Query Evaluation Over SLP-Represented Document Databases With Complex Document Editing

Markus L. Schmid, Nicole Schweikardt

Query Evaluation Over SLP-Represented Document Databases With Complex Document Editing

Markus L. Schmid
MLSchmid@MLSchmid.de
Humboldt-Universität zu Berlin
Germany

Nicole Schweikardt
schweikn@informatik.hu-berlin.de
Humboldt-Universität zu Berlin
Germany

ABSTRACT

It is known that the query result of a regular spanner over a single document $\mathbf{D}$ can be enumerated after $O(|\mathbf{D}|)$ preprocessing and with constant delay in data complexity (Florenzano et al., ACM TODS 2020, Amarilli et al., ACM TODS 2021). It has been shown (Schmid and Schweikardt, PODS'21) that if the document is represented by a straight-line program (SLP) $\mathcal{S}$, then enumeration is possible with a delay of $O(\log |\mathbf{D}|)$, but with preprocessing that is linear in $|\mathcal{S}|$ (which, in the best case, is logarithmic in $|\mathbf{D}|$). Hence, this compressed setting allows for spanner evaluation in sub-linear time, i. e., with logarithmic upper bounds for preprocessing and delay, if the document is highly-compressible.

In this work, we extend these results to the dynamic setting. We consider a document database $\mathrm{DDB} = \{\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_m\}$ that is represented by an SLP $\mathcal{S}_{\mathrm{DDB}}$, and that supports regular spanners $M_1, M_2, \ldots, M_k$ (meaning that we have data structures at our disposal that allow $O(\log |\mathbf{D}_i|)$-delay enumeration of the result of spanner M_j on document $\mathbf{D}_i$). Then we can perform an update by manipulating the existing documents of DDB by a sequence of text-editing operations commonly found in text-editors (like copy and paste, deleting, or copying factors, concatenating documents etc.), and add the thus constructed document to the database. Such an operation is called *complex document editing* and is given by an expression φ in a suitable algebra. Moreover, after this operation, the document database still supports all the regular spanners $M_1, \ldots, M_k$. The total time required for such an update is $O(k \cdot |\varphi| \cdot \log \mathbf{d})$, where $\mathbf{d}$ is the maximum length of any intermediate document constructed in the complex document editing described by φ. We stress the fact that the size $|\mathcal{S}_{\mathrm{DDB}}|$ of the SLP (which upper bounds the preprocessing in the static case) is potentially logarithmic in the data, but generally depends on the compressibility of the documents (in the worst case, it is even linear in the data). In contrast to that, we can guarantee that the dependency on the data of our updates is logarithmic regardless of the actual compression achieved by the SLP. In particular, any such update performed by complex document editing adds documents whose length may be exponentially larger than the time needed for performing such an update.

Our approach hinges on balancing properties of SLPs, and for our updates it is vital to manipulate the SLP that represents the database in such a way that these balancing properties are maintained.

PODS '22, June 12–17, 2022, Philadelphia, PA, USA

ACM ISBN 978-1-4503-9260-0/22/06…$15.00
https://doi.org/10.1145/3517804.3524158

CCS CONCEPTS

• **Information systems** → *Information retrieval*; • **Theory of computation** → *Database query languages (principles)*; *Data structures and algorithms for data management*; *Design and analysis of algorithms*; *Automata extensions*; *Regular languages.*

KEYWORDS

Algorithmics on Compressed Data, Document Spanners, Dynamic Setting, Information Extraction, Straight-Line Programs

ACM Reference Format:
Markus L. Schmid and Nicole Schweikardt. 2022. Query Evaluation Over SLP-Represented Document Databases With Complex Document Editing. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems (PODS '22), June 12–17, 2022, Philadelphia, PA, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3517804.3524158

1 INTRODUCTION

Since its introduction, the information extraction framework of *document spanners* [4] has intensively been investigated in the database theory community. For example, query results of regular spanners can be enumerated with linear preprocessing and constant delay in data complexity (see [1, 5]). In [22], spanner evaluation over *compressed data* has been investigated, where the document $\mathbf{D}$ is represented by a *straight-line program* (SLP) — a lossless compression scheme for textual data widely used in different areas of theoretical computer science and particularly well-suited for algorithmics on compressed data. The main result of [22] is that after a preprocessing linear in the size of the SLP the query result of a spanner can be enumerated with $O(\log |\mathbf{D}|)$ delay (in data complexity). In the best case, the SLP's size is logarithmic in $|\mathbf{D}|$ as well, which means that the SLP-compressed setting allows *sub-linear* spanner enumeration, where the dependency on the data size is only logarithmic. In this paper, we extend the SLP-compressed setting to the dynamic case, i. e., where the data is subject to updates.

The considered setting is as follows. A document database $\mathrm{DDB} = \{\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_m\}$ is represented by an SLP $\mathcal{S}$ and we have several data structures at hand that allow the enumeration of spanners $M_1, M_2, \ldots, M_k$ on any document $\mathbf{D}_i$ in DDB with delay $O(\log |\mathbf{D}_i|)$. We then want to manipulate the existing documents of DDB by a sequence of text-editing operations and evaluate chosen spanners on the resulting document. For example, we cut the subword from position 5 to 21 from document $\mathbf{D}_7$, insert it at position 12 into document $\mathbf{D}_3$, append this document to $\mathbf{D}_1$, and then run spanners M_5

and M_8 on the resulting document. Such a sequence of operations is called *complex document editing* and is described by an expression φ of a suitable algebra.

We can show that if $\mathcal{S}$ is *balanced* (to be explained in more detail later on), then we can construct and add the document $\mathbf{D}_\varphi$ described by φ to DDB, update all data structures needed for enumerating the spanners $M_1, \ldots, M_k$, and maintain the balancedness property of $\mathcal{S}$ in time $O(k \cdot |\varphi| \cdot \log \mathbf{d})$ in data complexity, where $\mathbf{d}$ is the maximum length of any intermediate document constructed in the complex document editing described by φ. Since $\mathbf{d}$ is upper bounded by $2^{|\varphi|} \cdot \mathbf{L}$ (this can be shown by induction), where $\mathbf{L} := \max\{|\mathbf{D}| : \mathbf{D} \in \mathrm{DDB}\}$ is the maximum length of any document in DDB, this running time can also be stated as $O(k \cdot |\varphi| \cdot (|\varphi| + \log \mathbf{L}))$.

This means that we can add large documents to the database in time that is only logarithmic in the size of the new documents, or logarithmic in the size of the current database's documents. Consequently, just like [22] shows that the SLP-compressed setting allows sub-linear (static) evaluation, we show that the SLP-compressed setting allows sub-linear updates. Moreover, while the preprocessing of [22] is sublinear only if the document is highly compressible, and only the delay is guaranteed to be logarithmic, we show in this paper that, regardless of the compressibility of the data, the dependency on the data is always at most logarithmic in our updates.

In order to give some intuition for this result, we shall briefly explain how SLPs work. An SLP is a directed acyclic graph whose nodes all have a *left* and a *right successor*, except for the sinks T_x of the graphs, which uniquely represent the symbols x of the alphabet (see Figure 1 for an illustration). Any node A with left and right successors B and C represents the document $\mathfrak{D}(A) = \mathfrak{D}(B)\mathfrak{D}(C)$, where $\mathfrak{D}(T_x) = x$ for the sinks T_x. For example, for node B in Figure 1, we have

$$\begin{aligned}\mathfrak{D}(B) &= \mathfrak{D}(E)\mathfrak{D}(C) = \mathfrak{D}(T_{\mathsf{a}})\mathfrak{D}(T_{\mathsf{b}})\mathfrak{D}(F)\mathfrak{D}(T_{\mathsf{a}}) \\ &= \mathfrak{D}(T_{\mathsf{a}})\mathfrak{D}(T_{\mathsf{b}})\mathfrak{D}(T_{\mathsf{b}})\mathfrak{D}(T_{\mathsf{c}})\mathfrak{D}(T_{\mathsf{a}}) = \mathsf{abbca}\,.\end{aligned}$$

For achieving this paper's main result, we exploit the fact that $\mathbf{D}_\varphi$ can be constructed from (parts of) documents that are already represented in a concise way, i. e., by parts of our SLP. However, for identifying the correct parts of the SLP, we need to traverse paths of the DAG. To ensure that these paths are short, we work with SLPs that are *balanced* in such a way that any path starting in some node A has length $O(\log |\mathfrak{D}(A)|)$. Consequently, the balancedness property is vital for our running time bound, and the main technical challenge tackled in this paper is that our updates must maintain this property of the SLP. It turns out that the particular balancing property known from AVL-trees and investigated in the context of SLPs in [7, 20] suits our needs here.

There are two further aspects to be discussed. Firstly, if we want to support a completely new spanner for our document database, then we have to compute the necessary data structures from scratch, which can be done (as in [22]) in time linear in the size of the SLP that represents the database (in data complexity). Secondly, if we want to add a new document $\mathbf{D}'$ that cannot conveniently be described by complex document editing of the existing documents, then we can proceed as follows. In case that $\mathbf{D}'$ is already represented by a balanced SLP $\mathcal{S}'$, we can just add it in time $O(|\mathcal{S}'|)$ and update our data structures in the same time. If $\mathcal{S}'$ is not balanced yet, then we have to first balance it. Using recent balancing results for SLPs [7, 8], this can be done in time $O(|\mathcal{S}'| \log |\mathbf{D}'|)$. Finally, if we are given $\mathbf{D}'$ as a plain text in an uncompressed form, then we cannot avoid to first compute an SLP for it. This, however, is a well-known algorithmic task with many existing efficient algorithms (cf. the discussion in Section 5).

Organisation. Since the original contributions of this paper concern the dynamic setting for SLP-represented document databases and complex document editing, we start with this aspect and defer the discussion of how our results can be applied to document spanners to the end of our exposition.

Section 2 fixes basic notation and recalls the concept of straight-line programs (SLPs). Section 3 presents our data model of SLP-represented document databases. In Section 4 we define our dynamic setting based on complex document editing, and we prove our main results on maintaining SLP-represented document databases under complex document editing. In Section 5 we discuss the task of adding new documents to the database. Section 6 is devoted evaluating document spanners on document databases in the presence of complex document editing. We conclude with some final remarks given in Section 7. Due to space restrictions, many proof details had to be deferred to the paper's full version.

2 PRELIMINARIES

Let $\mathbb{N} = \{1, 2, 3, \ldots\}$ and $[n] = \{1, 2, \ldots, n\}$ for $n \in \mathbb{N}$. For a (partial) mapping $f : X \to Y$, we write $f(x) = \perp$ for some $x \in X$ to denote that $f(x)$ is not defined; and we set $\mathrm{dom}(f) = \{x : f(x) \neq \perp\}$. By $\mathcal{P}(A)$ we denote the power set of a set A, and A^+ denotes the set of non-empty words over A, and $A^* = A^+ \cup \{\varepsilon\}$, where ε is the empty word. For a word $w \in A^*$, $|w|$ denotes its length (in particular, $|\varepsilon| = 0$), and for every $b \in A$, $|w|_b$ denotes the number of occurrences of b in w. A word $v \in A^+$ is a *factor* (or *subword*) of a word $w \in A^+$ if there are $u_1, u_2 \in A^*$ with $w = u_1 v u_2$.

For all our algorithmic considerations, we assume the RAM-model with logarithmic word-size as our computational model.

