal{N}_{\mathcal{E}(A), \rho}(a)$ would not be valid). Therefore, we have $\eta_p(a) = (pq, v) \in \text{val}(D)$. Since both substructures in (6) are induced substructures of $\text{val}(D)$, it suffices to show that $\eta_p(\mathcal{S}_{\mathcal{E}(A), \rho}(a)) = \mathcal{S}_{\text{val}(D), \rho}(\eta_p(a))$. The inclusion $\eta_p(\mathcal{S}_{\mathcal{E}(A), \rho}(a)) \subseteq \mathcal{S}_{\text{val}(D), \rho}(\eta_p(a))$ holds since every $\mathcal{G}(\mathcal{E}(A))$ -path (i.e., a path in the undirected graph $\mathcal{G}(\mathcal{E}(A))$) is mapped by η_p to a corresponding $\mathcal{G}(\text{val}(D))$ -path of the same length.

Assume now that there is a node $b' \in \mathcal{S}_{\text{val}(D), \rho}(\eta_p(a)) \setminus \eta_p(\mathcal{S}_{\mathcal{E}(A), \rho}(a))$. We will deduce a contradiction. There is a $\mathcal{G}(\text{val}(D))$ -path of length at most ρ from $\eta_p(a)$ to b' . W.l.o.g. we can assume that all nodes along this path except for the final node b' belong to the set $\eta_p(\mathcal{S}_{\mathcal{E}(A), \rho}(a))$. In particular, there is a node $\eta_p(b) \in \text{val}(D)$ with $b \in \mathcal{S}_{\mathcal{E}(A), \rho}(a)$ such that there is a $\mathcal{G}(\text{val}(D))$ -path $\Pi_{\eta_p(a), \eta_p(b)}$ of length at most $\rho - 1$ from $\eta_p(a)$ to $\eta_p(b)$ and all nodes along this path belong to the set $\eta_p(\mathcal{S}_{\mathcal{E}(A), \rho}(a))$. Moreover, there is an edge in $\mathcal{G}(\text{val}(D))$ between $\eta_p(b)$ and b' . The $\mathcal{G}(\text{val}(D))$ -path $\Pi_{\eta_p(a), \eta_p(b)}$ yields a corresponding $\mathcal{G}(\mathcal{E}(A))$ -path $\Pi_{a, b}$ of the same length (hence, at most $\rho - 1$) from a to b .³ We therefore have $\text{dist}_{\mathcal{E}(A)}(a, b) \leq \rho - 1$.

We claim that $b' \in \eta_p(\mathcal{E}(A))$: We have $b \in \mathcal{N}_{\mathcal{E}(A), \rho}(a)$ and $\mathcal{N}_{\mathcal{E}(A), \rho}(a)$ is a valid substructure of $\mathcal{E}(A)$. Moreover, if \mathcal{C} is any valid substructure of $\mathcal{E}(A)$ then all neighbors of $\eta_p(\mathcal{C})$ in $\mathcal{G}(\text{val}(D))$ belong to $\eta_p(\mathcal{E}(A))$ (this is the crucial property of valid substructures). Since b' is a neighbor of $\eta_p(b)$ in $\mathcal{G}(\text{val}(D))$, we obtain $b' \in \eta_p(\mathcal{E}(A))$. Let $b' = \eta_p(c)$ with $c \in \mathcal{E}(A)$.

³ We use here the following general fact: If we have an edge (a, b) in the graph $\mathcal{G}(\mathcal{U})$ for a structure \mathcal{U} such that a and b belong to a subset V of \mathcal{U} , then (a, b) is also an edge in the graph $\mathcal{G}(\mathcal{V})$, where \mathcal{V} is the substructure induced by V . This statement is true since we restrict to relational structures where all relations have arity at most two, but it is wrong if we allow relations of arity 3 or larger and $\mathcal{G}(\mathcal{U})$ would be the Gaifman graph of a structure \mathcal{U} . Take for instance the structure \mathcal{U} with three elements a, b, c and the ternary relation $\{(a, b, c)\}$. In $\mathcal{G}(\mathcal{U})$ there is an edge between a and b , but there is no such edge in $\mathcal{G}(\mathcal{V})$, where \mathcal{V} is the substructure of \mathcal{U} induced by $\{a, b\}$.

Then (b, c) is an edge in $\mathcal{G}(\mathcal{E}(A))$ and hence $c \in \mathcal{S}_{\mathcal{E}(A), \rho}(a)$, i.e., $b' = \eta_p(c) \in \eta_p(\mathcal{S}_{\mathcal{E}(A), \rho}(a))$, which is a contradiction. \blacktriangleleft

► **Lemma 18.** $h : S_1^{\mathcal{B}} \rightarrow S_2^{\mathcal{B}}$ is surjective.

Proof. Let $b = (p', v)$ be a \mathcal{B} -node of $\text{val}(D)$. Let A be the unique nonterminal A such that $\mathcal{S}_{\text{val}(D), \rho}(b)$ is a top-level A -subset (see Lemma 16), and let p be the A -origin of $\mathcal{S}_{\text{val}(D), \rho}(b)$. We obtain a factorization $p' = pq$ where p ends in A and q starts in A . Let $a = (q, v) \in \text{val}(A)$, which is an A -representation. It remains to show that $(p, a) \in S_1^{\mathcal{B}}$.

Since $\mathcal{S}_{\text{val}(D), \rho}(b)$ is a top-level A -subset, we have $\eta_p(\mathcal{N}_{\text{val}(A), \rho}(a)) = \mathcal{N}_{\text{val}(D), \rho}(b) \simeq \mathcal{B}$ and hence $\mathcal{N}_{\text{val}(A), \rho}(a) \simeq \mathcal{B}$. We next show that $\mathcal{N}_{\text{val}(A), \rho}(a)$ is a valid substructure of $\mathcal{E}(A)$, which implies $\mathcal{N}_{\mathcal{E}(A), \rho}(a) = \mathcal{N}_{\text{val}(A), \rho}(a) \simeq \mathcal{B}$. This shows that a is a valid \mathcal{B} -node in $\mathcal{E}(A)$.

In order to show that $\mathcal{N}_{\text{val}(A), \rho}(a)$ is a valid substructure of $\mathcal{E}(A)$, we first observe that since $\mathcal{S}_{\text{val}(D), \rho}(b)$ is a top-level A -subset, we have

$$\mathcal{S}_{\text{val}(A), \rho}(a) \cap \{(A, v) : v \in \text{ran}(\tau_A)\} = \emptyset$$

and there is a node $a_0 \in \mathcal{S}_{\text{val}(A), \rho}(a) \cap \text{In}_A$. We have $\text{dist}_{\text{val}(A)}(a_0, a) \leq \rho$. Hence, for every node $c \in \mathcal{S}_{\text{val}(A), \rho}(a)$ we have $\text{dist}_{\text{val}(A)}(a_0, c) \leq 2\rho$. Therefore, $\mathcal{N}_{\text{val}(A), \rho}(a)$ is a substructure of $\mathcal{E}(A)$ and $\mathcal{N}_{\text{val}(A), \rho}(a) \cap \{c \in \text{val}(A) : \text{dist}_{\text{val}(A)}(\text{In}_A, c) = 2\rho + 1\} = \emptyset$. \blacktriangleleft

► **Lemma 19.** $h : S_1^{\mathcal{B}} \rightarrow S_2^{\mathcal{B}}$ is injective.

