HAL
archives-ouvertes.fr

Grammars for Document Spanners

Liat Peterfreund

To cite this version:

Liat Peterfreund. Grammars for Document Spanners. ICDT 2021 - 24th International Conference on Extending Database Technology, Mar 2021, Nicosia / Virtual, Cyprus. hal-03104144

HAL Id: hal-03104144
https://hal.inria.fr/hal-03104144

Submitted on 8 Jan 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Grammars for Document Spanners

Liat Peterfreund
IRIF - CNRS & Université de Paris, Paris, France
University of Edinburgh, Edinburgh, UK
liatpf.cs@gmail.com

Abstract

We propose a new grammar-based language for defining information-extractors from documents (text) that is built upon the well-studied framework of document spanners for extracting structured data from text. While previously studied formalisms for document spanners are mainly based on regular expressions, we use an extension of context-free grammars, called *extraction grammars*, to define the new class of context-free spanners. Extraction grammars are simply context-free grammars extended with variables that capture interval positions of the document, namely spans. While regular expressions are efficient for tokenizing and tagging, context-free grammars are also efficient for capturing structural properties. Indeed, we show that context-free spanners are strictly more expressive than their regular counterparts. We reason about the expressive power of our new class and present a pushdown-automata model that captures it. We show that extraction grammars can be evaluated with polynomial data complexity. Nevertheless, as the degree of the polynomial depends on the query, we present an enumeration algorithm for unambiguous extraction grammars that, after quintic preprocessing, outputs the results sequentially, without repetitions, with a constant delay between every two consecutive ones.

2012 ACM Subject Classification Information systems → Relational database model; Information systems → Data model extensions

Keywords and phrases Information Extraction, Document Spanners, Context-free Grammars, Regular Expressions, Pushdown Automata.

Digital Object Identifier 10.4230/LIPIcs...

1 Introduction

The abundance and availability of valuable textual resources in the last decades position text analytics as a standard component in data-driven workflows. One of the core operations that aims to facilitate the analysis and integration of textual content is Information Extraction (IE), the extraction of structured data from text. IE arises in a large variety of domains, including social media analysis [4], health-care analysis [44], customer relationship management [1], information retrieval [46], and more.

Rules have always been a key component in various paradigms for IE, and their roles have varied and evolved over the time. Systems such as Xlog [39] and IBM's SystemT [28, 6] use rules to extract relations from text (e.g., tokenizer, dictionary lookup, and part-of-speech tagger) that are further manipulated with relational query languages. Other systems use rules to generate features for machine-learning classifiers [27, 36].

Document Spanners. The framework of document spanners that was presented by Fagin et al. provides a theoretical basis for investigating the principles of relational rule systems for IE [12]. The research on document spanners has focused on their expressive power [12, 15, 35, 18, 33, 20] their computational complexity [2, 14, 19, 34], incompleteness [30, 34], and other system aspects such as cleaning [13], dynamic complexity [21], distributed query planning [8] and an annotated variant [9].

In the documents spanners framework, a *document* $\mathbf{d}$ is a string over a fixed finite alphabet, and a *spanner* is a function that extracts from a document a relation over the spans of $\mathbf{d}$. A *span* x is a half-open interval of positions of $\mathbf{d}$ and it represents a substring

© Liat Peterfreund;
licensed under Creative Commons License CC-BY

x	y
$[1,2\rangle$	$[2,3\rangle$
$[3,4\rangle$	$[4,5\rangle$
$[3,4\rangle$	$[4,6\rangle$

Figure 1 Extracted relation

$$S \to B \vdash_x \mathtt{a}A\mathtt{b} \dashv_y B$$
$$A \to \mathtt{a}A\mathtt{b} \;\big|\; \dashv_x\vdash_y$$
$$B \to \mathtt{a}B \;\big|\; \mathtt{b}B \;\big|\; \epsilon$$

Figure 2 Production rules

$\vdash_x \mathtt{aa} \dashv_x\vdash_y \mathtt{bb} \dashv_y \mathtt{b}$
$\mathtt{aa} \vdash_x \mathtt{aa} \dashv_x\vdash_y \mathtt{bb} \dashv_y \mathtt{b}$
$\mathtt{aa} \vdash_x \mathtt{a} \dashv_x\vdash_y \mathtt{b} \dashv_y \mathtt{b}$

Figure 3 Ref-words

$\mathbf{d}_x$ of $\mathbf{d}$ that is identified by these positions. A natural way to specify a spanner is by a *regex formula*: a regular expression with embedded *capture variables* that are viewed as relational attributes. For instance, the spanner that is given by the regex formula $(\mathtt{a} \vee \mathtt{b})^* \vdash_x \mathtt{aa}^* \dashv_x\vdash_y \mathtt{bb}^* \dashv_y (\mathtt{a} \vee \mathtt{b})^*$ extracts from documents spans x and y that correspond, respectively, with a non-empty substring of $\mathtt{a}$'s followed by a non-empty substring of $\mathtt{b}$'s. In particular, it extracts from the document $\mathtt{ababb}$ the relation depicted in Figure 1.