We fix a finite set Σ, the *terminal alphabet*, whose size will be treated as a constant. *Documents* are just words $\mathbf{D} \in \Sigma^+$, and $|\mathbf{D}|$ denotes their length. We use *straight line programs* as concise representations of documents.

A *straight-line program* (SLP) is a tuple $\mathcal{S} = (N, \Sigma, R)$, where N is the finite set of *non-terminals*, Σ is the *terminal alphabet*, and $R \subseteq N \times (N \cup \Sigma)^+$ is a finite set of *rules* satisfying the following conditions: R is a function $N \to (N \cup \Sigma)^+$ and the relation $\{(A, B) : (A, w) \in R, |w|_B \geqslant 1\}$ is acyclic (this is also called the *acyclicity condition*). As a convention, we write rules $(A, w) \in R$ also in the form $A \to w$; and viewing R as a function, we write $R(A)$ to denote the rule's right hand side w. The *size* of $\mathcal{S}$ is defined by $|\mathcal{S}| := |N| + \sum_{(A,w)\in R} |w|$.

For every $A \in N$ and $a \in \Sigma$, the 1-*step derivation* is defined by $\mathrm{D}_\mathcal{S}(A) := R(A)$ and $\mathrm{D}_\mathcal{S}(a) := a$. We extend the 1-step derivation to a morphism $(N \cup \Sigma)^+ \to (N \cup \Sigma)^+$ by $\mathrm{D}_\mathcal{S}(\alpha_1 \ldots \alpha_n) := \mathrm{D}_\mathcal{S}(\alpha_1) \ldots \mathrm{D}_\mathcal{S}(\alpha_n)$, for $\alpha_i \in (N \cup \Sigma)$, $1 \leqslant i \leqslant n$. For every $\alpha \in (N \cup \Sigma)^+$, we set $\mathrm{D}^1_\mathcal{S}(\alpha) := \mathrm{D}_\mathcal{S}(\alpha)$ and $\mathrm{D}^k_\mathcal{S}(\alpha) := \mathrm{D}_\mathcal{S}(\mathrm{D}^{k-1}_\mathcal{S}(\alpha))$, for every $k \geqslant 2$. The word $\mathfrak{D}_\mathcal{S}(\alpha) := \mathrm{D}^{|N|}_\mathcal{S}(\alpha)$ is the *derivative* of α. Due to the acyclicity condition, $\mathfrak{D}_\mathcal{S}(\alpha) \in \Sigma^+$ for every $\alpha \in (N \cup \Sigma)^+$. The

order of a non-terminal $A \in N$ is defined by $\mathrm{ord}_{\mathcal{S}}(A) := \min\{k : \mathsf{D}^k_{\mathcal{S}}(A) = \mathfrak{D}_{\mathcal{S}}(A)\}$. If the SLP under consideration is clear from the context, we also drop the subscript $\mathcal{S}$ from $\mathsf{D}_{\mathcal{S}}(\cdot)$, $\mathfrak{D}_{\mathcal{S}}(\cdot)$ and $\mathrm{ord}_{\mathcal{S}}(\cdot)$.

SLPs as Compressions. The main purpose of SLPs is their use as compressed representations of words. Their prominence in various areas of computer science is due to the fact that they are mathematically easy, and that they cover many practically applied dictionary-based compression schemes (e. g., run-length encoding and the Lempel-Ziv-family LZ77, LZ78, LZW, etc.). A comprehensive introduction to SLPs is beyond the scope of this paper (we refer to the survey [12] and the papers [11, 13, 16, 17, 20, 22, 23]). In the best case, SLPs can be exponentially smaller than the strings they represent, and since redundancies in texts are likely in practical scenarios, good compressibility is to be expected in most cases. Moreover, many fast practical algorithms for computing SLPs exist.

Normal Forms and Balanced SLPs. An SLP $\mathcal{S} = (N, \Sigma, R)$ is in *Chomsky normal form* if, for every rule $(A \to w) \in R$ we have $w \in (N^2 \cup \Sigma)$. We say that an SLP is in *normal form* if it is in Chomsky normal form and $\{T_x : x \in \Sigma\} \subseteq N$, such that $(T_x \to x) \in R$ for every $x \in \Sigma$, and $(A \to BC) \in R$ with $B, C \in N$ for every $A \in N \setminus \{T_x : x \in \Sigma\}$. We call T_x the *leaf non-terminals* and all other $A \in N \setminus \{T_x : x \in \Sigma\}$ *inner non-terminals*. We note that if $\mathcal{S}$ is in normal form, then $|\mathcal{S}| \leqslant 3|N|$.

For an SLP in normal form, we shall also use the following convenient notation. For every rule $A \to BC$, we set $\mathsf{lc}(A) = B$ and $\mathsf{rc}(A) = C$. This means that we can also represent the set of rules by functions $\mathsf{lc}(\cdot)$ and $\mathsf{rc}(\cdot)$, and for every inner non-terminal A, the rule for A is $A \to \mathsf{lc}(A)\mathsf{rc}(A)$.

An SLP $\mathcal{S} = (N, \Sigma, R)$ in normal form can be represented by an edge-labelled directed acyclic graph (DAG) as follows. The set N of non-terminals is the set of nodes, and the edge-relation is given by the functions $\mathsf{lc}(\cdot)$ and $\mathsf{rc}(\cdot)$. I.e., from every inner non-terminal $A \in N$, there is a *left arc* to its *left child* $\mathsf{lc}(A)$, and a *right arc* to its *right child* $\mathsf{rc}(A)$. We assume that the DAG has arc labels that indicate whether an arc is a left or a right arc. This graph is acyclic due to the acyclicity condition of the SLP. In the following, we call this graph the $\mathcal{S}$-DAG (see Figure 1 for an $\mathcal{S}$-DAG). The sinks (i. e., nodes without outgoing arcs) are exactly the leaf non-terminals. The longest path from a node A to a sink has exactly length $\mathrm{ord}(A) - 1$ (note that sinks have a length-0 path to a sink and, by definition, they have order 1). We say that B is a *descendant* of A if $A \neq B$ and there is a path from A to B in the $\mathcal{S}$-DAG.

We say that non-terminal A is *c-shallow* for some $c \in \mathbb{N}$ if $\mathrm{ord}(A) \leqslant c \cdot \log|\mathfrak{D}(A)|$. For every non-terminal $A \in N$ with rule $A \to BC$, we define $\mathrm{bal}(A) = \mathrm{ord}(B) - \mathrm{ord}(C)$; we say that A is *balanced* if $\mathrm{bal}(A) \in \{-1, 0, 1\}$; and A is *strongly balanced* if A and all its descendants are balanced. We call an SLP $\mathcal{S}$ in normal form *strongly balanced* (*c-shallow*, resp.) if all its inner non-terminals are strongly balanced (*c*-shallow, resp.). The notions of *c*-shallow SLPs and strongly balanced SLPs have been studied in the literature (cf., [7, 8, 20]). The following can be easily shown by induction (cf., [7]; for the reader's convenience, a proof is also provided in Appendix A).

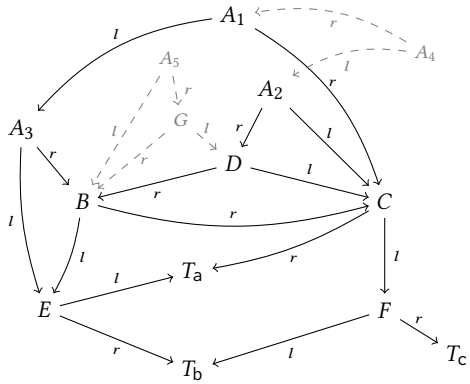


Figure 1: The $\mathcal{S}$-DAG of the SLP from Example 3.1 is shown in black (and solid arcs). Left and right arcs are indicated by labels l and r, respectively. The grey parts illustrate the extensions discussed in Example 4.1.

Lemma 2.1. *Let $\mathcal{S}$ be an SLP in normal form and let A be a strongly balanced non-terminal of $\mathcal{S}$. Then, every directed path of the $\mathcal{S}$-DAG that starts in A and ends in a leaf non-terminal of $\mathcal{S}$ has length at least $\frac{1}{2}\log|\mathfrak{D}(A)|$ and at most $2\log|\mathfrak{D}(A)|$. Furthermore, $\log|\mathfrak{D}(A)| \leqslant \mathrm{ord}(A) - 1 \leqslant 2 \cdot \log|\mathfrak{D}(A)|$.*

3 DOCUMENT DATABASES

A *document database* (*over* Σ) is a finite collection $\mathsf{DDB} = \{\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_m\}$ of documents (over Σ). We will represent a document database by an SLP as follows. For an SLP $\mathcal{S} = (N, \Sigma, R)$, we call $\mathrm{docs}(\mathcal{S}) := \{\mathfrak{D}(A) : A \in N\}$ the *set of documents represented by* $\mathcal{S}$. The SLP $\mathcal{S}$ is viewed as a *representation* for a document database DDB if $\mathsf{DDB} \subseteq \mathrm{docs}(\mathcal{S})$. This means that for every $\mathbf{D} \in \mathsf{DDB}$ there is a non-terminal $A_{\mathbf{D}} \in N$ with $\mathfrak{D}(A_{\mathbf{D}}) = \mathbf{D}$, but, since in general $\mathcal{S}$'s compression requires many factors of documents to be represented by non-terminals, there may also be non-terminals $A \in N$ that represent a document $\mathfrak{D}(A) \notin \mathsf{DDB}$.

We assume that along with the SLP that represents DDB we store a mapping $\mathbf{D} \mapsto A_{\mathbf{D}}$, such that for every document $\mathbf{D}$ from DDB we can access $A_{\mathbf{D}}$ in constant time. For simplicity, we also say that an SLP $\mathcal{S}$ represents a $\mathsf{DDB} = \{\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_m\}$ *by non-terminals* $A_1, A_2, \ldots, A_m$ to indicate that $\mathfrak{D}(A_i) = \mathbf{D}_i$ for every $i \in [m]$.