Proof. Let $(p_1, a_1), (p_2, a_2) \in S_1^{\mathcal{B}}$ such that $\eta_{p_1}(a_1) = h(p_1, a_1) = h(p_2, a_2) = \eta_{p_2}(a_2)$. We show that $(p_1, a_1) = (p_2, a_2)$. Assume that p_i is an S -to- A_i path in $\text{dag}(D)$ ($i \in \{1, 2\}$). Let (q_i, v_i) be the A_i -representation of a_i . Hence, we have $(p_1 q_1, v_1) = (p_2 q_2, v_2)$, i.e., $v_1 = v_2$ and $p_1 q_1 = p_2 q_2$. In order to show that $p_1 = p_2$ and $q_1 = q_2$ it suffices to show that $A_1 = A_2$ (then p_1 and p_2 end in the same nonterminal and therefore must be equal).

Recall that $\eta_{p_i}(\mathcal{S}_{\mathcal{E}(A_i), \rho}(a_i))$ is a top-level A_i -subset of $\text{val}(D)$. By (6) we have

$$\eta_{p_1}(\mathcal{S}_{\mathcal{E}(A_1), \rho}(a_1)) = \mathcal{S}_{\text{val}(D), \rho}(\eta_{p_1}(a_1)) = \mathcal{S}_{\text{val}(D), \rho}(\eta_{p_2}(a_2)) = \eta_{p_2}(\mathcal{S}_{\mathcal{E}(A_2), \rho}(a_2)).$$

Lemma 16 yields $A_1 = A_2$. \blacktriangleleft

D.3 Omitted details from Section 5.2

D.3.1 Computing all $(2\rho + 1)$ -expansions

The general idea to compute $\mathcal{E}(A)$ is to start a breadth-first search (BFS) in all nodes from In_A and thereby explore all nodes in $\text{val}(A)$ with distance of at most $2\rho + 1$ from any node in In_A . Obviously, this BFS has to be able to jump back and forth from one structure \mathcal{U}_B into another structure \mathcal{U}_C according to the references in the productions of D , which can be realized as follows.

The BFS visits triples (p, u, ℓ) , where (p, u) is a node of $\text{val}(A)$, i.e., p is an A -to- B path in $\text{dag}(D)$ for some $B \in N$ (this includes the case $B = A$, for which $p = A$) and $u \in \mathcal{U}_B$, where in addition $u \notin \text{ran}(\tau_B)$ in case $B \neq A$, and ℓ with $0 \leq \ell \leq 2\rho + 1$ indicates that (p, u) has distance ℓ from In_A . Initially, we visit all $(A, u, 0)$, where u is an internal node of \mathcal{U}_A . From the current triple (p, u, ℓ) , where p ends in $B \in N$ and $\ell \leq 2\rho$, we can visit unvisited triples of the following form:

- Stay1: $(p, v, \ell + 1)$ if $v \in \mathcal{U}_B$ is adjacent to u in the graph $\mathcal{G}(\mathcal{U}_B)$, where in addition $v \notin \text{ran}(\tau_B)$ in case $B \neq A$ (we stay in the structure \mathcal{U}_B and move via an edge from \mathcal{U}_B),

- Stay2: $(p, v, \ell + 1)$ if $v \in \mathcal{U}_B$ and there is a reference $(C, \sigma) \in E_B$ with $u = \sigma(i)$, $v = \sigma(j)$ (for $i, j \in [\text{rank}(C)]$) and in \mathcal{U}_C there is an edge in $\mathcal{G}(\mathcal{U}_C)$ from $\tau_C(i)$ to $\tau_C(j)$ (we stay in the structure \mathcal{U}_B but move via an edge that is produced from a reference),
- Move down: $(p, j, C, v, \ell + 1)$ if the j^{th} reference in E_B is (C, σ) , there is an $i \in [\text{rank}(C)]$ such that $\sigma(i) = u$ and $v \in \mathcal{U}_C \setminus \text{ran}(\tau_C)$ is adjacent to $\tau_C(i)$ in $\mathcal{G}(\mathcal{U}_C)$ (we move into a structure \mathcal{U}_C that is produced from a reference).
- Move up: $(p', C, \sigma(i), \ell + 1)$ if $p = p' C j B$, the j^{th} reference in E_C is (B, σ) , and in $\mathcal{G}(\mathcal{U}_B)$, u is adjacent to $\tau_B(i)$ for $i \in [\text{rank}(B)]$; note that $\sigma(i) \notin \text{ran}(\tau_C)$ by the apex condition (via a contact node of \mathcal{U}_B we move up to the structure from which the current copy of \mathcal{U}_B was produced).

This BFS visits exactly those triples (p, u, ℓ) such that (p, u) is a node from $\text{val}(A)$ with distance $\ell \leq 2\rho + 1$ from In_A . Consequently, we visit exactly the nodes of $\mathcal{E}(A)$. The nodes in $\text{Bd}_{A, 2\rho+1}$ can be easily detected in the BFS. Note also that for every computed triple (p, v, ℓ) the path p has length at most $\ell \leq 2\rho + 1$.

This construction requires time $\mathcal{O}(|\mathcal{E}(A)|) \leq |\mathcal{U}_A| \cdot f(d, |\phi|)$ (the size of a $(2\rho + 1)$ -sphere around a tuple of length at most $|\mathcal{U}_A|$ in a degree- d bounded structure). Summing over all $A \in N$ shows that all $(2\rho + 1)$ -expansions can be computed in time $|D| \cdot f(d, |\phi|)$.

D.3.2 Computing the ρ -neighborhood types and useful nonterminals

For every nonterminal A , every node $a \in \mathcal{E}(A)$, and every ρ -neighborhood type \mathcal{B}_i from (3) we check whether a is a valid \mathcal{B}_i -node. For this, we have to compute $\mathcal{N}_{\mathcal{E}(A), \rho}(a)$, which can be done in time $f(d, |\phi|)$. If this is the case, we store a in a list $\mathcal{L}_{i,A}$, and we also compute and store an isomorphism $\pi_a : \mathcal{B} \rightarrow \mathcal{N}_{\mathcal{E}(A), \rho}(a)$ that satisfies $\pi_a(1) = a$.

Since the size of every expansion $\mathcal{E}(A)$ can be bounded by $|\mathcal{U}_A| \cdot f(d, |\phi|)$, the total time needed for the above computations is bounded by $|D| \cdot f(d, |\phi|)$.

Recall that a nonterminal A is \mathcal{B}_i -useful if and only if there is a valid \mathcal{B}_i -node in $\mathcal{E}(A)$. Hence, A is \mathcal{B}_i -useful if and only if the list $\mathcal{L}_{i,A}$ is non-empty. Hence, we have also computed the set of \mathcal{B}_i -useful nonterminals.

D.3.3 Preprocessing for path enumeration

The following preprocessing is needed for our path enumeration algorithm in $\text{dag}(D) = (N, \gamma, S)$. Recall that we denote with \mathcal{P}_i the set of all initial paths in $\text{dag}(D)$ that end in a \mathcal{B}_i -useful nonterminal (see Section 5.2), where \mathcal{B}_i is from the factorization (3).

We first extend $\text{dag}(D)$ to the new dag $\text{dag}(D)' = (N \uplus N', \gamma', S)$, where $N' = \{A' : A \in N\}$ is a copy of N , $\gamma'(A) = A' \gamma(A)$ and $\gamma'(A') = \varepsilon$ for every $A \in N$. In other words, we add for every nonterminal A a new node A' (a leaf in $\text{dag}(D)'$) that becomes the first child of A . Recall that if $\gamma(A) = B_1 \cdots B_n$ then we write $(A, 1, B_1), \dots, (A, n, B_n)$ for the outgoing edges of A in $\text{dag}(D)$. It is convenient to keep these triples also in $\text{dag}(D)'$ and to write $(A, 0, A')$ for the new edge from A to A' . Clearly, $\text{dag}(D)'$ can be constructed in time $\mathcal{O}(|D|)$ from D .