The class of *regular spanners* is the class of spanners definable as the closure of regex formulas under positive relational algebra operations: projection, natural join and union. The class of regular spanners can be represented alternatively by finite state machines, namely *variable-set automata (vset-automata)*, which are nondeterministic finite-state automata that can open and close variables (that, as in the case of regex formulas, play the role of the attributes of the extracted relation). *Core* spanners [12] are obtained by extending the class of regular spanners with string-equality selection on span variables. Although core spanners can express strictly more than regular spanners, they are still quite limited as, e.g., there is no core spanner that extracts all pairs x and y of spans having the same *length* [12].

To date, most research on spanners has been focused on the regular representation, that is, regular expressions and finite state automata. While regular expressions are useful for segmentation and tokenization, they are not useful in describing complex nested structures (e.g., syntactic structure of a natural language sentence) and relations between different parts of the text. Regular languages also fall short in dealing with tasks such as syntax highlighting [31] and finding patterns in source code [40]. For all of the above mentioned tasks we have context-free grammars. It is well known that context-free languages are strictly more expressive than regular languages. While Büchi [5] has showed that regular languages are equivalent to monadic second order logic (over strings), Lautemann et al. [26] have showed that adding an existential quantification over a binary relation that is interpreted as a matching is enough to express all context-free languages. This quantification, intuitively, is what makes it possible to also express structural properties.

Contribution. In this work we propose a new grammar-based approach for defining the class of *context-free spanners*. Context-free spanners are defined with *extraction grammars* which, like regex formulas, incorporate *capture variables* that are viewed as relational attributes. Extraction grammars produce *ref-words* which are words over an extended alphabet that consists of standard terminal symbols along with *variable operations* that denote opening and closing of variables. The result of evaluating an extraction grammar on a document $\mathbf{d}$ is defined via the ref-words that are produced by the grammar and equal to $\mathbf{d}$ after erasing the variable operations. For example, the extraction grammar whose production rules appear in Figure 2 produces the ref-words $\vdash_x \mathtt{a} \dashv_x\vdash_y \mathtt{b} \dashv_y \mathtt{abb}$ and $\mathtt{ab} \vdash_x \mathtt{a} \dashv_x\vdash_y \mathtt{b} \dashv_y \mathtt{b}$, and more. Hence, it extracts from $\mathbf{d} := \mathtt{ababb}$ the two first tuples from the relation in Figure 1. In Figure 3 there are additional examples of ref-words produced by this grammar. In general, the given grammar extracts from documents the spans x and y that correspond, respectively, with a non-empty substring of $\mathtt{a}$'s followed by an equal-length substring of $\mathtt{b}$'s. With a slight adaptation of Fagin et al. inexpressibility proof [12, Theorem 4.21], it can show that this spanner is inexpressible by core spanners.

Indeed, we show that context-free spanners are strictly more expressive than regular spanners and that the restricted class of regular extraction grammars captures the regular spanners. We compare the expressiveness of context-free spanners against core and generalized core spanners and show that context-free spanners are incomparable to any of these classes. In addition to extraction grammars, we present a pushdown automata model that captures the context-free spanners.

In term of evaluation of context-free spanners we can evaluate extraction grammars in polynomial time in *data complexity*, where the spanner is regarded as fixed and the document as input. However, as the degree of this polynomial depends on the query (in particular, in the number of variables in the relation it extracts), we propose an enumeration algorithm for unambiguous extraction grammars. Our algorithm outputs the results consecutively, after quintic preprocessing, with constant delay between every two answers. In the first step of the preprocessing stage we manipulate the extraction grammar so that it will be adjusted to a specific document. In the second step of the preprocessing we change it in a way that its non-terminals include extra information on the variable operations. This extra information enables us to skip sequences of productions that do not affect the output, hence obtaining a delay that is independent of the input document and linear in the number of variables associated with our spanner.

Related Work. Grammar-based parsers are widely used in IE systems [45, 38]. There are, as well, several theoretical frameworks that use grammars for IE, one of which is Knuth's framework of attribute grammars [24, 25]. In this framework, the non-terminals of a grammar are attached with attributes[1] that pass semantic information up and down a parse tree. While both extraction grammars and attribute grammars extract information via grammars, it seems as if the expressiveness of these formalisms is incomparable to extraction grammars.

The problem of enumerating words of context-free grammars arises in different contexts [42, 32]. Providing complexity guarantees on the enumeration is usually tricky and requires assumptions either on the grammar or on the output. Mäkinen [29] has presented an enumeration algorithm for regular grammars and for unambiguous context-free grammars with additional restrictions (strongly prefix-free and length complete). Later, Dömösi [10] has presented an enumeration algorithm for unambiguous context-free grammars that outputs, with quadratic delay, only the words of a fixed length.

Organization. In Section 2, we present extraction grammars, their semantics and extraction pushdown automata. In Section 3, we shortly discuss the expressive power of context-free spanners and their evaluation. In Sections 4 and 5, we present our enumeration algorithm, and in Section 6 we conclude.