Example 3.1. *Let $\mathcal{S} = (N, \Sigma, R)$ be an SLP in normal form that represents a document database $\mathsf{DDB} = \{\mathbf{D}_1, \mathbf{D}_2, \mathbf{D}_3\}$ by non-terminals A_1, A_2, A_3 with the $\mathcal{S}$-DAG illustrated in Figure 1. Note that $\mathfrak{D}(F) = \mathsf{bc}$, $\mathfrak{D}(E) = \mathsf{ab}$, $\mathfrak{D}(C) = \mathfrak{D}(F)\mathsf{a} = \mathsf{bca}$, $\mathfrak{D}(B) = \mathfrak{D}(E)\mathfrak{D}(C) = \mathsf{abbca}$, $\mathfrak{D}(D) = \mathfrak{D}(C)\mathfrak{D}(B) = \mathsf{bcaabbca}$. This means that $\mathbf{D}_3 = \mathfrak{D}(A_3) = \mathfrak{D}(E)\mathfrak{D}(B) = \mathsf{ababbca}$, $\mathbf{D}_2 = \mathfrak{D}(A_2) = \mathfrak{D}(C)\mathfrak{D}(D) = \mathsf{bcabcaabbca}$ and $\mathbf{D}_1 = \mathfrak{D}(A_1) = \mathfrak{D}(A_3)\mathfrak{D}(C) = \mathsf{ababbcabca}$. The orders of $\mathcal{S}$'s non-terminals are as follows: $\mathrm{ord}(F) = \mathrm{ord}(E) = 2$, $\mathrm{ord}(C) = 3$, $\mathrm{ord}(B) = 4$, $\mathrm{ord}(D) = \mathrm{ord}(A_3) = 5$, $\mathrm{ord}(A_1) = \mathrm{ord}(A_2) = 6$. In particular, all non-terminals are balanced except for the non-terminals A_1, A_2, A_3, since $\mathrm{bal}(A_1) = 2$ and $\mathrm{bal}(A_2) = \mathrm{bal}(A_3) = -2$.*

In our algorithms, we will ensure that the SLP $\mathcal{S}$ representing a document database DDB is in normal form and strongly balanced. The SLP will be stored in a *basic* SLP *data structure of* $\mathcal{S}$, i.e., a data

$$
\begin{aligned}
\mathrm{concat}(\mathbf{D}, \mathbf{D}') &= \mathbf{D} \cdot \mathbf{D}' \,, \\
\mathrm{extract}(\mathbf{D}, i, j) &= \mathbf{D}[i, j{+}1\rangle \,, \\
\mathrm{delete}(\mathbf{D}, i, j) &= \mathbf{D}[1, i\rangle \cdot \mathbf{D}[j{+}1, \mathbf{d}{+}1\rangle \,, \\
\mathrm{insert}(\mathbf{D}, \mathbf{D}', k) &= \mathbf{D}[1, k\rangle \cdot \mathbf{D}' \cdot \mathbf{D}[k, \mathbf{d}{+}1\rangle \,, \\
\mathrm{copy}(\mathbf{D}, i, j, k) &= \mathrm{insert}(\mathbf{D}, \mathbf{D}[i, j{+}1\rangle, k) \\
&= \mathbf{D}[1, k\rangle \cdot \mathbf{D}[i, j{+}1\rangle \cdot \mathbf{D}[k, \mathbf{d}{+}1\rangle \,.
\end{aligned}
$$

Figure 2: The CDE-algebra operations; $\mathbf{D}, \mathbf{D}'$ are documents, $\mathbf{d} = |\mathbf{D}|$, i, j are positions of $\mathbf{D}$, and k is a gap of $\mathbf{D}$.

structure that allows, for every inner non-terminal A, constant-time access to the non-terminals $\mathrm{lc}(A)$ and $\mathrm{rc}(A)$ and to the numbers $\mathrm{ord}(A)$ and $|\mathfrak{D}(A)|$. When given the $\mathcal{S}$-DAG of $\mathcal{S}$, this data structure can easily be computed in time $O(|\mathcal{S}|)$ in a bottom-up fashion starting with the sinks of the $\mathcal{S}$-DAG. The purpose of this basic SLP data structure is to facilitate the efficient handling of SLPs.

4 COMPLEX DOCUMENT EDITING

Given a document database $\mathrm{DDB} = \{\mathbf{D}_1, \ldots, \mathbf{D}_m\}$, we want to create new documents by a sequence of text-editing operations. We use the following notation. The elements $1, 2, \ldots, |\mathbf{D}|$ are the *positions*, and the elements $1, 2, \ldots, |\mathbf{D}|{+}1$ are the *gaps* of a document $\mathbf{D}$ (here, "gap k" refers to the gap between positions $k{-}1$ and k of $\mathbf{D}$). For two documents $\mathbf{D}_1, \mathbf{D}_2$, we denote by $\mathbf{D}_1 \cdot \mathbf{D}_2$ the *concatenation* of $\mathbf{D}_1$ and $\mathbf{D}_2$, i. e., the document obtained by appending $\mathbf{D}_2$ at the end of $\mathbf{D}_1$. We often omit the operator '$\cdot$' and use juxtaposition. For a document $\mathbf{D}$ and positions i, j of $\mathbf{D}$ with $i \leqslant j$ we define $\mathbf{D}[i, j{+}1\rangle$ as the factor of $\mathbf{D}$ that starts at position i and ends at position j. For example, $\mathsf{abcaacb}[2, 5\rangle = \mathsf{bca}$.

We introduce an algebra for complex document editing (CDE-algebra) whose basic operations are defined in Figure 2. A *CDE-expression* φ over a document database $\mathrm{DDB} = \{\mathbf{D}_1, \ldots, \mathbf{D}_m\}$ is obtained by nested application of the basic CDE-algebra operations, starting with documents in DDB. For a CDE-expression φ, we denote by $|\varphi|$ the total number of its operations, i. e., the number of internal nodes of its syntax tree. We write $\mathrm{eval}(\varphi)$ for the document obtained from evaluating the expression φ on DDB. For example, the CDE-expression

$$\varphi \;:=\; \mathrm{concat}(\mathbf{D}_1, \mathrm{insert}(\mathbf{D}_3, \mathrm{extract}(\mathbf{D}_7, 5, 21), 12))$$

describes the complex document editing already mentioned in the introduction, i. e., the document obtained by taking the factor of $\mathbf{D}_7$ from position 5 to 21, inserting this at gap 12 into document $\mathbf{D}_3$, and then appending this document to $\mathbf{D}_1$. This expression φ has size $|\varphi| = 3$.

Recall from Section 3 that we represent a document database DDB by a strongly balanced SLP in normal form $\mathcal{S} = (N, \Sigma, R)$. In order to represent a new document described by a CDE-expression φ within the same framework, we introduce the notion of an *extension* of $\mathcal{S}$: An SLP $\mathcal{S}' = (N', \Sigma, R')$ is called an *extension* of $\mathcal{S}$ if $\mathcal{S}'$ is in normal form, $N' = N \cup \widetilde{N}$ with $N \cap \widetilde{N} = \emptyset$, and, for every $A \in N$, $R'(A) = R(A)$. We call N the *old* non-terminals and $\widetilde{N}$ the *new* non-terminals. Note that the extension $\mathcal{S}'$ still contains all old non-terminals with exactly the same rules, while new non-terminals $\widetilde{A} \in \widetilde{N}$ can have rules $\widetilde{A} \to BC$, where B and C can be old or new non-terminals. Moreover, for every $A \in N$, $\mathfrak{D}_{\mathcal{S}'}(A) = \mathfrak{D}_{\mathcal{S}}(A)$ and $\mathrm{ord}_{\mathcal{S}'}(A) = \mathrm{ord}_{\mathcal{S}}(A)$; and if A is (strongly) balanced in $\mathcal{S}$, then A is also (strongly) balanced in $\mathcal{S}'$. Another helpful point of view is that the $\mathcal{S}'$-DAG is obtained from the $\mathcal{S}$-DAG by just adding some new nodes with their left and right arcs. An example of an extension is illustrated by the grey part of Figure 1 (see also Example 4.1). For convenience, we often will describe extensions by defining only the set $\widetilde{N}$ of new non-terminals and defining, for each such new non-terminal, its rule.

Example 4.1. *Let* SLP $\mathcal{S} = (N, \Sigma, R)$ *be the SLP from Example 3.1 that is illustrated by Figure 1 and that represents the document database* $\mathrm{DDB} = \{\mathbf{D}_1, \mathbf{D}_2, \mathbf{D}_3\}$ *by the non-terminals* A_1, A_2, A_3.

Now consider the extension $\mathcal{S}_1 = (N \cup \{A_4\}, \Sigma, R \cup \{A_4 \to A_2 A_1\})$. *This extension adds document* $\mathbf{D}_4 := \mathbf{D}_2 \cdot \mathbf{D}_1$, *represented by the new non-terminal* A_4, *to the document database (see Figure 1 for an illustration). We can also create the slightly more complicated extension* $\mathcal{S}_2 = (N \cup \{A_5, G\}, \Sigma, R \cup \{A_5 \to BG, G \to DB\})$. *This extension adds document*

$$\mathbf{D}_5 := \mathfrak{D}(B)\mathfrak{D}(G) = \mathfrak{D}(B)\mathfrak{D}(D)\mathfrak{D}(B) = \mathsf{abbcabcaabbcaabbca}\,,$$

represented by the new non-terminal A_5, *to the document database (this is also illustrated by Figure 1).*

The extensions of Example 4.1 are quite simple, as they only add documents to DDB that are constructed by concatenating existing documents. If, for example, we want to add the document that is obtained by replacing factor $\mathbf{D}_1[2, 6\rangle$ from $\mathbf{D}_1$ with factor $\mathbf{D}_2[3, 7\rangle$, i. e., the document given by the CDE-expression $\mathrm{insert}(\mathrm{delete}(\mathbf{D}_1, 2, 5), \mathrm{extract}(\mathbf{D}_2, 3, 6), 2)$, then finding a suitable extension is more complex. This section's main result shows that such extensions can be computed efficiently.

The remainder of this section is devoted to the proof of our following main theorem (Theorem 4.3). In this theorem, we assume that the CDE-expression φ is represented in a reasonable way where the documents $\mathbf{D}_i \in \mathrm{DDB}$ that occur at the leaves of φ's syntax tree are represented by single non-terminals A_i of the SLP. The *maximum intermediate document size* $|\mathrm{max}_\varphi(\mathrm{DDB})|$ induced by a CDE-expression φ on the document database DDB is the maximum length of any document of DDB that occurs at a leaf of φ's syntax tree, any intermediate document that is represented by a subexpression of φ, and the finally resulting document $\mathrm{eval}(\varphi)$. By induction, one easily obtains the following (see Appendix B for a proof):

Lemma 4.2. *Let* DDB *be a document database and let* φ *be a CDE-expression over* DDB. *Then*

$$|\mathrm{max}_\varphi(\mathrm{DDB})| \;\leqslant\; 2^{|\varphi|} \cdot \max\{\,|\mathbf{D}| : \mathbf{D} \in \mathrm{DDB}\,\}\,.$$

Our main result's statement is that extending an SLP-represented document database DDB according to a given CDE-expression φ, while maintaining the property of being strongly balanced, can be done in time $\mathrm{O}(|\varphi| \cdot \log |\mathrm{max}_\varphi(\mathrm{DDB})|)$ — and by Lemma 4.2, this is in $\mathrm{O}(|\varphi|^2 + |\varphi| \cdot \log \max\{\,|\mathbf{D}| : \mathbf{D} \in \mathrm{DDB}\,\})$.

Theorem 4.3 (CDE extension theorem). *Let* DDB *be a document database that is represented by a strongly balanced* SLP $\mathcal{S}$ *in*

normal form. When given the basic SLP data structure for $\mathcal{S}$ and a CDE-expression φ over DDB*, we can construct a strongly balanced extension $\mathcal{S}'$ of $\mathcal{S}$, along with its basic SLP data structure, and a non-terminal $\widetilde{A}$ of $\mathcal{S}'$, such that $\mathfrak{D}_{\mathcal{S}'}(\widetilde{A}) = \mathsf{eval}(\varphi)$.*

This construction takes time $O(|\varphi|\cdot \log \mathbf{d})$ for $\mathbf{d} := |\max_\varphi(\mathrm{DDB})|$. In particular, the number $|\widetilde{N}|$ of new non-terminals is in $O(|\varphi|\cdot \log \mathbf{d})$.

Note that $O(|\varphi|\cdot \log \mathbf{d}) \subseteq O(|\varphi|^2 + |\varphi|\cdot \log\{\,|\mathbf{D}| : \mathbf{D} \in \mathrm{DDB}\})$.

The proof proceeds by induction on the construction of φ, i.e., bottom-up along φ's syntax tree. For each of the CDE-algebra's operations depicted in Figure 2, we prove a lemma for extending the SLP so that it also represents the result of this operation. The main challenge is to prove according lemmas for the operations concat and extract; the remaining operations can then be implemented easily as they are defined via concatenation and extraction.