There is a one-to-one correspondence between initial paths in $\text{dag}(D)$ and initial-to-leaf paths in $\text{dag}(D)'$. Formally, if p is an initial-to- A path in $\text{dag}(D)$, then $p' := p 0 A'$ is an initial-to-leaf path in $\text{dag}(D)'$. Moreover, every initial-to-leaf path in $\text{dag}(D)'$ is of the form $p 0 A'$ for a unique initial-to- A path p in $\text{dag}(D)$. This means that in the following, we can talk about initial-to-leaf paths of $\text{dag}(D)'$ instead of initial paths of $\text{dag}(D)$. Note that in constant time we can always obtain the initial path from $\text{dag}(D)$ that corresponds to some initial-to-leaf path of $\text{dag}(D)'$.

Recall the lexicographical ordering of paths with the same starting node from Section 2.1. For a path p in $\text{dag}(D)$ that starts in A , we define $\text{lex}_A(p)$ as the position of p in the lexicographically ordered list of all paths in $\text{dag}(D)$ that start in A , where $\text{lex}_A(A) = 0$ (note that A is the lexicographically smallest path that starts in A). We write $\text{lex}(p)$ for $\text{lex}_S(p)$. Note that for an S -to- A path p and an A -to- B path q we have $\text{lex}(pq) = \text{lex}(p) + \text{lex}_A(q)$.

For an initial-to-leaf path p' in $\text{dag}(D)$ we write $\text{lex}'(p')$ for the position of p' in the lexicographically ordered list of all initial-to-leaf paths in $\text{dag}(D)'$, where we start again with position 0. For every initial-to- A path p in $\text{dag}(D)$ we then have $\text{lex}(p) = \text{lex}'(p0A')$. Note that lex' always refers to the dag $\text{dag}(D)'$, whereas lex always refers to $\text{dag}(D)$.

Next, we add edge-weights to $\text{dag}(D)'$ as follows in order to compute $\text{lex}'(p')$ for an initial-to-leaf path p' in $\text{dag}(D)'$. For every node A of $\text{dag}(D)'$, let $\#\text{paths}(A)$ be the number of different A -to-leaf paths in $\text{dag}(D)'$. These values can be easily computed in time $\mathcal{O}(|\text{dag}(D)'|) \leq \mathcal{O}(|D|)$. Indeed, for every leaf A' , we set $\#\text{paths}(A) = 1$. For an inner node A with $\gamma'(A) = A_0A_1 \cdots A_m$ (where $A_0 = A'$), we set $\#\text{paths}(A) = \sum_{i=0}^m \#\text{paths}(A_i)$.

Next, we compute the number $\text{weight}(e)$ for every edge e of $\text{dag}(D)'$. Let A be an arbitrary inner node with $\gamma'(A) = A_0A_1 \cdots A_m$ (where $A_0 = A'$). Then, for every $0 \leq i \leq m$, we set $\text{weight}(A, i, A_i) = \sum_{j=0}^{i-1} \#\text{paths}(A_j)$. Obviously, these numbers $\text{weight}(e)$ for every edge e can be also computed in time $\mathcal{O}(|\text{dag}(D)'|) \leq \mathcal{O}(|D|)$.

By definition, $\#\text{paths}(A)$ is the total number of A -to-leaf paths in $\text{dag}(D)'$. Thus, for every $0 \leq i \leq m$, there are exactly $\text{weight}(A, i, A_i)$ many A -to-leaf paths that are lexicographically smaller than the smallest A -to-leaf path that starts with the edge (A, i, A_i) .

We extend the function weight to paths in $\text{dag}(D)'$ in the obvious way: for a path $p' = A_0j_1A_1 \cdots A_{n-1}j_nA_n$, we set $\text{weight}(p') = \sum_{i=1}^n \text{weight}(A_{i-1}, j_i, A_i)$. It can be verified by induction that for every initial-to-leaf path p' in $\text{dag}(D)'$ we have $\text{weight}(p') = \text{lex}'(p')$. Hence, for $p' = p0A'$ we obtain

$$\text{weight}(p0A') = \text{lex}'(p0A') = \text{lex}(p). \quad (7)$$

Next, for every ρ -neighborhood type \mathcal{B}_i from the factorization (3), we create a dag dag_i from $\text{dag}(D)'$ as follows. Initially, let dag_i be just a copy of $\text{dag}(D)'$. In a first step, we delete all nodes in dag_i from which no leaf A' (with $A \in N$) is reachable such that A is \mathcal{B}_i -useful in the SLP D . We can assume that the initial nonterminal S is not removed here, otherwise D contains no \mathcal{B}_i -useful nonterminals and the factorization (3) will not lead to any output tuples. We normalize the outgoing edges (A, i, B) for every node A such that the middle indices i form an interval $[n]$ for some n .

The paths from \mathcal{P}_i correspond exactly to the initial-to-leaf paths in dag_i . Note that we only delete nodes and edges, but do not change edge-weights when construct dag_i from $\text{dag}(D)'$. This means that if p is an initial-to-leaf path in dag_i then p has the same weight in $\text{dag}(D)'$ and dag_i .

For an arbitrary DAG $G = (V, \gamma, \iota)$ and a node $v \in V$ we define the *outdegree* of v as $|\gamma(v)|$.

A *maximal non-branching* path in dag_i is a path $p = A j B_1 1 B_2 1 \cdots B_{n-1} 1 B_n$ where $n \geq 1$, A has outdegree at least 2, every B_i for $i \in [n-1]$ has outdegree 1 (so B_{i+1} is the unique child of B_i), and B_n has outdegree 0 or at least 2. For technical reasons that shall become clear later on, we want to contract each such maximal non-branching path p into a single edge (A, j, B_n) with $\text{weight}(A, j, B_n) = \text{weight}(p)$. This can be done as follows.

We first determine the set V_1 of nodes of outdegree one. Then, for every node $A \in V_1$, we compute the unique pair $(\omega(A), g(A))$, where $\omega(A)$ is the unique node with $\omega(A) \notin V_1$ that is reached from A by the unique path p_A consisting of edges $(A_1, 1, A_2)$ with $A_1 \in V_1$, and

$g(A) = \text{weight}(p_A)$. This can be done bottom-up in time $\mathcal{O}(|\text{dag}_i|)$ as follows: For every edge $(A, 1, B)$ with $A \in V_1$ we set $\omega(A) = B$ and $g(A) = \text{weight}(A, 1, B)$ if $B \notin V_1$ (this includes the case where B is a leaf), and we set $\omega(A) = \omega(B)$ and $g(A) = \text{weight}(A, 1, B) + g(B)$ if $B \in V_1$ has already been processed. We then replace every edge (B, i, A) with $A \in V_1$ by the edge $(B, i, \omega(A))$ with $\text{weight}(B, i, \omega(A)) = \text{weight}(B, i, A) + g(A)$. After this step, there is no edge that ends in a node of outdegree one. In particular, if a node has outdegree one, then it has no incoming edges. We then remove all nodes of outdegree one and their outgoing edges from dag_i , except the initial node S (which might have outdegree 1 as well). Now all nodes except possibly S have outdegree 0 or at least 2. If the initial node S has outdegree at least 2, we are done. If S has outdegree 1, then, by our construction, it has an edge $(S, 1, A)$ and A has outdegree 0 or at least 2. In this case, we delete S and make A the new initial node, and we store the information that every initial-to-leaf path of dag_i has to be interpreted as the initial-to-leaf path obtained by prepending the edge $(S, 1, A)$.