2 Context-Free Spanners

In this section we present the class of context-free spanners by presenting two representation systems: extraction grammars and extraction pushdown automata.

[1] The term "attributes" was previously used in the relational context; Here the meaning is different.

2.1 Preliminaries

We start by presenting the formal setup based on notations and definitions used in previous works on document spanners (e.g., [12, 19]).

Strings and Spans. We set an infinite set Vars of variables and fix a finite alphabet Σ that is disjoint of Vars. In what follows we assume that our alphabet Σ consists of at least two letters. A *document* $\mathbf{d}$ is a finite sequence over Σ whose length is denoted by $|\mathbf{d}|$. A *span* identifies a substring of $\mathbf{d}$ by specifying its bounding indices. Formally, if $\mathbf{d} = \sigma_1 \cdots \sigma_n$ where $\sigma_i \in \Sigma$ then a span of $\mathbf{d}$ has the form $[i, j\rangle$ where $1 \leq i \leq j \leq n+1$ and $\mathbf{d}_{[i,j\rangle}$ denotes the substring $\sigma_i \cdots \sigma_{j-1}$. When $i = j$ it holds that $\mathbf{d}_{[i,j\rangle}$ equals the empty string, which we denote by ϵ. We denote by $\mathsf{Spans}(\mathbf{d})$ the set of all possible spans of a document $\mathbf{d}$.

Document Spanners. Let $X \subseteq \mathsf{Vars}$ be a finite set of variables and let $\mathbf{d}$ be a document. An $(X, \mathbf{d})$-mapping assigns spans of $\mathbf{d}$ to variables in X. An $(X, \mathbf{d})$-relation is a finite set of $(X, \mathbf{d})$-mappings. A *document spanner* (or *spanner*, for short) is a function associated with a finite set X of variables that maps documents $\mathbf{d}$ into $(X, \mathbf{d})$-relations.

2.2 Extraction Grammars

The *variable operations* of a variable $x \in \mathsf{Vars}$ are $\vdash_x$ and $\dashv_x$ where, intuitively, $\vdash_x$ denotes the opening of x, and $\dashv_x$ its closing. For a finite subset $X \subseteq \mathsf{Vars}$, we define the set $\Gamma_X := \{\vdash_x, \dashv_x \mid x \in X\}$. That is, Γ_X is the set that consists of all the variable operations of all variables in X. We assume that Σ and Γ_X are disjoint. We extend the classical definition of context-free grammars [23] by treating the variable operations as special terminal symbols. Formally, a *context-free extraction grammar*, or *extraction grammar* for short, is a tuple $G := (X, V, \Sigma, P, S)$ where

- $X \subseteq \mathsf{Vars}$ is a finite set of variables,
- V is a finite set of *non-terminal* symbols[2],
- Σ is a finite set of *terminal* symbols;
- P is a finite set of *production rules* of the form $A \to \alpha$ where A is a non-terminal and $\alpha \in (V \cup \Sigma \cup \Gamma_X)^*$, and
- S is a designated non-terminal symbol referred to as the *start symbol*.

We say that the extraction grammar G is *associated* with X.

▶ **Example 1.** In this and in the following examples we often denote the elements in V by upper case alphabet letters from the beginning of the English alphabet $(A, B, C, \ldots)$. Let $\Sigma = \{\mathtt{a}, \mathtt{b}\}$, and let us consider the grammar DISJEQLEN associated with the variables $\{x, y\}$ that is given by the following production rules:

- $S \to B \vdash_x A \dashv_y B \mid B \vdash_y A \dashv_x B$
- $A \to \mathtt{a}A\mathtt{a} \mid \mathtt{a}A\mathtt{b} \mid \mathtt{b}A\mathtt{b} \mid \mathtt{b}A\mathtt{a}$
- $A \to \dashv_x B \vdash_y \mid \dashv_y B \vdash_x$
- $B \to \epsilon \mid \mathtt{a}B \mid \mathtt{b}B$

Here and in what follows, we use the compact notation for production rules by writing $A \to \alpha_1 \mid \cdots \mid \alpha_n$ instead of the productions $A \to \alpha_1, \cdots, A \to \alpha_n$. As we shall later see, this grammar extracts pairs of disjoint spans with the same length. ◀

[2] Note that these are often referred to as variables, however, here we use the term 'non-terminals' to distinguish between these symbols and elements in Vars.

While classical context-free grammars generate strings, extraction grammars generate words over the extended alphabet $\Sigma \cup \Gamma_X$. These words are referred to as *ref-words* [37]. Similarly to (classical) context-free grammars, the process of deriving ref-words is defined via the notations $\Rightarrow, \Rightarrow^n, \Rightarrow^*$ that stand for one, n, and several (possibly zero) derivation steps, respectively. To emphasize the grammar being discussed, we sometime use the grammar as a subscript (e.g., $\Rightarrow^*_G$). For the full definitions we refer the reader to Hopcroft et al. [23]. A non-terminal A is called *useful* if there is some derivation of the form $S \Rightarrow^* \alpha A \beta \Rightarrow^* w$ where $w \in (\Sigma \cup \Gamma_X)^*$. If A is not useful then it is called *useless*. For complexity analysis, we define the *size* $|G|$ of an extraction grammar G as the sum of the number of symbols at the right-hand sides (after the $\rightarrow$) of its rules.