The remainder of this section is devoted to the lemmas for concat and extract, which we call *concatenation* and *extraction lemma*.

The concatenation lemma. In our constructions, it can happen that for some nodes the balance factor of balanced nodes changes from -1, 0, or 1 to -2 or 2 (i. e., nodes become unbalanced). As a remedy, a main ingredient of our construction for operation concat is a balancing lemma that allows to re-balance such nodes:

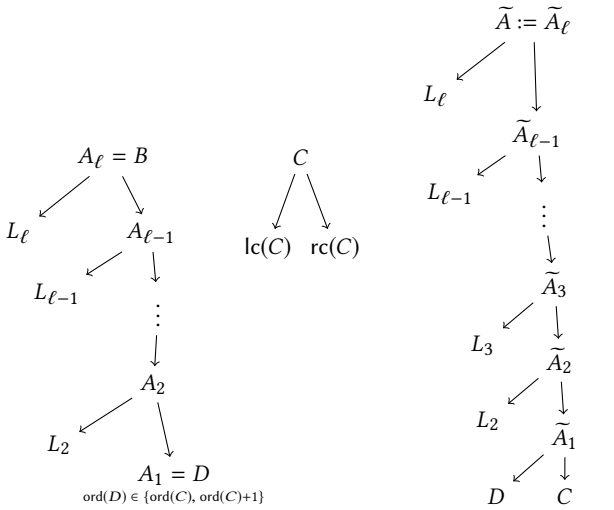
Lemma 4.4 (balancing lemma). *Let $\mathcal{S} = (N, \Sigma, R)$ be an* SLP *in normal form let $A \in N$ with rule either $A \to BC$ or $A \to CB$, let $B \neq C$, let C be strongly balanced, let $\mathrm{ord}(C) = \mathrm{ord}(B) + 2$, and let node C in the $\mathcal{S}$-DAG have in-degree 1 (i. e., A is the only non-terminal such that C has an occurrence in $R(A)$). We can construct an* SLP *$\mathcal{S}' = (N \cup \widetilde{N}, \Sigma, R')$ with the following properties:*

(a) $\widetilde{N} = \{\widetilde{V}\}$ or $\widetilde{N} = \emptyset$.
(b) R' is obtained from R by only changing the rules of A and C and, if $\widetilde{N} \neq \emptyset$, adding the rule for $\widetilde{V}$. The in-degree of non-terminals of $\mathcal{S}$ does not increase in $\mathcal{S}'$.
(c) For every $F \in N \setminus \{C\}$, $\mathfrak{D}_{\mathcal{S}'}(F) = \mathfrak{D}_{\mathcal{S}}(F)$.
(d) A is balanced in $\mathcal{S}'$. Every balanced descendant of A in $\mathcal{S}$ is still balanced in $\mathcal{S}'$. Every descendant of A in $\mathcal{S}'$ either belongs to $\widetilde{N}$ or is a descendant of A in $\mathcal{S}$. If $\widetilde{N} \neq \emptyset$, then $\widetilde{V}$ is a balanced descendant of A in $\mathcal{S}'$.
(e) $\mathrm{ord}_{\mathcal{S}'}(A) \in \{\mathrm{ord}_{\mathcal{S}}(A), \mathrm{ord}_{\mathcal{S}}(A){-}1\}$.

The construction can be carried out in constant time, provided that we have available the basic SLP data structure for all non-terminals of $\mathcal{S}$ that are reachable from A.

The proof of this lemma relies on rotations similar to the ones commonly used for balanced search trees (like AVL-trees). How such rotations can be used for SLPs is folklore and has already been sketched in [20]. For our applications, we needed to formulate and prove the balancing lemma with several additional conditions, and we argue directly on the $\mathcal{S}$-DAG instead of derivation trees. A detailed proof will be provided in the paper's full version.

What follows is the precise statement of the concatenation lemma, along with a brief proof sketch. The lemma's essence and its proof idea have already been sketched in [20]. But for our purposes — as we want to apply it repeatedly within an inductive construction — we need the following more detailed statement and more precise algorithmic properties. Working out these details in a way that fits to our overall framework was non-trivial.



(a) The B-to-D path with $\mathrm{ord}(B) - \mathrm{ord}(C) \geqslant 2$.

(b) The extension $\mathcal{S}_1$ of $\mathcal{S}$.

Figure 3: The main construction of Lemma 4.5. The new non-terminal $\widetilde{A}_1$ is balanced, but all other new non-terminals $\widetilde{A}_2, \ldots, \widetilde{A}_\ell$ may be unbalanced.

Lemma 4.5 (concatenation lemma). *Given the basic SLP data structure of an SLP $\mathcal{S} = (N, \Sigma, R)$ in normal form, and strongly balanced $B, C \in N$, we can compute in time $O(|\mathrm{ord}(B) - \mathrm{ord}(C)|)$ an extension $\mathcal{S}' = (N \cup \widetilde{N}, \Sigma, R')$ of $\mathcal{S}$, along with its basic SLP data structure, and an $\widetilde{A} \in \widetilde{N}$, such that*

$$\mathfrak{D}_{\mathcal{S}'}(\widetilde{A}) = \mathrm{concat}(\mathfrak{D}_{\mathcal{S}}(B), \mathfrak{D}_{\mathcal{S}}(C)). \tag{1}$$

Furthermore, the following is true:

(a) Every $\widetilde{X} \in \widetilde{N}$ is strongly balanced in $\mathcal{S}'$,
(b) $|\widetilde{N}| \leqslant \max\{\,1\,,\; 2\cdot|\mathrm{ord}(B){-}\mathrm{ord}(C)| - 1\,\}$, and
(c) $\mathrm{ord}(\widetilde{A}) \in \{m, m{+}1\}$ for $m := \max\{\mathrm{ord}(B), \mathrm{ord}(C)\}$.

Proof Sketch. We focus on the case where $\mathrm{ord}(B) \geqslant \mathrm{ord}(C){+}2$. We start in node B of the $\mathcal{S}$-DAG and move along the right-arcs until we find a node D with $\mathrm{ord}(D) - \mathrm{ord}(C) \in \{0, 1\}$ (see Figure 3(a)). Then we make a copy of this B-to-D-path (with new non-terminals $\widetilde{A}_i$) and the non-terminal C inserted, as shown in Figure 3(b). It can be seen that $\mathfrak{D}(\widetilde{A}) = \mathrm{concat}(\mathfrak{D}(B), \mathfrak{D}(C))$ and that $\widetilde{A}_1$ is strongly balanced. However, for every $i \in [\ell{-}1]$, $\mathrm{ord}(\widetilde{A}_i) = \mathrm{ord}(A_i) + 1$ is possible, and therefore, $\mathrm{bal}(\widetilde{A}_{i+1}) = \mathrm{bal}(A_{i+1}) - 1 = -2$ is possible, i. e., some $\widetilde{A}_{i+1}$ might be unbalanced. The idea is now to use the balancing lemma (Lemma 4.4) to re-balance all unbalanced $\widetilde{A}_i$. However, the balancing lemma may decrease the order of $\widetilde{A}_i$ (and therefore the balance factors of $\widetilde{A}_j$ with $j > i$), and it can only be used for re-balancing unbalanced non-terminals with balance factor 2 or -2. Therefore, we have to carefully argue inductively that the balancing lemma can in fact always be applied until all non-terminals are balanced. A detailed proof will be provided in the paper's full version. □

The extraction lemma. We now provide the precise statement and a proof sketch of the extraction lemma. The proof relies on two ingredients: First, we identify suitable non-terminals that represent documents whose concatenation precisely yields the desired factor $\mathbf{D}[i, j{+}1\rangle$. Afterwards, we iteratively apply the concatenation lemma to construct the desired extension of the SLP. The main challenge is to achieve all this within the desired time bound.

Lemma 4.6 (extraction lemma). *Given the basic SLP data structure of an SLP $\mathcal{S} = (N, \Sigma, R)$ in normal form, a strongly balanced $A \in N$, and $i, j \in \{1, 2, \ldots, |\mathfrak{D}_{\mathcal{S}}(A)|\}$ with $i \leqslant j$, we can compute in time $O(\mathrm{ord}(A))$ an extension $\mathcal{S}' = (N \cup \widetilde{N}, \Sigma, R')$ of $\mathcal{S}$, along with its basic SLP data structure, and an $\widetilde{A} \in N \cup \widetilde{N}$ such that $\mathfrak{D}_{\mathcal{S}'}(\widetilde{A}) = \mathrm{extract}(\mathfrak{D}_{\mathcal{S}}(A), i, j)$. Furthermore, every $\widetilde{X} \in \widetilde{N} \cup \{\widetilde{A}\}$ is strongly balanced in $\mathcal{S}'$, and $|\widetilde{N}| \leqslant 16{\cdot}\mathrm{ord}(A)$.*

Proof Sketch. Let us first assume we have already proven the lemma's statement for the special cases of $i = 1$ $(*)$ or $j = |\mathfrak{D}(A)|$ $(**)$, but with a smaller size bound on the number of new non-terminals. We argue by induction on $\mathrm{ord}(A)$ that then the lemma's statement also holds in its general form: Let $i, j \in \{1, 2, \ldots, |\mathfrak{D}_{\mathcal{S}}(A)|\}$ with $2 \leqslant i \leqslant j < |\mathfrak{D}(A)|$, and let $A \to BC$ be the rule of A. If $j \leqslant |\mathfrak{D}(B)|$ or if $i > |\mathfrak{D}(B)|$, then we are done by induction (i. e., we can use the lemma's statement on B or C, respectively). Hence, we assume that $i \leqslant |\mathfrak{D}(B)|$ and $j > |\mathfrak{D}(B)|$. Then, we have $\mathfrak{D}(A)[i, j\rangle = D_1 \cdot D_2$ for $D_1 := \mathfrak{D}(B)[i, |\mathfrak{D}(B)|{+}1\rangle$ and $D_2 := \mathfrak{D}(C)[1, j'{+}1\rangle$ with $j' := j - |\mathfrak{D}(B)|$. This means that we have the special case $(**)$ with respect to B and the special case $(*)$ with respect to C. Hence, by assumption, we can compute a non-terminal $\widetilde{B}$ with $\mathfrak{D}(\widetilde{B}) = D_1$ and a non-terminal $\widetilde{C}$ with $\mathfrak{D}(\widetilde{C}) = D_2$. With an application of the concatenation lemma (Lemma 4.5), we can therefore compute a non-terminal $\widetilde{A}$ with $\mathfrak{D}(\widetilde{A}) = D_1 \cdot D_2 = \mathrm{extract}(\mathfrak{D}_{\mathcal{S}}(A), i, j)$.

Next, we discuss how $(*)$ can be proven (the case $(**)$ is symmetric). Let $j \in \{1, \ldots, |\mathfrak{D}(A)|{-}1\}$. Since $|\mathfrak{D}(A)| > j$, we can start in node A and move along the left arcs, until we find for the first time a non-terminal A_1 (with direct predecessor X_1) that satisfies $|\mathfrak{D}(A_1)| \leqslant j$. We also know that $|\mathfrak{D}(X_1)| = |\mathfrak{D}(A_1) \cdot \mathfrak{D}(\mathrm{rc}(X_1))| > j$, since otherwise X_1 would have been a valid choice instead of A_1. Therefore, we can now repeat this step: from $\mathrm{rc}(X_1)$ we move along the left arcs, until we find for the first time a non-terminal A_2 (with direct predecessor X_2) that satisfies $|\mathfrak{D}(A_1){\cdot}\mathfrak{D}(A_2)| \leqslant j$. By iterating this, we can compute descendants $A_1, \ldots, A_\ell$ and $X_1, \ldots, X_\ell$ of A, such that $A_i = \mathrm{lc}(X_i)$ for all $i \in [\ell]$, and X_{i+1} is a descendant of X_i for every $i \in [\ell{-}1]$, and $\mathfrak{D}(A)[1, j{+}1\rangle = \mathfrak{D}(A_1) \cdots \mathfrak{D}(A_\ell)$ (see Figure 4). Moreover, $\ell \leqslant \mathrm{ord}(A)$ and these non-terminals can be computed in time $\mathrm{O}(\mathrm{ord}(A))$.