Recall that the paths from \mathcal{P}_i in $\text{dag}(D)$ are in a one-to-one correspondence with the initial-to-leaf paths of dag_i . The only difference is that we added an edge (A, i, A') at the end and that the maximal non-branching subpaths of the initial paths of $\text{dag}(D)'$ are contracted into single edges in dag_i . Moreover, the weight of an initial-to-leaf path in dag_i is the same as the weight of the corresponding path from \mathcal{P}_i , which is its lex-value.

Every dag_i can be computed in time $\mathcal{O}(|\text{dag}(D)'|) \leq \mathcal{O}(|D|)$ and we have to construct $m \leq k \leq |\phi|$ many of them. Hence, the total running time is bounded by $|D| \cdot f(d, |\phi|)$.

D.3.4 Enumerating paths from \mathcal{P}_i in $\text{dag}(D)$

The general idea is that we enumerate with constant delay the initial-to-leaf paths in dag_i in lexicographic order. As explained in Section D.3.3, these paths correspond to the paths from \mathcal{P}_i in $\text{dag}(D)$. In order to be able to go in constant time from an initial-to-leaf path of dag_i to the lexicographical next initial-to-leaf path, we use an idea from [46] (which in turn is based on [26]). In [46] only binary dags are considered, i.e., dags where every non-leaf node has a left and a right outgoing edge. Then the idea is to represent an initial-to-leaf path in a compact form where every maximal subpath p (going from u to v) that consist only of left (right, respectively) edges is contracted into a single triple (u, ℓ, v) ((u, r, v) , respectively).

In our dags dag_i , every non-leaf node can have an arbitrary outdegree larger than one. We therefore have to adapt the approach from [46] to handle also nodes with more than two outgoing edges. The idea will be to contract only maximal subpaths consisting of leftmost (rightmost, respectively) edges into single triples.

We describe the procedure for a general dag $G = (V, \gamma, \iota)$ where, for every $v \in V$, we have $|\gamma(v)| = 0$ or $|\gamma(v)| \geq 2$, and every edge (u, j, v) has an edge-weight $\text{weight}(u, j, v) \in \mathbb{N}$ (note that the dags dag_i have this property). This also means that paths of G have weights. We call every edge $(u, 1, v)$ a *min-edge*, every edge $(u, |\gamma(u)|, v)$ a *max-edge*, and edges that are neither min- nor max-edges are called *middle-edges*. This means that every node is either a leaf with no outgoing edges, or it is an inner node with one min- and one max-edge and a (possibly zero) number of middle edges.

Min-max-contracted representations. In this section, it will be convenient to write paths in G as sequences of edge triples (u, i, v) instead of words from $(V\mathbb{N})^*V$. Thus the path $v_0j_1v_1j_2v_2 \cdots j_nv_n$ will be written $(v_0, j_1, v_1)(v_1, j_2, v_2) \cdots (v_{n-1}, j_n, v_n)$. With this notation, the concatenation of paths corresponds to the concatenation of sequences of edges.

Let $p = (v_0, j_1, v_1) \cdots (v_{n-1}, j_n, v_n)$ be a path in G . The *min-max-contracted representation* of p is obtained by contracting every maximal sequence of min-edges (max-edges,

respectively) into a single triple. Here, a sequence of min-edges (max-edges, respectively) is *maximal* if it cannot be extended to the left or right by min-edges (max-edges, respectively). More formally, the min-max-contracted representation of p is obtained by replacing each non-empty sequence $(v_i, 1, v_{i+1})(v_{i+1}, 1, v_{i+2}) \cdots (v_{i'-1}, 1, v_{i'})$ of min-edges that is neither preceded nor followed by a min-edge with the single triple $(v_i, \min, v_{i'})$, and, analogously, replacing each non-empty sequence $(v_i, |\gamma(v_i)|, v_{i+1}) \cdots (v_{i'-1}, |\gamma(v_{i'-1})|, v_{i'})$ of max-edges that is neither preceded nor followed by a max-edge with the single triple $(v_i, \max, v_{i'})$. Intuitively speaking, a triple $(v_i, \min, v_{i'})$ ($(v_i, \max, v_{i'})$, respectively) means that we go from v_i to $v_{i'}$ by only taking the first (last, respectively) outgoing edge for every node. For example, the path

$$(v_0, 1, v_1)(v_1, 1, v_2)(v_2, 7, v_3)(v_3, 4, v_4)(v_4, 1, v_5)(v_5, |\gamma(v_5)|, v_6)(v_6, |\gamma(v_6)|, v_7)(v_7, 5, v_8)$$

(where $|\gamma(v_2)| > 7$, $|\gamma(v_3)| > 4$, and $|\gamma(v_7)| > 5$) has the min-max-contracted representation

$$(v_0, \min, v_2)(v_2, 7, v_3)(v_3, 4, v_4)(v_4, \min, v_5)(v_5, \max, v_7)(v_7, 5, v_8).$$

For every node v_0 , there is a one-to-one correspondence between v_0 -paths and the min-max-contracted representations of v_0 -paths.

We extend **weight** to triples (u, \min, v) and (u, \max, v) , i.e., $\text{weight}(u, \min, v) = \text{weight}(p)$, where p is the unique min-path from u to v , and $\text{weight}(u, \max, v) = \text{weight}(p)$, where p is the unique max-path from u to v . If \tilde{p} is the min-max-contracted representation of the path p then we define $\text{weight}(\tilde{p}) = \text{weight}(p)$. Moreover, we call the values $\text{weight}(e)$ for all triples e in \tilde{p} the **weight-values** of \tilde{p} . Hence, $\text{weight}(\tilde{p})$ is the sum of all the **weight-values** of \tilde{p} .

Data structures. We describe next certain data structures for G that need to be computed in order to enumerate all initial-to-leaf paths. We will see that all these data structures can be computed in time $\mathcal{O}(|G|)$, which means that we can compute them for all dags dag_i in the preprocessing phase of our final enumeration algorithm in time $|D| \cdot f(d, |\phi|)$.

The *min-path* of a node v is the v -to-leaf path obtained by starting in v and only taking min-edges until we reach a leaf, which we denote by $\min(v)$. In particular, $\min(v) = v$ if and only if v is a leaf. The *max-path* of v and the node $\max(v)$ are defined analogously. We can compute the value $\min(v)$ and the corresponding weight $\text{weight}(v, \min, \min(v))$ as follows ($\max(v)$ and $\text{weight}(v, \max, \max(v))$ are computed analogously): For every leaf v , we set $\min(v) = v$ and $\text{weight}(v, \min, \min(v)) = 0$ (actually, the triple $\text{weight}(v, \min, v)$ does not appear in a min-max-contracted representation, it is used here only in order to facilitate the computation of the weights). For every inner node v with min-edge $(v, 1, u)$, we set $\min(v) = \min(u)$ and $\text{weight}(v, \min, \min(v)) = \text{weight}(v, 1, u) + \text{weight}(u, \min, \min(u))$.

For a node u on v 's min-path with $v \neq u$, we denote by $\wp_{\min}(v, u)$ the unique parent node of u on v 's min-path. Observe that if u is the direct successor of v on v 's min-path, then $\wp_{\min}(v, u) = v$. For a node u on v 's max-path with $v \neq u$, we define $\wp_{\max}(v, u)$ analogously.