2.3 Semantics of Extraction Grammars

Following Freydenberger [16] we define the semantics of extraction grammars using ref-words. A ref-word $\mathbf{r} \in (\Sigma \cup \Gamma_X)^*$ is *valid (for X)* if each variable of X is opened and then closed exactly once, or more formally, for each $x \in X$ the string $\mathbf{r}$ has precisely one occurrence of $\vdash_x$, precisely one occurrence of $\dashv_x$, and the former is before (i.e., to the left of) the latter.

▶ **Example 2.** The ref-word $\mathbf{r}_1 := \vdash_x \texttt{aa} \dashv_y \vdash_x \texttt{ab} \dashv_y$ is not valid for $\{x, y\}$ whereas the ref-words $\mathbf{r}_2 := \vdash_x \texttt{aa} \dashv_x \vdash_y \texttt{ab} \dashv_y$ and $\mathbf{r}_3 := \vdash_y \texttt{a} \dashv_y \vdash_x \texttt{a} \dashv_x \texttt{ab}$ are valid for $\{x, y\}$. ◀

To connect ref-words to terminal strings and later to spanners, we define a morphism $\mathsf{clr}\colon (\Sigma \cup \Gamma_X)^* \rightarrow \Sigma^*$ by $\mathsf{clr}(\sigma) := \sigma$ for $\sigma \in \Sigma$, and $\mathsf{clr}(\tau) := \epsilon$ for $\tau \in \Gamma_X$. For $\mathbf{d} \in \Sigma^*$, let $\mathsf{Ref}(\mathbf{d})$ be the set of all valid ref-words $\mathbf{r} \in (\Sigma \cup \Gamma_X)^*$ with $\mathsf{clr}(\mathbf{r}) = \mathbf{d}$. By definition, every $\mathbf{r} \in \mathsf{Ref}(\mathbf{d})$ has a unique factorization $\mathbf{r} = \mathbf{r}'_x \cdot \vdash_x \cdot \mathbf{r}_x \cdot \dashv_x \cdot \mathbf{r}''_x$ for each $x \in X$. With these factorizations, we interpret $\mathbf{r}$ as a $(X, \mathbf{d})$-mapping $\mu^{\mathbf{r}}$ by defining $\mu^{\mathbf{r}}(x) := [i, j\rangle$, where $i := |\mathsf{clr}(\mathbf{r}'_x)| + 1$ and $j := i + |\mathsf{clr}(\mathbf{r}_x)|$. An alternative way of understanding $\mu^{\mathbf{r}} = [i, j\rangle$ is that i is chosen such that $\vdash_x$ occurs between the positions in $\mathbf{r}$ that are mapped to σ_{i-1} and σ_i, and $\dashv_x$ occurs between the positions that are mapped to σ_{j-1} and σ_j (assuming that $\mathbf{d} = \sigma_1 \cdots \sigma_{|\mathbf{d}|}$, and slightly abusing the notation to avoid a special distinction for the non-existing positions σ_0 and $\sigma_{|\mathbf{d}|+1}$).

▶ **Example 3.** Let $\mathbf{d} = \texttt{aaab}$. The ref-word $\mathbf{r}_2$ from Example 2 is interpreted as the $(\{x, y\}, \mathbf{d})$-mapping $\mu^{\mathbf{r}_2}$ with $\mu^{\mathbf{r}_2}(x) = [1, 3\rangle$ and $\mu^{\mathbf{r}_2}(y) = [3, 5\rangle$. ◀

Extraction grammars define ref-languages which are sets of ref-words. The ref-language $\mathcal{R}(G)$ of an extraction grammar $G := (X, V, \Sigma, P, S)$ is defined by $\mathcal{R}(G) := \{\mathbf{r} \in (\Sigma \cup \Gamma_X)^* \mid S \Rightarrow^* \mathbf{r}\}$. Note that we use $\mathcal{R}(G)$ instead of $\mathcal{L}(G)$ being used for standard grammars, to emphasize that the produced language is a ref-language. (We also use $\mathcal{L}(G)$ when G is a standard grammar.) To illustrate the definition let us consider the following example.