The next task is to apply the concatenation lemma for $(\ell{-}1)$ times with respect to the non-terminals A_i: The first application constructs a new non-terminal $\widetilde{A}_{\ell-1}$ with $\mathfrak{D}(\widetilde{A}_{\ell-1}) = \mathfrak{D}(A_{\ell-1}) \cdot \mathfrak{D}(A_\ell)$, the second application constructs a new non-terminal $\widetilde{A}_{\ell-2}$ with $\mathfrak{D}(\widetilde{A}_{\ell-2}) = \mathfrak{D}(A_{\ell-2}) \cdot \mathfrak{D}(\widetilde{A}_{\ell-1}) = \mathfrak{D}(A_{\ell-2} \cdot A_{\ell-1} \cdot A_\ell)$, and, in general, the i-th application constructs a new non-terminal $\widetilde{A}_{\ell-i}$ with $\mathfrak{D}(\widetilde{A}_{\ell-i}) = \mathfrak{D}(A_{\ell-i}) \cdot \mathfrak{D}(\widetilde{A}_{\ell-(i-1)}) = \mathfrak{D}(A_{\ell-i} \cdot A_{\ell-i+1} \cdots A_\ell)$.

The overall runtime guarantee obtained from the concatenation lemma is $O(t)$ for $t := \sum_{i=1}^{\ell-1} |\mathrm{ord}(A_{\ell-i}) - \mathrm{ord}(\widetilde{A}_{\ell-(i-1)})|$ (letting $\widetilde{A}_\ell := A_\ell$). We have to show that this is in $\mathrm{O}(\mathrm{ord}(A))$. This is far from trivial, and the naive analysis only yields a poorer runtime guarantee of $O(\mathrm{ord}(A)^2)$. To see why, consider the particular case where ℓ is of size roughly $\mathrm{ord}(A)$ and $X_{i+1} = \mathrm{rc}(X_i)$ for all $i \in [\ell{-}1]$. The concatenation lemma only ensures $\mathrm{ord}(\widetilde{A}) \in \{m, m{+}1\}$ for $m := \max\{\mathrm{ord}(B), \mathrm{ord}(C)\}$, where $\widetilde{A}$ is the non-terminal with $\mathfrak{D}(\widetilde{A}) = \mathrm{concat}(\mathfrak{D}(B), \mathfrak{D}(C))$. Thus, in our iterated application of the concatenation lemma it seems possible that $\mathrm{ord}(\widetilde{A}_{\ell-i}) \approx \mathrm{ord}(A_{\ell-i}) + i$ and hence t might be as large as $\sum_{i=1}^{\ell-1} i$, i.e., of size $\Omega(\ell^2)$.

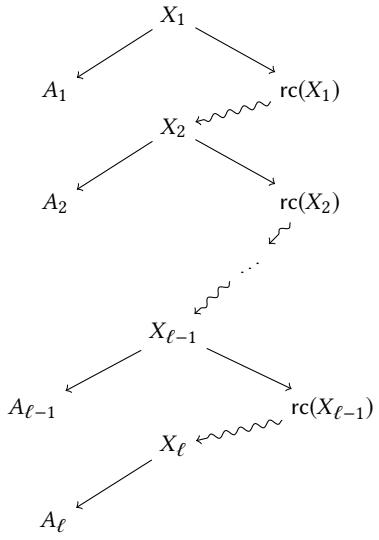


Figure 4: Illustration for handling case $(*)$ in the proof of Lemma 4.6. The solid arcs are left and right arcs; while the snake-shaped arcs are paths (of length $\geqslant 0$) of only left arcs.

To achieve the runtime guarantee of $O(\mathrm{ord}(A))$, we have to make a deeper analysis to obtain a more thorough estimation. By induction we can show that for every $i \in [\ell{-}1]$ there exists an $r_i \in \{0, 1, 2\}$ such that $\mathrm{ord}(\widetilde{A}_{\ell-i}) = \mathrm{ord}(A_{\ell-i}) + r_i$. This implies that $t \leqslant \sum_{i=1}^{\ell-1} r_i + \sum_{i=1}^{\ell-1} (\mathrm{ord}(A_{\ell-i}) - \mathrm{ord}(A_{\ell-(i-1)}))$. The first sum is in $O(\ell)$, the second sum is a telescope adding up to $\leqslant \mathrm{ord}(A)$. In summary, this yields the claimed runtime guarantee of $O(\mathrm{ord}(A))$. A detailed proof will be provided in the paper's full version. □

5 ADDING DOCUMENTS DIRECTLY

In this section, we briefly discuss the following scenario: We already have available the basic SLP data structure of a strongly balanced SLP $\mathcal{S}_{\mathrm{DDB}}$ that represents a document database DDB. Now we want to insert a new document $\mathbf{D}$ into the document database (by suitably extending the SLP $\mathcal{S}_{\mathrm{DDB}}$ and its basic SLP data structure), and we want to do this without applying complex document editing.

We use known results on *single rooted* SLPs, i. e., SLPs whose $\mathcal{S}$-DAG has only one root (a node without incoming edge(s)), which are also denoted by $\mathcal{S} = (N, \Sigma, R, S_0)$, where S_0 is the root. Such SLPs are usually interpreted as representing the single document $\mathfrak{D}(S_0)$. Two single rooted SLPs are *equivalent* if their roots derive the same document.

If $\mathbf{D}$ is given by a single rooted SLP $\mathcal{S}$ with root S_0, then we transform it into an equivalent strongly balanced $\mathcal{S}'$ in normal form. This can be done in time $\mathrm{O}(|\mathcal{S}| \cdot \log |\mathbf{D}|)$ as follows: We first

use the balancing result from [7] or [8] (which bound, for some constant c, each $\mathrm{ord}(A)$ by $c \cdot \log |\mathfrak{D}(A)|$ or by $c \cdot \log |\mathbf{D}|$, respectively). Afterwards, we apply the concatenation lemma (Lemma 4.5) to all rules in a bottom-up fashion, and while doing so we ensure that the non-terminals of the newly generated SLP $\mathcal{S}'$ are disjoint from the non-terminals already present in $\mathcal{S}_{\mathrm{DDB}}$. Finally, we extend $\mathcal{S}_{\mathrm{DDB}}$ by simply adding the non-terminals of $\mathcal{S}'$ and their associated rules.

An obvious question is if this constuction can even be carried out in time $O(|\mathcal{S}|)$, avoiding the factor $\log |\mathbf{D}|$. The answer is "no", as shown in [7].

In case that the new document $\mathbf{D}$ is not provided in form of an SLP, but given as a plain text, we first compute an SLP $\mathcal{S}$ for $\mathbf{D}$, and afterwards proceed as before. This adds an additive term $t_{compress}(\mathbf{D})$ to the overall running time, where $t_{compress}(\mathbf{D})$ is the running time of computing $\mathcal{S}$ — i.e., of compressing the document $\mathbf{D}$ by an SLP. This task is theoretically well-understood and there also exists a rich toolbox of practical methods solving this task (cf., [2, 3, 9, 10, 20]). In fact, it is justified to assume that $t_{compress}(\mathbf{D})$ is linear or a low-degree polynomial in $|\mathbf{D}|$; and the size of the constructed SLP can be expected to be rather small in most practical cases.

If we want to add several documents $\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_m$ to an SLP-represented document database, or if we want to build the document database $\{\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_m\}$ from scratch, then we can iterate the approach from above. However, in order to achieve a better compression (at the cost of a slightly increased running time), we may, instead, proceed as follows: First, construct the document $\mathbf{D} = \mathbf{D}_1 \cdot \mathbf{D}_2 \cdots \mathbf{D}_m$ and then proceed as described above. The thus obtained SLP $\mathcal{S}$ for $\mathbf{D}$ can afterwards, by using m applications of the extraction lemma (Lemma 4.6), be transformed into an SLP that represents the document database $\mathsf{DDB} = \{\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_m\}$ by suitable non-terminals $A_1, A_2, \ldots, A_m$.

6 QUERYING DOCUMENT DATABASES

This section is devoted to efficient query evaluation on document databases in the presence of complex document editing. The scenario is as follows: Assume we have represented a document database DDB by a strongly balanced SLP $\mathcal{S}$ in normal form, we have available the basic SLP data structure, and this basic SLP data structure has already been enriched to a *query data structure* that supports the efficient evaluation of queries $Q_1, \ldots, Q_k$ that have previously been registered in our document database system. When given a CDE-expression φ, the CDE extension theorem (Theorem 4.3) tells us how to extend the SLP $\mathcal{S}$ and its basic SLP data structure such that the extension contains a new non-terminal A_φ with $\mathfrak{D}(A_\varphi) = \mathrm{eval}(\varphi)$. This section now deals with the question of how to extend the existing *query data structure* to the new non-terminals such that the queries $Q_1, \ldots, Q_k$ can be efficiently evaluated also on the new document described by φ. It turns out that, when building upon the framework of [22], this can be achieved in time linear in the number of new non-terminals, for each of the registered queries $Q_1, \ldots, Q_k$. Recall from Theorem 4.3 that the number of new non-terminals is in $O(|\varphi| \cdot \log \mathbf{d})$ for $\mathbf{d} := |\mathrm{max}_\varphi(\mathsf{DDB})|$.

To present details on this construction, we have to provide background on the particular query model and the approach of [22] for evaluating these queries on SLP-represented documents.

Document spanners. The query model we adopt here is the information extraction framework of *document spanners* [4]. We assume that the reader is familiar with this framework, and we only provide the necessary notations here. Let $\mathbf{D}$ be a document. For every $i, j \in [|\mathbf{D}|{+}1]$ with $i \leqslant j$, the "tuple" $[i, j\rangle$ is called a *span of* $\mathbf{D}$. Such a span $[i, j\rangle$ obviously represents the factor $\mathbf{D}[i, j\rangle$ of $\mathbf{D}$. $\mathrm{Spans}(\mathbf{D})$ denotes the set of all spans of $\mathbf{D}$, and we let $\mathrm{Spans} = \{[i, j\rangle \;:\; i, j \in \mathbb{N}, i \leqslant j\}$.

For a finite set $\mathcal{X}$ of variables, an $(\mathcal{X}, \mathbf{D})$-*tuple* (or just *span-tuple* if $\mathcal{X}$ and $\mathbf{D}$ are clear from the context) is a partial function $\mathcal{X} \to \mathrm{Spans}(\mathbf{D})$. An $(\mathcal{X}, \mathbf{D})$-*relation* is a set of $(\mathcal{X}, \mathbf{D})$-tuples. A *spanner* (*over terminal alphabet* Σ *and variables* $\mathcal{X}$) is a function that maps every document $\mathbf{D} \in \Sigma^+$ to an $(\mathcal{X}, \mathbf{D})$-relation. In this work, we consider *regular* spanners (over Σ and $\mathcal{X}$), i.e., spanners that can be represented by a finite automaton M (cf. [1, 4–6, 14, 18, 21]). Since we want to build upon the approach of [22], we have to use their representation of regular spanners, where M is a DFA or an NFA that accepts a certain language of *subword-marked words*. For describing this, we need some more notation.