We will next discuss how to compute in time $\mathcal{O}(|G|)$ a data structure that allows us to retrieve in constant time the value $\wp_{\min}(v, u)$ for a node u on v 's min-path and the value $\wp_{\max}(v, u)$ for a node u on v 's max-path.

Let us only discuss the min-case, since the max-case can be dealt with analogously. We first remove from G all edges that are not min-edges and then we reverse all edge directions. This gives us a forest F_{\min} . For every node v of G , there is a unique rooted tree $T(v)$ in F_{\min} that contains v , and v is the root of $T(v)$ if and only if v is a leaf in G . Moreover, the path from v to the root of $T(v)$ is exactly the min-path of v in G . This forest F_{\min} can be computed in time $\mathcal{O}(|G|)$.

For an arbitrary rooted tree T and nodes $u, v \in T$ such that there is a non-empty path from u to v , we denote by $\text{nl}_T(u, v)$ the unique child of u that belongs to the unique u -to- v path in T . A *next link data structure* for T is a data structure that allows us to retrieve $\text{nl}_T(u, v)$ for given $u, v \in T$ as above. We call these queries *next link queries*. The following result from [26] will be used:

► **Theorem 20.** *For a rooted tree T , one can construct in time $\mathcal{O}(|T|)$ a next link data structure for T that allows to solve next link queries in constant time.*

For every $v \in V$ and $u \neq v$ on v 's min-path we have that $T(v) = T(u)$, and the node $\wp_{\min}(v, u)$ is the unique child of u on the u -to- v path in $T(v)$, i.e., $\wp_{\min}(v, u) = \text{nl}_{T(v)}(u, v)$. Consequently, by computing the next link data structure from Theorem 20 for every tree of the forest F_{\min} , we can retrieve $\wp_{\min}(v, u)$ for given $v \in V$ and $u \neq v$ on v 's min-path in constant time.

Enumeration procedure. Provided we have computed the data structures mentioned above, we can support the following operation for G . Given the min-max-contracted representation of an initial-to-leaf path p , all its **weight**-values and its total weight, we can construct in constant time the min-max-contracted representation of the lexicographical successor of p (among all initial-to-leaf paths in G) along with all its **weight**-values and its total weight. If this lexicographical successor does not exist then the algorithm will report that p is the lexicographically largest path.

Let p be the min-max-contracted representation of an initial-to-leaf path. We first show how to obtain the min-max-contracted representation of the lexicographically next initial-to-leaf path in constant time. To this end, we consider three different cases depending on whether p 's last triple is a min-triple, a middle-edge or a max-triple. After that, we discuss how to update the **weight**-values and the total weight. Note that in each of the following cases, p is only modified in a suffix of constant length. Hence, constant time suffices in each case. In order to avoid some further case distinctions we interpret a triple (v, \min, v) by the empty sequence ε . Recall that $\min(v) = v$ if and only if v is a leaf.

Case 1: p ends with a min-triple. Let $p = p'(v_1, \min, v_2)$. This means that $v_3 := \wp_{\min}(v_1, v_2)$ is defined and there is an edge $(v_3, 2, v_4)$.

Case 1.1: $v_1 = v_3$. If $(v_1, 2, v_4)$ is a middle-edge (i.e., $|\gamma(v_1)| > 2$) then we set $p := p'(v_1, 2, v_4)(v_4, \min, \min(v_4))$.

If on the other hand $(v_1, 2, v_4)$ is a max-edge then we have to check whether the last triple of p' (if it exists) is a max-triple or not. If not, then we set $p := p'(v_1, \max, v_4)(v_4, \min, \min(v_4))$. On the other hand, if p' is of the form $p' = p''(v_5, \max, v_1)$ (note that p' ends with v_1) then we set $p := p''(v_5, \max, v_4)(v_4, \min, \min(v_4))$.

Case 1.2: $v_1 \neq v_3$. If $(v_3, 2, v_4)$ is a middle-edge (i.e., $|\gamma(v_3)| > 2$) then we set $p := p'(v_1, \min, v_3)(v_3, 2, v_4)(v_4, \min, \min(v_4))$. If $(v_3, 2, v_4)$ is a max-edge, then we set $p := p'(v_1, \min, v_3)(v_3, \max, v_4)(v_4, \min, \min(v_4))$.

Case 2: p ends with middle-edge. Let $p = p'(v_1, j, v_2)$. Then there exists a middle- or max-edge $(v_1, j+1, v_3)$ in G . If $(v_1, j+1, v_3)$ is a middle edge in G , then we set $p := p'(v_1, j+1, v_3)(v_3, \min, \min(v_3))$.

Now assume that $(v_1, j+1, v_3)$ is a max-edge. If p' does not end with a max-triple (this includes the case that p' is empty) then we set $p := p'(v_1, \max, v_3)(v_3, \min, \min(v_3))$. Finally, if $p' = p''(v_4, \max, v_1)$ ends with a max-triple then we set $p := p''(v_4, \max, v_3)(v_3, \min, \min(v_3))$.

Case 3: p ends with a max-triple. If $p = (\iota, \max, v_1)$, then the algorithm reports that p is the lexicographically largest path. Assume now that $p = p'(v_1, j, v_2)(v_2, \max, v_3)$, where (v_1, j, v_2) is a middle-edge or a min-triple (in a min-max-contracted representation there do not exist two consecutive max-triples). We then set $p := p'(v_1, j, v_2)$ and continue with either Case 1 (if (v_1, j, v_2) is a min-triple) or Case 2 (if (v_1, j, v_2) is a middle-edge).

For obtaining the **weight**-values of the new min-max-contracted representation, we only have to compute the **weight**-values for the new triples (for the old triples the **weight**-values remain the same). If a new triple (u, j, v) is a middle edge, then the value **weight** (u, j, v) has been explicitly computed in the preprocessing. For triples of the form (u, \min, v) , we can argue as follows: Whenever we add a new min-triple (u, \min, v) in the above algorithm, then one of the following three cases holds:

- The min-edge $(u, 1, v)$ exists in G : We then set **weight** $(u, \min, v) = \text{weight}(u, 1, v)$, where the latter has been precomputed.
- $v = \min(u)$: Then **weight** $(u, \min, \min(u))$ has been precomputed.
- There is an old min-triple (u, \min, v') such that $v = \wp_{\min}(u, v')$. In this case we set **weight** $(u, \min, v) = \text{weight}(u, \min, v') - \text{weight}(v, 1, v')$. Here $(v, 1, v')$ is an edge of G for which the weight is known.

For new max-triples we can compute the weights in a similar way.

Now with all the **weight**-values of the new p at our disposal, we can easily compute the total weight **weight** (p) for the new p in constant time. Indeed, notice that in the above algorithm, p will be modified by removing a constant number of triples from the end and then adding a constant number of triples at the end. Hence, **weight** (p) can be updated by a constant number of subtractions and additions.

When we speak about the min-max-contracted representation of a path in the following, we assume that all **weight**-values as well as the total weight of the path are also computed.

Enumerating the initial-to-leaf paths of dag_i . In order to enumerate the initial-to-leaf paths of dag_i , we start with the min-max-contracted representation of the first initial-to-leaf path of dag_i , which can be obtained in constant time, since it is simply $(S, \min, \min(S))$. Then, by the procedure described above, we can obtain the min-max-contracted representation of the lexicographically next initial-to-leaf path from the min-max-contracted representation of the previous initial-to-leaf path in constant time, which gives us a constant delay enumeration algorithm for the min-max-contracted representations of all such paths. Thereby, also the **weight**-values and the total weight are correctly updated.