▶ **Example 4.** Both ref-words $\mathbf{r}_1$ and $\mathbf{r}_2$ from Example 2 are in $\mathcal{R}(\textsc{disjEqLen})$ where $\textsc{disjEqLen}$ is the grammar described in Example 1. Producing both $\mathbf{r}_1$ and $\mathbf{r}_2$ starts similarly with the sequence: $S \Rightarrow B \vdash_x A \dashv_y B \Rightarrow^2 \vdash_x A \dashv_y \Rightarrow \vdash_x \texttt{a}A\texttt{b} \dashv_y \Rightarrow \vdash_x \texttt{aa}A\texttt{ab} \dashv_y$. The derivation of $\mathbf{r}_1$ continues with $\Rightarrow \vdash_x \texttt{aa} \dashv_y B \vdash_x \texttt{ab} \dashv_y \Rightarrow \vdash_x \texttt{aa} \dashv_y \vdash_x \texttt{ab} \dashv_y$ whereas that of $\mathbf{r}_2$ continues with $\Rightarrow \vdash_x \texttt{aa} \dashv_x B \vdash_y \texttt{ab} \dashv_y \Rightarrow \vdash_x \texttt{aa} \dashv_x \vdash_y \texttt{ab} \dashv_y$. ◀

We denote by $\mathsf{Ref}(G)$ the set of all ref-words in $\mathcal{R}(G)$ that are valid for X. Finally, we define the set $\mathsf{Ref}(G, \mathbf{d})$ of ref-words in $\mathsf{Ref}(G)$ that clr maps to $\mathbf{d}$. That is, $\mathsf{Ref}(G, \mathbf{d}) := \mathsf{Ref}(G) \cap \mathsf{Ref}(\mathbf{d})$. The result of evaluating the spanner $[\![G]\!]$ on a document $\mathbf{d}$ is then defined as

$$[\![G]\!](\mathbf{d}) := \{\mu^{\mathbf{r}} \mid \mathbf{r} \in \mathsf{Ref}(G, \mathbf{d})\}.$$

▶ **Example 5.** Let us consider the document $\mathbf{d} := \mathtt{aaba}$. The grammar DISJEQLEN maps $\mathbf{d}$ into a set of $(\{x,y\},\mathbf{d})$-mappings, amongst are $\mu^{\mathbf{r}_2}$ that is defined by $\mu^{\mathbf{r}_2}(x) := [1,3\rangle$ and $\mu^{\mathbf{r}_2}(y) := [3,5\rangle$ and $\mu^{\mathbf{r}_3}$ that is defined by $\mu^{\mathbf{r}_3}(x) := [2,3\rangle$ and $\mu^{\mathbf{r}_3}(y) := [1,2\rangle$. It can be shown that the grammar DISJEQLEN maps every document $\mathbf{d}$ into all possible $(\{x,y\},\mathbf{d})$-mappings μ such that $\mu(x)$ and $\mu(y)$ are disjoint (i.e., do not overlap) and have the same length (i.e., $|\mathbf{d}_{\mu(x)}| = |\mathbf{d}_{\mu(y)}|$). ◀

A spanner S is said to be *definable* by an extraction grammar G if $S(\mathbf{d}) = [\![G]\!](\mathbf{d})$ for every document $\mathbf{d}$.

▶ **Definition 6.** *A* context-free spanner *is a spanner definable by an extraction grammar.*

2.4 Extraction Pushdown Automata

An *extraction pushdown automaton*, or *extraction* PDA, is associated with a finite set $X \subseteq \mathsf{Vars}$ of variables and can be viewed as a standard pushdown automata over the extended alphabet $\Sigma \cup \Gamma_X$. Formally, an *extraction* PDA is a tuple $A := (X, Q, \Sigma, \Delta, \delta, q_0, Z, F)$ where X is a finite set of variables; Q is a finite set of states; Σ is the input alphabet; Δ is a finite set which is called *the stack alphabet*; δ is a mapping $Q \times \left(\Sigma \cup \{\epsilon\} \cup \Gamma_X\right) \times \Delta \to 2^{Q \times \Delta^*}$ which is called the *transition function*; $q_0 \in Q$ is the *initial state*; $Z \in \Delta$ is the *initial stack symbol*; and $F \subseteq Q$ is the set of *accepting states*. Indeed, extraction PDAs run on ref-words (i.e., finite sequences over $\Sigma \cup \Gamma_X$), as opposed to classical PDAs whose input are words (i.e., finite sequences over Σ). Similarly to classical PDAs, the computation of extraction PDAs can be described using sequences of configurations: a *configuration* of A is a triple (q, w, γ) where q is the state, w is the remaining input, and γ is the stack content such that the top of the stack is the left end of γ and its bottom is the right end. We use the notation $\vdash^*$ similarly to how it is used in the context of PDAs [22] and define the ref-language $\mathcal{R}(A)$:

$$\mathcal{R}(A) := \{\mathbf{r} \in (\Sigma \cup \Gamma_X)^* \mid \exists \alpha \in \Delta^*, q_f \in F : (q_0, \mathbf{r}, Z) \vdash^* (q_f, \epsilon, \alpha)\}\,.$$

We denote the language of A by $\mathcal{R}(A)$ to emphasize that it is a ref-language, and denote by $\mathsf{Ref}(A)$ the set of all ref-words in $\mathcal{R}(A)$ that are valid for X. The result of evaluating the spanner $[\![A]\!]$ on a document $\mathbf{d}$ is then defined as

$$[\![A]\!](\mathbf{d}) := \{\mu^{\mathbf{r}} \mid \mathbf{r} \in \mathsf{Ref}(A) \cap \mathsf{Ref}(\mathbf{d})\}\,.$$