For a set $\mathcal{X}$ of variables, we use a special alphabet $\Gamma_{\mathcal{X}} = \{{}^{\mathsf{x}}\triangleright, \triangleleft^{\mathsf{x}} : \mathsf{x} \in \mathcal{X}\}$ of *markers*. An $(\mathcal{X}, \mathbf{D})$-tuple t is represented as a *marker set* $\widehat{t} := \{({}^{\mathsf{x}}\triangleright, i), (\triangleleft^{\mathsf{x}}, j) : \mathsf{x} \in \mathrm{dom}(t),\ t(\mathsf{x}) = [i, j\rangle\}$. A set $\Lambda \subseteq \Gamma_{\mathcal{X}} \times \mathbb{N}$ is a *partial* marker set if it is a subset of some marker set that represents an $(\mathcal{X}, \mathbf{D})$-tuple. For a partial marker set Λ and a document $\mathbf{D}$ with $\max\{\ell : (\sigma, \ell) \in \Lambda\} \leqslant |\mathbf{D}|{+}1$, the *partial subword-marked word* $\mathfrak{m}(\mathbf{D}, \Lambda)$ is obtained by inserting Λ's markers into $\mathbf{D}$, i. e., $\mathfrak{m}(\mathbf{D}, \Lambda) := A_1 b_1 \ldots A_{|\mathbf{D}|} b_{|\mathbf{D}|} A_{|\mathbf{D}|+1}$, where b_i is the letter at position i in $\mathbf{D}$ and $A_i = \{\sigma : (\sigma, i) \in \Lambda\}$. To avoid clutter, we will often omit those A_i where $A_i = \emptyset$. If Λ is a (non-partial) marker set that describes an $(\mathcal{X}, \mathbf{D})$-tuple t, then $\mathfrak{m}(\mathbf{D}, \Lambda)$ is called a *subword-marked word*. For simplicity, we shall also write $\mathfrak{m}(\mathbf{D}, t)$ instead of $\mathfrak{m}(\mathbf{D}, \Lambda)$ in case that Λ is the marker-set representation of t.

Now, a finite automaton M represents the spanner $[\![M]\!]$ defined as follows: for every document $\mathbf{D}$, an $(\mathcal{X}, \mathbf{D})$-tuple t is in $[\![M]\!](\mathbf{D})$ if and only if the subword-marked word $\mathfrak{m}(\mathbf{D}, t)$ is accepted by M. Any NFA or DFA that represents a spanner in this way is called a *spanner automaton*.

Example 6.1. *Let* $\mathcal{X} = \{\mathsf{x}, \mathsf{y}\}$, $\Sigma = \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$ *and* $\mathbf{D} = \mathsf{abcbcaab}$. *Then* t *with* $t(\mathsf{x}) = [2, 6\rangle$ *and* $t(\mathsf{y}) = [6, 8\rangle$ *is an* $(\mathcal{X}, \mathbf{D})$*-tuple (or, in tuple notation,* $t = ([2, 6\rangle, [6, 8\rangle)$*). The marker set representation of* t *is* $\{({}^{\mathsf{x}}\triangleright, 2), (\triangleleft^{\mathsf{x}}, 6), ({}^{\mathsf{y}}\triangleright, 6), (\triangleleft^{\mathsf{y}}, 8)\}$, *and* $\mathfrak{m}(\mathbf{D}, t) = \mathsf{a}\ \{{}^{\mathsf{x}}\triangleright\}\ \mathsf{b}\ \mathsf{c}\ \mathsf{b}\ \mathsf{c}\ \{\triangleleft^{\mathsf{x}}, {}^{\mathsf{y}}\triangleright\}\ \mathsf{a}\ \mathsf{a}\ \{\triangleleft^{\mathsf{y}}\}\ \mathsf{b}$.

Figure 5 shows a regular spanner (over Σ *and* $\mathcal{X}$*) represented by a* DFA M*. For example,* M *accepts* $\mathsf{abcbc}\{{}^{\mathsf{x}}\triangleright\}\mathsf{aa}\{\triangleleft^{\mathsf{x}}\}\mathsf{b}$, *which means that* $([6, 8\rangle, \bot) \in [\![M]\!](\mathbf{D})$. *Analogously,* $([1, 3\rangle, \bot) \in [\![M]\!](\mathbf{D})$ *and* $(\bot, [5, 6\rangle) \in [\![M]\!](\mathbf{D})$.

Spanner evaluation on SLP-represented documents. We focus on the following query evaluation task: Let M be a spanner automaton. Enumerate, without repetitions, and with a guaranteed bound on the maximum delay, all span-tuples in $[\![M]\!](\mathfrak{D}(A))$, where A is a non-terminal of the SLP that represents our document database. In [22] it has been shown that this problem can be solved with a delay of $O(\log |\mathfrak{D}(A)|)$ after a preprocessing phase that takes time linear in the size of the SLP. This section's aim is to lift this approach to the setting of complex document editing. To do so, we

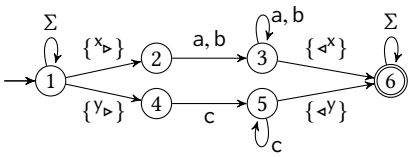


Figure 5: A DFA that represents a spanner over alphabet {a, b, c} and variables {x, y} (1 is the initial state and 6 is the only accepting state).

first have to recall the auxiliary data structures that are computed in the preprocessing phase of the algorithms of [22]. Fortunately, we only need to argue that these auxiliary data structures can be extended when extending the SLP, so that the results of [22] still apply. Thus, we shall present these data structures on an intuitive level and defer more detailed definitions and technical considerations to the paper's full version.

Let M be a spanner automaton with state space $Q = \{1, 2, \ldots, q\}$, initial state 1, accepting states F and transition function δ. For every non-terminal A of the SLP $\mathcal{S}$ that represents our document database, we define $(q \times q)$-matrices $\mathbf{T}_A^M$, $\mathbf{R}_A^M$ and, if A is an inner non-terminal, $\mathbf{I}_A^M$. These matrices are called A's *tuple matrix*, A's *reachability matrix*, and A's *intermediate states matrix*, respectively. For every pair i, j of states of M, $\mathbf{T}_A^M[i, j]$ contains all marker sets Λ such that M can go from state i to state j by reading the word $\mathfrak{D}(A)$ with the markers of Λ inserted, i.e., the word $\mathfrak{m}(\mathfrak{D}(A), \Lambda)$. Note that this means that $\mathbf{T}_A^M$ contains a full representation of the entire query result $[\![M]\!](\mathfrak{D}(A))$ — we will neither have time nor space to precompute and store all this explicitly. But we do have time and space available for computing and storing the matrices $\mathbf{R}_A^M$ and $\mathbf{I}_A^M$, which are defined as follows.

$\mathbf{R}_A^M[i, j]$ contains one of three possible flags that indicate whether $\mathbf{T}_A^M[i, j]$ is the empty set, whether $\mathbf{T}_A^M[i, j]$ only contains the empty marker set (meaning that we can read $\mathfrak{D}(A)$ between states i and j, but only without any markers), or whether neither of these two apply. For an inner non-terminal A with rule $A \to BC$, we define $\mathbf{I}_A^M[i, j]$ to consist of all the states k such that M can read some marked version of $\mathfrak{D}(B)$ between states i and k, and some marked version of $\mathfrak{D}(C)$ between states k and j.

As noted above, we will avoid to precompute and store the tuple matrices $\mathbf{T}_A^M$ for all the non-terminals A of the SLP — we compute and store them only for the *leaf non-terminals*, i.e., the non-terminals T_x for $x \in \Sigma$ whose rule is $T_x \to x$.

We are now ready to precisely define the *query data structure* mentioned at the beginning of this section. Let $\mathcal{S}$ be an SLP in normal form that represents a document database DDB, and let $M_1, \ldots, M_k$ be deterministic spanner automata. When speaking of the *query data structure for $\mathcal{S}$ and $M_1, \ldots, M_k$* we mean the basic SLP data structure of $\mathcal{S}$, enriched by the following auxiliary information: for each $M \in \{M_1, \ldots, M_k\}$ and every

- non-terminal A of $\mathcal{S}$ we have stored the matrix $\mathbf{R}_A^M$,
- inner non-terminal A of $\mathcal{S}$ we have stored the matrix $\mathbf{I}_A^M$,
- leaf non-terminal T_x (for $x \in \Sigma$) we have stored the matrix $\mathbf{T}_{T_x}^M$.

Using these notions, the main result of [22] is as follows (in terms of data complexity, i.e., treating the size of the spanner automaton as a constant).

Theorem 6.2 ([22]). *Given an* SLP $\mathcal{S}$ *and given a deterministic spanner automaton M, the query data structure for $\mathcal{S}$ and M can be computed in time $O(|\mathcal{S}|)$.*

Furthermore, given the query data structure for $\mathcal{S}$ and M, and given a non-terminal A of $\mathcal{S}$, the query result $[\![M]\!](\mathfrak{D}(A))$ can be enumerated with delay $O(\mathrm{ord}(A))$.

If $\mathcal{S}$ is an arbitrary SLP in normal form, then the delay bound $\mathrm{O}(\mathrm{ord}(A))$ can in the worst case be of the same order as the total size of the SLP. This problem was handled in [22] by ensuring that A is c-shallow for some constant c, since then $\mathrm{ord}(A)$ is in $O(\log |\mathfrak{D}(A)|)$. As already discussed in Section 5, for single rooted SLPs with root S_0 it is known from [7, 8] that c-shallowness of S_0 can be achieved in time linear in the size of the SLP — and this was used in the preprocessing phase of [22]. In the present paper, we enforce that the SLP is *strongly balanced*, which also implies that every non-terminal of the SLP is c-shallow (cf., Lemma 2.1). Moreover, as demonstrated in Section 4, the property of being strongly balanced can be maintained by our updates based on complex document editing (note that c-shallowness for all non-terminals can also be achieved in linear time by the result of [7], but it seems rather difficult to maintain this property upon extensions of the SLP).

Spanner evaluation in the presence of complex document editing. The next lemma shows how to update the query data structure for $\mathcal{S}$ and $M_1, \ldots, M_k$ in case that $\mathcal{S}$ is extended by adding a number of new non-terminals $\widetilde{N}$ along with their associated rules. The runtime statement, again, is in terms of data complexity, i.e., treating the size of the spanner automaton as a constant.

Lemma 6.3. *Let $\mathcal{S}$ be an* SLP *in normal form and let M be a deterministic spanner automaton such that the query data structure for $\mathcal{S}$ and M has already been computed. Let $\mathcal{S}'$ be an extension of $\mathcal{S}$ by a set $\widetilde{N}$ of new non-terminals (and their rules). Within time $O(|\widetilde{N}|)$ we can extend the existing query data structure into the query data structure for $\mathcal{S}'$ and M.*

The proof uses the methods of [22] and visits the new non-terminals (and only the new non-terminals) in a bottom-up fashion according to the $\mathcal{S}'$-DAG (i.e., the lower their order in $\mathcal{S}'$ is, the earlier they are visited). Details will be provided in the paper's full version.