D.3.5 Counting the number of \mathcal{B}_i -nodes

For every ρ -neighborhood \mathcal{B}_i , we need the total number of \mathcal{B}_i -nodes in $\text{val}(D)$. In the uncompressed setting, this number is given by the size of the explicitly computed list L_i (see the algorithm described in Section 3.1). In the compressed setting, it can be computed (thanks to Lemma 8) as

$$\beta_i = \sum_{A \in N_i} P_A \cdot |\mathcal{L}_{i,A}|,$$

where N_i is the set of all nonterminals that are \mathcal{B}_i -useful, P_A is the number of S -to- A paths in $\text{dag}(D)$, and $\mathcal{L}_{i,A}$ is the list of all valid \mathcal{B}_i -nodes in $\mathcal{E}(A)$. Recall that in Section D.3.2, we have already computed the sets N_i and the lists $\mathcal{L}_{i,A}$.

The numbers P_A can be computed as follows. We first remove from $\text{dag}(D)$ all nonterminals A such that there is no S -to- A path in $\text{dag}(D)$ (such nonterminals could have been removed in the very beginning without changing $\text{val}(D)$). This can be done in time $\mathcal{O}(|\text{dag}(D)|) \leq \mathcal{O}(|D|)$. Then we can compute all numbers P_A inductively top-down in $\text{dag}(D)$ by first setting $P_S = 1$ and then setting $P_A = \sum_{1 \leq i \leq \ell} P_{A_i}$ for every node A with incoming edges $(A_1, i_1, A), \dots, (A_\ell, i_\ell, A)$.

The total number of (binary) additions is bounded by the number of edges of $\text{dag}(D)$, which is bounded by $|D|$. Moreover, all numbers P_A need only $\mathcal{O}(|D|)$ many bits (see Section 4.1). Therefore, on our machine model, every addition needs constant time (see Section 4.2). Hence, the whole procedure needs time $\mathcal{O}(|\text{dag}(D)|)$. Therefore, all numbers β_i can be computed in time $f(d, |\phi|) \cdot |D|$.

D.3.6 Checking distance constraints

Recall that we fixed the consistent factorization $(\mathcal{B}_1, \sigma_1, \dots, \mathcal{B}_m, \sigma_m)$ (see (3)) of the fixed (k, r) -neighborhood type \mathcal{B} at the beginning of Section 5.2. We also noticed that during the enumeration phase we have to check whether $\text{dist}_{\text{val}(D)}(t_{b_i, \sigma_i}, t_{b_j, \sigma_j}) > 2r + 1$ for a \mathcal{B}_i -node b_i and a \mathcal{B}_j -node b_j . Here, the following assumptions hold for i (and analogously for j):

- 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) that is given by the min-max-contracted representation of the corresponding path in dag_i .
- $c_i := (q_i, v_i)$ is a node (written in A_i -representation) from $\mathcal{E}(A_i)$ such that c_i has the 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 $\mathcal{E}(A_i)$.

Moreover, let $\Gamma_i = \{(A_i, v) : v \in \text{ran}(\tau_{A_i})\}$ be the set of contact nodes of $\text{val}(A_i)$. Since $(\mathcal{B}_1, \sigma_1, \dots, \mathcal{B}_m, \sigma_m)$ is a consistent factorization of the (k, r) -neighborhood type \mathcal{B} and $\mathcal{N}_{\mathcal{E}(A_i), \rho}(c_i) \simeq \mathcal{B}_i$ we have for every $c \in \text{ran}(t_{c_i, \sigma_i})$: $\mathcal{S}_{\mathcal{E}(A_i), r}(c) \subseteq \mathcal{S}_{\mathcal{E}(A_i), r}(t_{c_i, \sigma_i}) \subseteq \mathcal{S}_{\mathcal{E}(A_i), \rho}(c_i)$. Since $\mathcal{N}_{\mathcal{E}(A_i), \rho}(c_i)$ is a valid substructure of $\mathcal{E}(A_i)$, we obtain $\mathcal{S}_{\text{val}(A_i), r}(c) \cap \Gamma_i = \emptyset$, i.e.,

$$\text{dist}_{\text{val}(A_i)}(c, \Gamma_i) \geq r + 1 \quad (8)$$

for all $i \in [m]$ and all $c \in \text{ran}(t_{c_i, \sigma_i})$. We next state some sufficient conditions for $\text{dist}_{\text{val}(D)}(t_{b_i, \sigma_i}, t_{b_j, \sigma_j}) > 2r + 1$.

► **Lemma 21.** *If neither p_i is a prefix of p_j nor the other way around, then we have $\text{dist}_{\text{val}(D)}(t_{b_i, \sigma_i}, t_{b_j, \sigma_j}) > 2r + 1$.*

Proof. We have to show that if neither p_i is a prefix of p_j nor the other way around, then $\text{dist}_{\text{val}(D)}(b, b') > 2r + 1$ for every $b \in \text{ran}(t_{b_i, \sigma_i})$ and $b' \in \text{ran}(t_{b_j, \sigma_j})$. To this end, let $b \in \text{ran}(t_{b_i, \sigma_i})$ and $b' \in \text{ran}(t_{b_j, \sigma_j})$ be arbitrarily chosen. If b and b' are not connected in $\mathcal{G}(\text{val}(D))$, then $\text{dist}_{\text{val}(D)}(b, b') = \infty > 2r + 1$. So, let us assume that b and b' are connected. Let $c \in \text{ran}(t_{c_i, \sigma_i})$ and $c' \in \text{ran}(t_{c_j, \sigma_j})$ such that $\eta_{p_i}(c) = b$ and $\eta_{p_j}(c') = b'$.

If neither p_i is a prefix of p_j nor the other way around, then the intersection of the (universes of the) substructures $\eta_{p_i}(\text{val}(A_i))$ and $\eta_{p_j}(\text{val}(A_j))$ is $\eta_{p_i}(\Gamma_i) \cap \eta_{p_j}(\Gamma_j)$, which can be non-empty. Consequently, any path Π from b to b' has a shortest prefix Π_1 that goes from $b = \eta_{p_i}(c)$ to $\eta_{p_i}(\Gamma_i)$ as well as a shortest suffix Π_2 that goes from $\eta_{p_j}(\Gamma_j)$ to $b' = \eta_{p_j}(c')$. The path Π_1 must be the η_{p_i} -image of a path from c to Γ_i in $\text{val}(A_i)$ and the path Π_2 must be the η_{p_j} -image of a path from Γ_j to c' in $\text{val}(A_j)$. From (8) it follows that Π_1 and Π_2 have both length at least $r + 1$. Hence, every path from b to b' has length at least $2(r + 1) = 2r + 2$, i.e., $\text{dist}_{\text{val}(D)}(b, b') > 2r + 1$. This shows the lemma. ◀

► **Lemma 22.** *Assume that $p_j = p_i q$ for some path q . Let $b \in \text{ran}(t_{b_i, \sigma_i})$ and $b' \in \text{ran}(t_{b_j, \sigma_j})$ and take the unique $c \in \text{ran}(t_{c_i, \sigma_i})$ and $c' \in \text{ran}(t_{c_j, \sigma_j})$ such that $\eta_{p_i}(c) = b$ and $\eta_{p_j}(c') = \eta_{p_i}(\eta_q(c')) = b'$. If $\text{dist}_{\text{val}(D)}(b, b') \leq 2r + 1$ then $\text{dist}_{\text{val}(A_i)}(c, \eta_q(c')) \leq 2r + 1$.*

Proof. Since $p_j = p_i q$, we know that η_q is an embedding of $\text{val}(A_j)$ into $\text{val}(A_i)$.