▶ **Example 7.** We define the extraction PDA that maps a document $\mathbf{d}$ into the set of $(\{x,y\},\mathbf{d})$-mappings μ where $\mu(x)$ ends before $\mu(y)$ starts and their lengths are the same. The stack alphabet consists of the bottom symbol $\perp$ and C, and the transition function δ is described in Figure 4 where a transition from state q to state q' that is labeled with $\tau, A/\,\gamma$ denotes that the automaton moves from state q to state q' upon reading τ with A at the top of the stack, while replacing A with γ. We can extend the automaton in a symmetric way such that it will represent the same spanner as that represented by the grammar DISJEQLEN from Example 1. ◀

We say that a spanner S is *definable* by an extraction PDA A if for every document $\mathbf{d}$ it holds that $[\![A]\!](\mathbf{d}) = S(\mathbf{d})$. Treating the variable operations as terminal symbols enables us to use the equivalence of PDAs and context-free grammars and conclude the following straightforward observation.

▶ **Proposition 8.** *The class of spanners definable by extraction grammars is equal to the class of spanners definable by extraction PDAs.*

Thus, we have also an automata formalism for defining the context-free spanners.

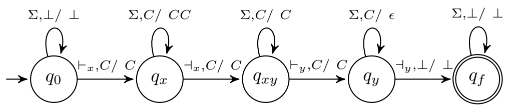

Figure 4 Transition function of Example 7

$$S \to B \vdash_x A_1 \dashv_y B \mid B \vdash_y A_2 \dashv_x B$$
$$A_i \to \mathtt{a}A_i\mathtt{a} \mid \mathtt{a}A_i\mathtt{b} \mid \mathtt{b}A_i\mathtt{b} \mid \mathtt{b}A_i\mathtt{a},\ i = 1, 2$$
$$A_1 \to \dashv_x B \vdash_y\ ,\quad A_2 \to \dashv_y B \vdash_x$$
$$B \to \epsilon \mid \mathtt{a}B \mid \mathtt{b}B$$

Figure 5 Productions of Example 9

2.5 Functional Extraction Grammars

Freydenberger and Holldack [17] have presented the notion of *functionality* in the context of regular spanners. We now extend it to extraction grammars. The intuition is that interpreting an extraction grammar as a spanner disregards ref-words that are not valid. We call an extraction grammar G *functional* if every ref-word in $\mathcal{R}(G)$ is valid.

▶ **Example 9.** The grammar disjEqLen in our running example is not functional. Indeed, we saw in Example 4 that the ref-word $\mathbf{r}_1$, although it is not valid, is in $\mathcal{R}(\text{disjEqLen})$. We can, however, simply modify the grammar to obtain an equivalent functional one. Notice that the problem arises due to the production rules $S \to B \vdash_x A \dashv_y B$ and $S \to B \vdash_y A \dashv_x B$. For variable A we have $A \Rightarrow^* \mathbf{r}_1$ where $\mathbf{r}_1$ contains both $\dashv_x$ and $\vdash_y$, and we also have $A \Rightarrow^* \mathbf{r}_2$ where $\mathbf{r}_2$ contains both $\dashv_y$ and $\vdash_x$. To fix that, we can replace the non-terminal A with two non-terminals, namely A_1 and A_2, and change the production rules so that for every ref-word $\mathbf{r}$ if $A_1 \Rightarrow^* \mathbf{r}$ then $\mathbf{r}$ contains both $\dashv_x$ and $\vdash_y$, and if $A_2 \Rightarrow^* \mathbf{r}$ then $\mathbf{r}$ contains both $\dashv_y$ and $\vdash_x$. It can be shown that the grammar G whose production rules appear in Figure 5 is functional and that $[\![G]\!] = [\![\text{disjEqLen}]\!]$. ◀

▶ **Proposition 10.** *Every extraction grammar G can be converted into an equivalent functional extraction grammar G' in $O(|G|^2 + 3^{2k}|G|)$ time where k is the number of variables G is associated with.*

Inspired by Chomsky's hierarchy, we say that an extraction grammar is in *Chomsky Normal Form (CNF)* if it is in CNF when viewed as a grammar over the extended alphabet $\Sigma \cup \Gamma_X$. We remark that, in Proposition 10, G' is in CNF.

2.6 Unambiguous Extraction Grammars

A grammar G is said to be unambiguous if every word it produces has a unique parse-tree. We extend this definition to extraction grammars. An extraction grammar G is said to be *unambiguous* if for every document $\mathbf{d}$ and every $(X, \mathbf{d})$-mapping $\mu \in [\![G]\!](\mathbf{d})$ it holds that there is a unique ref-word $\mathbf{r}$ for which $\mu^{\mathbf{r}} = \mu$ and this ref-word has a unique parse tree. Unambiguous extraction grammars are less expressive than their ambiguous counterparts as the Boolean case shows (i.e., unambiguous context-free grammars are less expressive than ambiguous context-free grammars [23]).