Combining this with the CDE extension theorem (Theorem 4.3) and with Theorem 6.2, we obtain this section's main result:

Theorem 6.4. *Let* $\mathsf{DDB} = \{\mathbf{D}_1, \mathbf{D}_2, \ldots, \mathbf{D}_m\}$ *be a document database that is represented by a strongly balanced* SLP $\mathcal{S}$ *in normal form. Let $M_1, \ldots, M_k$ be deterministic spanner automata. When given the query data structure for $\mathcal{S}$ and $M_1, \ldots, M_k$ and a CDE-expression φ over* DDB*, we can construct a strongly balanced extension $\mathcal{S}'$ of $\mathcal{S}$ and the query data structure for $\mathcal{S}'$ and $M_1, \ldots, M_k$, and a new non-terminal $\widetilde{A}$ of $\mathcal{S}'$, such that $\mathfrak{D}(\widetilde{A}) = \mathsf{eval}(\varphi)$. This construction takes time $\mathrm{O}(k \cdot |\varphi| \cdot \log \mathbf{d})$ for $\mathbf{d} := |\mathsf{max}_\varphi(\mathsf{DDB})|$.*

Afterwards, upon input of any $\mathbf{D} \in \mathsf{docs}(\mathcal{S}')$ (represented by a non-terminal of $\mathcal{S}'$) and any $i \in [k]$, the query result $[\![M_i]\!](\mathbf{D})$ can be enumerated with delay $O(\log |\mathbf{D}|)$.

7 FINAL REMARKS

We conclude with an outlook on implications and possible extensions of this work.

A note on extraction and further manipulation of the actual data. In the information extraction framework of [4], regular spanners are employed as basic extractors of span-relations, which are then further queried or manipulated by a *spanner algebra* (see also [19]). In this regard, the *string equality selection* plays a prominent role, which, for given columns x and y of a span-relation, selects those tuples whose x-span $[i_x, j_x\rangle$ and y-span $[i_y, j_y\rangle$ satisfy $\mathbf{D}[i_x, j_x\rangle = \mathbf{D}[i_y, j_y\rangle$. In the SLP-compressed setting it is not entirely obvious how the actual data $\mathbf{D}[i, j\rangle$ for a given span $[i, j\rangle$ can be accessed. Of course, for the non-terminal A that describes $\mathbf{D}$, we could construct the entire document $\mathfrak{D}(A)[i, j\rangle$ as a plain text in time $O((j-i)\cdot \log|\mathfrak{D}(A)|)$, or just access the symbol on position i of $\mathfrak{D}(A)$ in time $O(\log|\mathfrak{D}(A)|)$. However, the real asset of the SLP-compressed setting is that for any span $[i, j\rangle$, by one application of the extraction lemma (Lemma 4.6) we can also get in time $O(\log|\mathfrak{D}(A)|)$ a strongly balanced SLP-representation for $\mathfrak{D}(A)[i, j\rangle$. Moreover, if we are enumerating $[\![M]\!](\mathfrak{D}(A))$ with delay $O(\log|\mathfrak{D}(A)|)$, then the application of the extraction lemma is subsumed by the delay, which means that we can assume that for every individual span we are also provided with a non-terminal that derives exactly this span. Since one can decide whether $\mathfrak{D}(A) = \mathfrak{D}(B)$ in time polynomial in the compressed size $|\mathcal{S}|$, instead of the uncompressed size $|\mathfrak{D}(A)|$ and $|\mathfrak{D}(B)|$ (see [12, Section 5]), we can even use the spanner algebra without explicitly decompressing any SLP-represented data.

A note on other evaluation problems. In this paper, we focus on regular spanners as query class, and on the enumeration problem in particular. However, our approach of complex document editing is also applicable to other evaluation tasks, including testing (i. e., checking whether $t \in [\![M]\!](\mathbf{D})$ for a given span-tuple t) and non-emptiness (i. e., checking whether $[\![M]\!](\mathbf{D}) \neq \emptyset$). Precise results concerning these evaluation tasks in the setting of complex document editing will be incorporated in this paper's full version.

A note on a more general kind of queries. The spanner automata considered as query class in this paper can be viewed as finite automata that read an input over Σ and just insert some meta-symbols (i. e., sets of markers) between symbols of this input. Hence, spanner automata are essentially finite transducers. It is not difficult to see that our results also extend to arbitrary finite transducers, i.e., to the more general query class considered in [15].

ACKNOWLEDGMENTS

The first author is supported by the German Research Foundation — project number 416776735 (gefördert durch die Deutsche Forschungsgemeinschaft (DFG) – Projektnummer 416776735).
The second author is partially supported by the ANR project EQUUS ANR-19-CE48-0019; funded by the German Research Foundation — project number 431183758 (gefördert durch die Deutsche Forschungsgemeinschaft (DFG) – Projektnummer 431183758).
The authors thank the anonymous referees for their careful evaluation and feedback.

REFERENCES

[1] Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. 2021. Constant-Delay Enumeration for Nondeterministic Document Spanners. *ACM Trans. Database Syst.* 46, 1 (2021), 2:1–2:30. https://doi.org/10.1145/3436487
[2] K. Casel, H. Fernau, S. Gaspers, B. Gras, and M.L. Schmid. 2020. On the Complexity of the Smallest Grammar Problem over Fixed Alphabets. *Theory of Computing Systems* (2020). https://doi.org/10.1007/s00224-020-10013-w
[3] M. Charikar, E. Lehman, D. Liu, R. Panigrahy, M. Prabhakaran, A. Sahai, and A. Shelat. 2005. The smallest grammar problem. *IEEE Transactions on Information Theory* 51, 7 (2005), 2554–2576.
[4] R. Fagin, B. Kimelfeld, F. Reiss, and S. Vansummeren. 2015. Document Spanners: A Formal Approach to Information Extraction. *J. ACM* 62, 2 (2015), 12:1–12:51.
[5] Fernando Florenzano, Cristian Riveros, Martín Ugarte, Stijn Vansummeren, and Domagoj Vrgoc. 2020. Efficient Enumeration Algorithms for Regular Document Spanners. *ACM Trans. Database Syst.* 45, 1 (2020), 3:1–3:42. https://doi.org/10.1145/3351451
[6] D. Freydenberger, B. Kimelfeld, and L. Peterfreund. 2018. Joining Extractions of Regular Expressions. In *Proc. PODS'18*. 137–149.
[7] Moses Ganardi. 2021. Compression by Contracting Straight-Line Programs. In *29th Annual European Symposium on Algorithms, ESA 2021, September 6-8, 2021, Lisbon, Portugal (Virtual Conference)*. 45:1–45:16. https://doi.org/10.4230/LIPIcs.ESA.2021.45
[8] Moses Ganardi, Artur Jez, and Markus Lohrey. 2021. Balancing Straight-line Programs. *J. ACM* 68, 4 (2021), 27:1–27:40. https://doi.org/10.1145/3457389
[9] K. Goto, S. Maruyama, S. Inenaga, H. Bannai, H. Sakamoto, and M. Takeda. 2011. Restructuring Compressed Texts without Explicit Decompression. *CoRR* abs/1107.2729 (2011). http://arxiv.org/abs/1107.2729
[10] Artur Jez. 2015. Approximation of grammar-based compression via recompression. *Theor. Comput. Sci.* 592 (2015), 115–134. https://doi.org/10.1016/j.tcs.2015.05.027
[11] J. C. Kieffer and E.-H. Yang. 2000. Grammar-based codes: A new class of universal lossless source codes. *IEEE Trans. on Information Theory* 46, 3 (2000), 737–754.
[12] M. Lohrey. 2012. Algorithmics on SLP-compressed strings: A survey. *Groups Complex. Cryptol.* 4, 2 (2012), 241–299. https://doi.org/10.1515/gcc-2012-0016
[13] M. Lohrey. 2014. *The Compressed Word Problem for Groups* (Springer Briefs in Mathematics ed.). Springer.
[14] F. Maturana, C. Riveros, and D. Vrgoc. 2018. Document Spanners for Extracting Incomplete Information: Expressiveness and Complexity. In *Proc. PODS'18*.
[15] Martin Muñoz and Cristian Riveros. 2020. Constant-delay enumeration algorithms for document spanners over nested documents. *CoRR* abs/2010.06037 (2020). arXiv:2010.06037 https://arxiv.org/abs/2010.06037
[16] C. Nevill-Manning and I. Witten. 1997. Identifying Hierarchical Structure in Sequences: A linear-time algorithm. *J. Artif. Intelligence Research* 7 (1997), 67–82.
[17] C. G. Nevill-Manning. 1996. *Inferring Sequential Structure*. Ph.D. Dissertation. University of Waikato, NZ.
[18] L. Peterfreund. 2019. *The Complexity of Relational Queries over Extractions from Text*. Ph.D. Dissertation.
[19] Liat Peterfreund, Dominik D. Freydenberger, Benny Kimelfeld, and Markus Kröll. 2019. Complexity Bounds for Relational Algebra over Document Spanners. In *Proceedings of the 38th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2019, Amsterdam, The Netherlands, June 30 - July 5, 2019*. 320–334.
[20] W. Rytter. 2003. Application of Lempel-Ziv factorization to the approximation of grammar-based compression. *Theor. Comput. Sci.* 302, 1-3 (2003), 211–222.
[21] Markus L. Schmid and Nicole Schweikardt. 2021. A Purely Regular Approach to Non-Regular Core Spanners. In *24th International Conference on Database Theory, ICDT 2021, March 23-26, 2021, Nicosia, Cyprus*. 4:1–4:19. https://doi.org/10.4230/LIPIcs.ICDT.2021.4
[22] Markus L. Schmid and Nicole Schweikardt. 2021. Spanner Evaluation over SLP-Compressed Documents. In *PODS'21: Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Virtual Event, China, June 20-25, 2021*. 153–165. https://doi.org/10.1145/3452021.3458325
[23] J. A. Storer and T. G. Szymanski. 1982. Data compression via textual substitution. *Journal of the ACM* 29, 4 (1982), 928–951.

APPENDIX

A PROOF OF LEMMA 2.1

Proof of Lemma 2.1. Let $k := \mathrm{ord}(A)-1$. We represent the subgraph of the $\mathcal{S}$-DAG spanned by A as a binary tree: the tree's root is labelled by A, each node labelled by an inner non-terminal A' has a left child labelled by $\mathrm{lc}(A')$ and a right child labelled by $\mathrm{rc}(A')$, and nodes labelled by leaf non-terminals are leaves of the tree. Note

that this tree has height k, and it has exactly $|\mathfrak{D}(A)|$ leaves. This implies that $|\mathfrak{D}(A)| \leqslant 2^k$ and therefore $\log|\mathfrak{D}(A)| \leqslant k = \mathrm{ord}(A){-}1$.

Since A is strongly balanced, we know that on any path from A to some leaf, the order can decrease from a node to its direct successor by at most 2, and hence any such path has length at least $\lceil \frac{k}{2} \rceil$. Since $\frac{1}{2}\log|\mathfrak{D}(A)| \leqslant \frac{1}{2}k$, this means that any path from A to some leaf has length at least $\frac{1}{2}\log|\mathfrak{D}(A)|$.

In order to finish the lemma's proof, it only remains to show that $\mathrm{ord}(A){-}1 \leqslant 2{\cdot}\log|\mathfrak{D}(A)|$ holds for all strongly balanced non-terminals A. We prove this by induction on $k := \mathrm{ord}(A){-}1$.

For the induction base, $k = 0$ and A is a leaf non-terminal. Hence, $|\mathfrak{D}(A)| = 1$ and $2{\cdot}\log|\mathfrak{D}(A)| = 0 = k$.