There is a path Π between b and b' of length at most $2r + 1$ in the graph $\mathcal{G}(\text{val}(D))$. We first show that Π does not contain a node from $\eta_{p_i}(\Gamma_i)$. If this would not be the case then we can take the shortest prefix Π_1 of Π that goes from $b = \eta_{p_i}(c)$ to $\eta_{p_i}(\Gamma_i)$ and the shortest suffix Π_2 of Π that goes from $\eta_{p_j}(\Gamma_j) = \eta_{p_i}(\eta_q(\Gamma_j))$ to $b' = \eta_{p_j}(c') = \eta_{p_i}(\eta_q(c'))$. The path Π_1 is contained in $\eta_{p_i}(\text{val}(A_i))$ and visits a node from $\eta_{p_i}(\Gamma_i)$ only at the very end. Similarly, the path Π_2 is contained in $\eta_{p_j}(\text{val}(A_j))$ and visits a node from $\eta_{p_j}(\Gamma_j)$ only in the beginning. By (8) we have $\text{dist}_{\text{val}(A_i)}(c, \Gamma_i) \geq r + 1$ and $\text{dist}_{\text{val}(A_j)}(c', \Gamma_j) \geq r + 1$, which implies that Π_1 and Π_2 have both length at least $r + 1$. Hence, Π has length at least $2r + 2$, which is a contradiction.

Therefore, Π does not visit a node from $\eta_{p_i}(\Gamma_i)$, which implies that Π is a path in $\eta_{p_i}(\text{val}(A_i))$ and does not contain edges between nodes from $\eta_{p_i}(\Gamma_i)$. Hence, Π arises from a path between c and $\eta_q(c')$ in $\text{val}(A_i)$, which finally yields $\text{dist}_{\text{val}(A_i)}(c, \eta_q(c')) \leq 2r + 1$. ◀

► **Lemma 23.** *If $p_j = p_i q$ or $p_i = p_j q$, and the path q has length at least $3\rho - r + 1$, then $\text{dist}_{\text{val}(D)}(t_{b_i, \sigma_i}, t_{b_j, \sigma_j}) > 2r + 1$.*

Proof. Assume that $p_j = p_i q$ for some path q of length at least $3\rho - r + 1$. In addition, assume that $\text{dist}_{\text{val}(D)}(t_{b_i, \sigma_i}, t_{b_j, \sigma_j}) \leq 2r + 1$, which means that $\text{dist}_{\text{val}(D)}(b, b') \leq 2r + 1$ for some $b \in \text{ran}(t_{b_i, \sigma_i})$ and $b' \in \text{ran}(t_{b_j, \sigma_j})$. We will deduce a contradiction. Let $c \in \text{ran}(t_{c_i, \sigma_i})$ and $c' \in \text{ran}(t_{c_j, \sigma_j})$ such that $\eta_{p_i}(c) = b$ and $\eta_{p_j}(c') = b'$. Since $p_j = p_i q$, we know that η_q is an embedding of $\text{val}(A_j)$ into $\text{val}(A_i)$.

By Lemma 22 we have $\text{dist}_{\text{val}(A_i)}(c, \eta_q(c')) \leq 2r + 1$. Since the neighborhoods $\mathcal{N}_{\mathcal{E}(A_i), r}(t_{c_i, \sigma_i})$ and $\mathcal{N}_{\mathcal{E}(A_j), r}(t_{c_j, \sigma_j})$ are connected, it follows that $\mathcal{N}_{\mathcal{E}(A_i), r}(t_{c_i, \sigma_i} \sqcup \eta_q \circ t_{c_j, \sigma_j}) \subseteq \text{val}(A_i)$ is connected. Since $t_{c_i, \sigma_i} \sqcup (\eta_q \circ t_{c_j, \sigma_j})$ is a partial k -tuple, we obtain

$$\text{dist}_{\text{val}(A_i)}(c_i, \eta_q(c_j)) \leq (2r + 1)(k - 1) = (2rk - r + k - 1) - r = \rho - r.$$

Since $\mathcal{N}_{\mathcal{E}(A_i), \rho}(c_i) \simeq \mathcal{B}_i$ and $\mathcal{N}_{\mathcal{E}(A_j), \rho}(c_j) \simeq \mathcal{B}_j$ are valid substructures of $\mathcal{E}(A_i)$ and $\mathcal{E}(A_j)$, respectively, we know that $\text{dist}_{\text{val}(A_i)}(\text{In}_{A_i}, c_i) \leq \rho$ and $\text{dist}_{\text{val}(A_j)}(\text{In}_{A_j}, c_j) \leq \rho$. Consequently, $\text{dist}_{\text{val}(A_i)}(\text{In}_{A_i}, \eta_q(\text{In}_{A_j})) \leq 2\rho + \rho - r = 3\rho - r$. This is a contradiction, since the fact that q has length at least $3\rho - r + 1$ means that $\text{dist}_{\text{val}(A_i)}(\text{In}_{A_i}, \eta_q(\text{In}_{A_j})) > 3\rho - r$ due to the apex condition for D .

We conclude that $\text{dist}_{\text{val}(D)}(b, b') > 2r + 1$ for every $b \in \text{ran}(t_{b_i, \sigma_i})$ and $b' \in \text{ran}(t_{b_j, \sigma_j})$, which means that $\text{dist}_{\text{val}(D)}(t_{b_i, \sigma_i}, t_{b_j, \sigma_j}) > 2r + 1$. ◀

By Lemmas 21 and 23 it remains to show the following:

- (i) We can 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 $3\rho - r$.
- (ii) Assuming (i) holds we can check in time $f(d, |\phi|)$ whether $\text{dist}_{\text{val}(D)}(t_{b_i, \sigma_i}, t_{b_j, \sigma_j}) > 2r + 1$. Let us start with (ii) and assume that $p_j = p_i q$ (the case $p_i = p_j q$ can be handled analogously) with q of length at most $3\rho - r$. In order to check whether $\text{dist}_{\text{val}(D)}(t_{b_i, \sigma_i}, t_{b_j, \sigma_j}) > 2r + 1$, we have to check whether $\text{dist}_{\text{val}(D)}(b, b') > 2r + 1$ for all $b \in \text{ran}(t_{b_i, \sigma_i})$ and $b' \in \text{ran}(t_{b_j, \sigma_j})$. We explain how this can be done for fixed $b \in \text{ran}(t_{b_i, \sigma_i})$ and $b' \in \text{ran}(t_{b_j, \sigma_j})$. As before, let $c \in \text{ran}(t_{c_i, \sigma_i})$ and $c' \in \text{ran}(t_{c_j, \sigma_j})$ such that $\eta_{p_i}(c) = b$ and $\eta_{p_j}(c') = \eta_{p_i}(\eta_q(c')) = b'$. By Lemma 22, $\text{dist}_{\text{val}(D)}(b, b') \leq 2r + 1$ implies $\text{dist}_{\text{val}(A_i)}(c, \eta_q(c')) \leq 2r + 1$. Since the other direction is trivial, it suffices for (ii) to check $\text{dist}_{\text{val}(A_i)}(c, \eta_q(c')) > 2r + 1$. Note that the

A_i -representation of $\eta_q(c')$ can be computed in time $\mathcal{O}(\rho)$: c' is given in its A_j -representation (q', v') , hence, the A_i -representation of $\eta_q(c')$ is (qq', v') , where q has length at most $3\rho - r$.