▶ **Example 11.** The extraction grammar given in Example 9 is not unambiguous since it produces the ref-words $\vdash_x\dashv_x\vdash_y\dashv_y$ and $\vdash_y\dashv_y\vdash_x\dashv_x$ that correspond with the same mapping. It can be shown that replacing the derivation $B \to \epsilon$ with $B \to \mathtt{a} \mid \mathtt{b}$ results in an unambiguous extraction grammar which is equivalent to disjEqLen on any document different than ϵ. (Note however that this does not imply that the ref-languages they produce are equal.) ◀

Our main enumeration algorithm for extraction grammars relies on unambiguity and the following observation.

▶ **Proposition 12.** *In Proposition 10, if G is unambiguous then so is G'.*

3 Expressive Power and Evaluation

In this section we compare the expressiveness of context-free spanners compared to other studied classes of spanners and discuss its evaluation shortly.

3.1 Regular Spanners

A *variable-set automaton* A (or *vset-automaton*, for short) is a tuple $A := (X, Q, q_0, q_f, \delta)$ where $X \subseteq \mathsf{Vars}$ is a finite set of variables also referred to as $\mathsf{Vars}(A)$, Q is the set of *states*, $q_0, q_f \in Q$ are the *initial* and the *final* states, respectively, and $\delta \colon Q \times (\Sigma \cup \{\epsilon\} \cup \Gamma_X) \to 2^Q$ is the *transition function*. To define the semantics of A, we interpret A as a non-deterministic finite state automaton over the alphabet $\Sigma \cup \Gamma_X$, and define $\mathcal{R}(A)$ as the set of all ref-words $\mathbf{r} \in (\Sigma \cup \Gamma_X)^*$ such that some path from q_0 to q_f is labeled with $\mathbf{r}$. Like for regex formulas, we define $\mathsf{Ref}(A, \mathbf{d}) = \mathcal{R}(A) \cap \mathsf{Ref}(\mathbf{d})$ and finally we define for every document $\mathbf{d} \in \Sigma^*$: $[\![A]\!](\mathbf{d}) := \{\mu^{\mathbf{r}} \mid \mathbf{r} \in \mathsf{Ref}(A, \mathbf{d})\}$. The class of *regular spanners* equals the class of spanners that are expressible as a vset-automaton [12].

Inspired by Chomsky's hierarchy, we say that an extraction grammar G is *regular* if its productions are of the form $A \to \sigma B$ and $A \to \sigma$ where A, B are non-terminals and $\sigma \in (\Sigma \cup \Gamma_X)$. We then have the following equivalence that is strongly based on the equivalence of regular grammars and finite state automata.

▶ **Proposition 13.** *The class of spanners definable by regular extraction grammars is equal to the class of regular spanners.*

3.2 (Generalized) Core Spanners

An alternative way to define regular spanners is based on the notion of regex formulas: Formally, a *regex formula* is defined recursively by $\alpha := \emptyset \mid \epsilon \mid \sigma \mid \alpha \vee \alpha \mid \alpha \cdot \alpha \mid \alpha^* \mid \vdash_x \alpha \dashv_x$ where $\sigma \in \Sigma$ and $x \in \mathsf{Vars}$. We denote the set of variables whose variable operations occur in α by $\mathsf{Vars}(\alpha)$ and interpret each regex formula α as a generator of a ref-word language $\mathcal{R}(\alpha)$ over the extended alphabet $\Sigma \cup \Gamma_{\mathsf{Vars}(\alpha)}$. For every document $\mathbf{d} \in \Sigma^*$, we define $\mathsf{Ref}(\alpha, \mathbf{d}) = \mathcal{R}(\alpha) \cap \mathsf{Ref}(\mathbf{d})$, and the spanner $[\![\alpha]\!]$ by $[\![\alpha]\!](\mathbf{d}) := \{\mu^{\mathbf{r}} \mid \mathbf{r} \in \mathsf{Ref}(\alpha, \mathbf{d})\}$. The class of regular spanners is then defined as the closure of regex formulas under the relational algebra operators: union, projection and natural join.

In their efforts to capture the core of AQL which is IBM's SystemT query language, Fagin et al. [12] have presented the class of core spanners which is the closure of regex formulas under the positive operators, i.e., union, natural join and projection, along with the string equality selection that is defined as follows Let S be a spanner and let $x, y \in \mathsf{Vars}(S)$, the *string equality selection* $\zeta^{=}_{x,y} S$ is defined by $\mathsf{Vars}(\zeta^{=}_{x,y} S) = \mathsf{Vars}(S)$ and, for all $\mathbf{d} \in \Sigma^*$, $\zeta^{=}_{x,y} S(\mathbf{d})$ is the set of all $\mu \in S(\mathbf{d})$ where $\mathbf{d}_{\mu(x)} = \mathbf{d}_{\mu(y)}$. Note that unlike the join operator that joins mappings that have identical spans in their shared variables, the selection operator compares the substrings of $\mathbf{d}$ that are described by the spans and does not distinguish between different spans that span the same substrings.