For the induction step, A is an inner non-terminal with rule $A \to BC$, and the induction hypothesis holds for B and C. I.e., for each $D \in \{B, C\}$ we have $\mathrm{ord}(D){-}1 \leqslant 2{\cdot}\log|\mathfrak{D}(D)|$, i.e., $|\mathfrak{D}(D)|^2 \geqslant 2^{\mathrm{ord}(D)-1}$. Let $k := \mathrm{ord}(A){-}1$. Since A is balanced, one of its two children B, C has order k and the other one has order either k or $k{-}1$. Thus, $|\mathfrak{D}(A)|^2 = \big(|\mathfrak{D}(B)| + |\mathfrak{D}(C)|\big)^2 = |\mathfrak{D}(B)|^2 + |\mathfrak{D}(C)|^2 + 2{\cdot}|\mathfrak{D}(B)|{\cdot}|\mathfrak{D}(C)| \geqslant 2^{k-1}{+}2^{k-2}{+}2{\cdot}\sqrt{2^{k-2}}{\cdot}\sqrt{2^{k-1}} \geqslant 2^{k-1}{+}2^{k-1} = 2^k$. Hence, $2{\cdot}\log|\mathfrak{D}(A)| \geqslant k = \mathrm{ord}(A){-}1$. □

B DETAILS ABOUT CDE-EXPRESSIONS

We first define CDE-expressions over a document database DDB in more detail. For every $\mathbf{D}_1, \mathbf{D}_2 \in$ DDB, all positions i, j of $\mathbf{D}_1$, and every gap k of $\mathbf{D}_1$, each $\varphi \in \{\mathrm{concat}(\mathbf{D}_1, \mathbf{D}_2), \mathrm{extract}(\mathbf{D}_1, i, j), \mathrm{delete}(\mathbf{D}_1, i, j), \mathrm{insert}(\mathbf{D}_1, \mathbf{D}_2, k), \mathrm{copy}(\mathbf{D}_1, i, j, k)\}$ is a CDE-expression over DDB; the *value* $\mathrm{eval}(\varphi)$ of φ is a document defined by the equalities shown in Figure 2. Moreover, for all CDE-expressions φ_1, φ_2 over DDB, all positions i, j of $\mathrm{eval}(\varphi_1)$, and every gap k of $\mathrm{eval}(\varphi_1)$, we let $\mathrm{concat}(\varphi_1, \varphi_2)$, $\mathrm{extract}(\varphi_1, i, j)$, $\mathrm{delete}(\varphi_1, i, j)$, $\mathrm{insert}(\varphi_1, \varphi_2, k)$, and $\mathrm{copy}(\varphi_1, i, j, k)$ be CDE-expressions over DDB with

$$\begin{aligned}
\mathrm{eval}(\mathrm{concat}(\varphi_1, \varphi_2)) &= \mathrm{concat}(\mathrm{eval}(\varphi_1), \mathrm{eval}(\varphi_2))\,,\\
\mathrm{eval}(\mathrm{extract}(\varphi_1, i, j)) &= \mathrm{extract}(\mathrm{eval}(\varphi_1), i, j)\,,\\
\mathrm{eval}(\mathrm{delete}(\varphi_1, i, j)) &= \mathrm{delete}(\mathrm{eval}(\varphi_1), i, j)\,,\\
\mathrm{eval}(\mathrm{insert}(\varphi_1, \varphi_2, k)) &= \mathrm{insert}(\mathrm{eval}(\varphi_1), \mathrm{eval}(\varphi_2), k), \quad \text{and}\\
\mathrm{eval}(\mathrm{copy}(\varphi_1, i, j, k)) &= \mathrm{copy}(\mathrm{eval}(\varphi_1), i, j, k)\,.
\end{aligned}$$

For a CDE-expression φ over a document database DDB, we also consider its syntax tree $\mathcal{T}_\varphi = (N_\varphi, E_\varphi)$, i. e., an ordered tree of maximum degree 2. Every inner node $t \in N_\varphi$ is labelled with (concat) or (insert, k) and has two children, or it is labelled with (extract, i, j), (delete, i, j), or (copy, i, j, k) and has only one child; every leaf is labelled with a document from DDB. We observe that there is a natural bijection between φ's subexpressions and N_φ. For every node $t \in N_\varphi$, we define $\mathbf{D}_{\varphi,t} := \mathrm{eval}(\varphi_t)$, i. e., $\mathbf{D}_{\varphi,t}$ is the document represented by the subexpression of node t. We also define $|\varphi|$ to be the number of $\mathcal{T}_\varphi$'s internal nodes. The *maximum intermediate document size* of φ is defined by $|\max_\varphi(\mathrm{DDB})| = \max\{|\mathbf{D}_{\varphi,t}| : t \in N_\varphi\}$.

Next, we provide a proof of Lemma 4.2.

Proof of Lemma 4.2. Let $\max_{\mathrm{DDB}} := \max\{|\mathbf{D}| : \mathbf{D} \in \mathrm{DDB}\}$. By induction we show that

$$|\mathrm{eval}(\varphi)| \quad\leqslant\quad 2^{|\varphi|} \cdot \max_{\mathrm{DDB}} \tag{2}$$

holds for every CDE-expression φ. Note that the lemma's statement is an immediate consequence of this.

First, we observe that (2) holds if φ is a single operation of the CDE-algebra: In this case, we have $|\varphi| = 1$. Moreover, if $\varphi = \mathrm{extract}(\mathbf{D}, i, j)$ or $\varphi = \mathrm{delete}(\mathbf{D}, i, j)$, then $|\mathrm{eval}(\varphi)| \leqslant |\mathbf{D}| \leqslant \max_{\mathrm{DDB}}$. If $\varphi = \mathrm{concat}(\mathbf{D}, \mathbf{D}')$ or $\varphi = \mathrm{insert}(\mathbf{D}, \mathbf{D}', k)$, then $|\mathrm{eval}(\varphi)| \leqslant |\mathbf{D}| + |\mathbf{D}'| \leqslant 2\max_{\mathrm{DDB}}$. If $\varphi = \mathrm{copy}(\mathbf{D}, i, j, k)$, then $|\mathrm{eval}(\varphi)| \leqslant 2|\mathbf{D}| \leqslant 2\max_{\mathrm{DDB}}$.

Next, we consider the induction step. If $\varphi = \mathrm{extract}(\psi, i, j)$ or $\varphi = \mathrm{delete}(\psi, i, j)$, then

$$|\mathrm{eval}(\varphi)| \leqslant |\mathrm{eval}(\psi)| \leqslant 2^{|\psi|} \cdot \max_{\mathrm{DDB}} \leqslant 2^{|\varphi|} \cdot \max_{\mathrm{DDB}}\,.$$

If $\varphi = \mathrm{concat}(\psi, \psi')$ or $\varphi = \mathrm{insert}(\psi, \psi', k)$, then

$$\begin{aligned}
|\mathrm{eval}(\varphi)| &= |\mathrm{eval}(\psi)| + |\mathrm{eval}(\psi')|\\
&\leqslant 2^{|\psi|} \cdot \max_{\mathrm{DDB}} + 2^{|\psi'|} \cdot \max_{\mathrm{DDB}}\\
&= \max_{\mathrm{DDB}} \cdot (2^{|\psi|} + 2^{|\psi'|})\\
&\leqslant \max_{\mathrm{DDB}} \cdot (2^{|\varphi|})\,.
\end{aligned}$$

If $\varphi = \mathrm{copy}(\psi, i, j, k)$, then

$$\begin{aligned}
|\mathrm{eval}(\varphi)| &\leqslant 2 \cdot |\mathrm{eval}(\psi)|\\
&\leqslant 2 \cdot 2^{|\psi|} \cdot \max_{\mathrm{DDB}}\\
&= 2^{|\psi|+1} \cdot \max_{\mathrm{DDB}}\\
&= 2^{|\varphi|} \cdot \max_{\mathrm{DDB}}\,.
\end{aligned}$$

□

C THE CDE-ALGEBRA LEMMAS FOR THE OPERATIONS DELETE, INSERT, AND COPY

For completeness, we state here the lemmas for the CDE-algebra operations delete, insert and copy. Since these operations can be expressed by the operations concat and extract, the corresponding lemmas can easily be proven by using the Lemmas 4.5 and 4.6.

Lemma C.1 (deletion lemma). *Given the basic SLP data structure of an* SLP $\mathcal{S} = (N, \Sigma, R)$ *in normal form, a strongly balanced* $A \in N$, *and* $i, j \in \{1, 2, \ldots, |\mathfrak{D}_\mathcal{S}(A)|\}$ *with* $i \leqslant j$, *we can compute in time* $\mathrm{O}(\mathrm{ord}(A))$ *an extension* $\mathcal{S}' = (N \cup \widetilde{N}, \Sigma, R')$ *of* $\mathcal{S}$, *along with its basic SLP data structure, and an* $\widetilde{A} \in \widetilde{N}$, *such that* $\mathfrak{D}_{\mathcal{S}'}(\widetilde{A}) = \mathrm{delete}(\mathfrak{D}(A), i, j)$. *Furthermore, every* $\widetilde{X} \in \widetilde{N}$ *is strongly balanced in* $\mathcal{S}'$.

Lemma C.2 (insertion lemma). *Given the basic SLP data structure of an* SLP $\mathcal{S} = (N, \Sigma, R)$ *in normal form, strongly balanced* $A, B \in N$, *and a* $k \in \{1, 2, \ldots, |\mathfrak{D}_\mathcal{S}(A)|{+}1\}$, *we can compute in time* $\mathrm{O}(\max\{\mathrm{ord}(A), \mathrm{ord}(B)\})$ *an extension* $\mathcal{S}' = (N \cup \widetilde{N}, \Sigma, R')$ *of* $\mathcal{S}$, *along with its basic SLP data structure, and an* $\widetilde{A} \in \widetilde{N}$, *such that* $\mathfrak{D}_{\mathcal{S}'}(\widetilde{A}) = \mathrm{insert}(\mathfrak{D}(A), \mathfrak{D}(B), k)$. *Furthermore, every* $\widetilde{X} \in \widetilde{N}$ *is strongly balanced in* $\mathcal{S}'$.

Lemma C.3 (copying lemma). *Given the basic SLP data structure of an* SLP $\mathcal{S} = (N, \Sigma, R)$ *in normal form, a strongly balanced* $A \in N$, *and* $i, j \in \{1, 2, \ldots, |\mathfrak{D}_\mathcal{S}(A)|\}$ *with* $i \leqslant j$, *and a* $k \in \{1, 2, \ldots, |\mathfrak{D}_\mathcal{S}(A)| + 1\}$, *we can compute in time* $\mathrm{O}(\mathrm{ord}(A))$ *an extension* $\mathcal{S}' = (N \cup \widetilde{N}, \Sigma, R')$ *of* $\mathcal{S}$, *along with its basic SLP data structure, and an* $\widetilde{A} \in \widetilde{N}$, *such that* $\mathfrak{D}_{\mathcal{S}'}(\widetilde{A}) = \mathrm{copy}(\mathfrak{D}(A), i, j, k)$. *Furthermore, every* $\widetilde{X} \in \widetilde{N}$ *is strongly balanced in* $\mathcal{S}'$.

By these lemmas, together with Lemma 4.5 and Lemma 4.6, we have a lemma for each of the basic operations of the CDE-algebra. Using this, the proof of Theorem 4.3 is straightforward.