In order to check $\text{dist}_{\text{val}(A_i)}(c, \eta_q(c')) > 2r + 1$, we 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 . This is analogous to the construction of the expansions in Section D.3.1. Then, we check whether $\eta_q(c') \in \mathcal{N}_{\text{val}(A_i), 2r+1}(c)$. Since both c and $\eta_q(c')$ are given by A_i -representations of size $f(d, |\phi|)$, this can be done in time $f(d, |\phi|)$.

For (i) we show that one can check in time $f(d, |\phi|)$ whether $p_i = p_j q$ for a path q of length at most $3\rho - r$. The general idea is to check whether $p_i = p_j$ and, if this is not the case, to remove repeatedly the last edge of p_i (for at most $3\rho - r$ times) and check whether the resulting path equals p_j . Recall that the path p_i is stored as the min-max-contracted representation \tilde{p}_i (in dag_i) of the corresponding path $p_i 0 A'_i$ in $\text{dag}(D)'$, where in addition all maximal non-branching paths in $p_i 0 A'_i$ have been contracted, and analogously for p_j (see Section D.3.3). Checking $p_i = p_j$ can be done by simply checking whether $\text{lex}(p_i) = \text{weight}(\tilde{p}_i) = \text{weight}(\tilde{p}_j) = \text{lex}(p_j)$; see (7). Hence, it remains to show how we can obtain from \tilde{p}_i a min-max-contracted representation $\text{shorten}(\tilde{p}_i)$, which represents the path obtained from p_i by removing the last edge (using this, one can also remove in a first step the terminal edge $(A_i, 0, A'_i)$ that was added to $\text{dag}(D)$ in the construction of $\text{dag}(D)'$). Here we have to consider two aspects:

- Maximal non-branching subpaths have been contracted into single edges in the construction of dag_i from $\text{dag}(D)'$.
 - Maximal subpaths consisting of min-edges (max-edges, respectively) have been contracted.
- To address both aspects, we compute $\text{shorten}(\tilde{p}_i)$ in two steps. Let $\tilde{p}_i = \tilde{q}(B, \ell, A)$ (we assume that we have already done some shortening steps, so that the A might be no longer A'_i).

Step 1: If (u, ℓ, v) is neither a min- nor a max-triple, then (B, ℓ, A) is an edge in dag_i and we pass $\tilde{p}' := \tilde{q}$ together with the removed edge (B, ℓ, A) to Step 2 below. If $\ell = \text{min}$ and $(B, 1, A)$ is an edge of dag_i then we again pass $\tilde{p}' := \tilde{q}$ and the edge $(B, 1, A)$ to Step 2. If $(B, 1, A)$ is not an edge of dag_i , then $(\wp_{\text{min}}(B, A), 1, A)$ is an edge in dag_i and we pass $\tilde{p}' := \tilde{q}(B, \text{min}, \wp_{\text{min}}(B, A))$ and the edge $(\wp_{\text{min}}(B, A), 1, A)$ to Step 2. Finally, the case $\ell = \text{max}$ can be handled analogously. Just like described in the enumeration procedure of Section D.3.4, we can also compute the **weight**-values and total weight for \tilde{p}' .

Step 2: From Step 1 we obtain a min-max-contracted path representation \tilde{p}' together with an edge (B', ℓ, A') from dag_i . In general we removed from \tilde{p} not only a single edge in $\text{dag}(D)'$ but a maximal non-branching subpath that is represented by the dag_i -edge (B', ℓ, A') . Therefore, we have to add to \tilde{p}' the prefix of this maximal non-branching subpath without the last edge. To do this, we store in the preprocessing for every edge in dag_i the information whether it is a single edge of $\text{dag}(D)'$ or an edge that represents a contracted maximal non-branching path of length at least 2. If (B', ℓ, A') is a single edge from dag_i , then we are done and set $\text{shorten}(\tilde{p}_i) = \tilde{p}'$. If (B', ℓ, A') is an edge of dag_i that represents a maximal non-branching path from B' to A' of length at least 2 in $\text{dag}(D)'$, then we set $\text{shorten}(\tilde{p}_i) = \tilde{p}'(B', \ell, A'')$, where A'' is the parent node of A' on the maximal non-branching path from B' to A' (and we also store the information whether (B', ℓ, A'') represents still a maximal non-branching path of length at least 2).

This correctly constructs the min-max-contracted representation $\text{shorten}(\tilde{p}_i)$ for the path p_i with the last edge removed. However, it remains to explain how the construction from Step 2 can be implemented in constant time. This can again be achieved by a next link data structure; see Theorem 20. In the preprocessing, we compute a forest of all the reversed maximal non-branching paths of $\text{dag}(D)'$ and then we compute a next link data structures for

all the trees in this forest. So, in some sense we use a two-level next-link data structure: the upper level handles contracted sequences of min-edges and max-edges, respectively, whereas the lower level handles contracted sequences of maximal non-branching edges.

► **Remark 24.** Note that the above shortening of p_i destroys the representation of the path p_i (and similarly for p_j). This is a problem, since later in the enumeration phase p_i might be needed again for comparison with other paths (and the number of these comparisons depends on the structure $\text{val}(D)$). Producing a copy of p_i before we start shortening p_i is not an option since the min-max-contracted representation of p_i does not fit into a constant number of registers. Therefore making a copy of p_i would flaw the constant delay requirement. Fortunately, there is a simple solution. Whenever we remove a terminal edge from the min-max-contracted representation of the path p_i we store this edge. In total we have to store only $3\rho - r$ many edges. Then, when we want to restore p_i we add the removed edges at the end, which can be easily done with the min-max-contracted representations (it is similar to the shortening procedure).

► **Remark 25.** The algorithm from this section also yields the last piece for checking in the preprocessing phase, whether a sentence of the form (5) holds (see the last paragraph in Appendix D.1). Recall that (5) expresses that there exists at least $q = f(|\phi|)$ many nodes a_1, \dots, a_q with the following properties: $\text{dist}_{\text{val}(D)}(a_i, a_j) > 2r$ and the r -neighborhood type of each a_i is from a fixed set of r -neighborhood types.

D.3.7 Final Remarks

For a fixed consistent factorization $\Lambda = (\mathcal{B}_1, \sigma_1, \dots, \mathcal{B}_m, \sigma_m)$ of a (k, r) -neighborhood type \mathcal{B} , the algorithm keeps a min-max-contracted initial-to-leaf path in dag_i for every $i \in [m]$. We have seen in Section D.3.4 how all initial-to-leaf paths of dag_i can be enumerated by exploiting the min-max-contracted representation. In Appendix D.3.6, we discussed how the stored min-max-contracted initial-to-leaf paths can be used in order to check the disjointness of the r -neighborhoods of the produced tuples. Consequently, we can perform Algorithm 1 also in the compressed setting.

Recall from Theorem 7 that nodes of $\text{val}(D)$ are output in lex-presentation. Internally, the algorithm represents a node $b \in \text{val}(D)$ by a triple $(\text{lex}(p), q, v)$, where the initial path p is given by a min-max-contracted representation, and (q, v) is the A -representation (A is the nonterminal, where p ends) of a node from the expansion $\mathcal{E}(A)$. Hence, q is a path in $\text{dag}(D)$ of length at most $2\rho + 1$ (due to the apex condition). We can therefore compute from the precomputed edge weights also $\text{weight}(q)$ with 2ρ binary additions. Since (pq, v) is the D -representation of the node b , its lex-representation is $(\text{lex}(pq), v) = (\text{lex}(p) + \text{lex}_A(q), v) = (\text{weight}(p) + \text{weight}(q) + 1, v)$ which can be computed in time $\mathcal{O}(\rho)$ from the internal representation $(\text{lex}(p), q, v)$ of b .