The class of *generalized core spanners* is obtained by adding the difference operator. That is, it is defined as the closure of regex formulas under union, natural join, projection, string equality, and difference. We say that two classes $\mathcal{S}, \mathcal{S}'$ of spanners are *incomparable* if both $\mathcal{S} \setminus \mathcal{S}'$ and $\mathcal{S}' \setminus \mathcal{S}$ are not empty.

▶ **Proposition 14.** *The classes of core spanners and generalized core spanners are each incomparable with the class of context-free spanners.*

We conclude the discussion by a straightforward result on closure properties.

▶ **Proposition 15.** *The class of context-free spanners is closed under union and projection, and not closed under natural join and difference.*

3.3 Evaluating Context-Free Spanners

The *evaluation* problem of extraction grammars is that of computing $[\![G]\!](\mathbf{d})$ where $\mathbf{d}$ is a document and G is an extraction grammar. Our first observation is the following.

▶ **Proposition 16.** *For every extraction grammar G and every document $\mathbf{d}$ it holds that $[\![G]\!](\mathbf{d})$ can be computed in $O(|G|^2+|\mathbf{d}|^{2k+3}\, k^3\, |G|)$ time where G is associated with k variables.*

The proof of this proposition is obtained by iterating through all valid ref-words and using the Cocke-Younger-Kasami (CYK) parsing algorithm [23] to check whether the current valid ref-word is produced by G. We can, alternatively, use Valiant's parser [41] and obtain $O(|G|^2 + |\mathbf{d}|^{2k+\omega}\, k^{\omega}\, |G|)$ where $\omega < 2.373$ is the matrix multiplication exponent [43].

While the evaluation can be done in polynomial time in data complexity (where G is regarded as fixed and $\mathbf{d}$ as input), the output size might be quite big. To be more precise, for an extraction grammar G associated with k variables the output might consist of up to $|\mathbf{d}|^{2k}$ mappings. Instead of outputting these mappings altogether, we can output them sequentially (without repetitions) after some preprocessing.

Our main enumeration result is the following.

▶ **Theorem 17.** *For every unambiguous extraction grammar G and every document $\mathbf{d}$ there is an algorithm that outputs the mappings in $[\![G]\!](\mathbf{d})$ with delay $O(k)$ after $O(|\mathbf{d}|^5|G|^2 3^{4k})$ preprocessing where k is the number of variables G is associated with.*

Our algorithm consists of two main stages: preprocessing and enumeration. In the preprocessing stage, we manipulate the extraction grammar and do some precomputations which are later exploited in the enumeration stage in which we output the results sequentially. We remark that unambiguity is crucial for the enumeration stage as it allows to output the mappings without repetition.

Through the lens of data complexity, our enumeration algorithm outputs the results with constant delay after quintic preprocessing. That should be contrasted with regular spanners for which there exists a constant delay enumeration algorithm whose preprocessing is linear [2, 14]. In the following sections, we present the enumeration algorithm and discuss its correctness but before we deal with the special case $\mathbf{d} := \epsilon$. In this case, $[\![G]\!](\mathbf{d})$ is either empty or contains exactly one mapping (since, by definition, the document ϵ has exactly one span, namely $[1, 1\rangle$). Notice that $[\![G]\!](\mathbf{d})$ is empty if and only if G does not produce a ref-word that consists only of variable operations. To check this, it suffices to change the production rules of G by replacing every occurrence of $\tau \in \Gamma_X$ with ϵ and check whether the new grammar produces ϵ. This can be done in linear time [22] which completes the proof of this case.

4 Preprocessing of the Enumeration Algorithm

Due to Propositions 10 and 12, we can assume that our unambiguous extraction grammar is functional and in CNF. As this conversion requires $O(3^{2k}|G|^2)$, it can be counted as part of our preprocessing.

The preprocessing stage consists of two steps: in the first we adjust the extraction grammar to a given document and add subscripts to non-terminals to track this connection, and in the second we use superscripts to capture extra information regarding the variable operations.

4.1 Adjusting the Extraction Grammar to $\mathbf{d}$

Let $G := (X, V, \Sigma, P, S)$ be an extraction grammar in CNF and let $\mathbf{d} := \sigma_1 \cdots, \sigma_n, n \geq 1$ be a document. The goal of this step is to restrict G so that it will produce only the ref-words which clr maps to $\mathbf{d}$. To this end, we define the grammar $G_{\mathbf{d}}$ that is associated with the same set X of variables as G, and is defined as follows:

- The non-terminals are $\{A_{i,j} \mid A \in V,\, 1 \leq i \leq j \leq n\} \cup \{A_\epsilon \mid A \in V\}$,
- the terminals are Σ,
- the initial non-terminal is $S_{1,n}$, and
- the production rules are defined as follows:
  - $A_{i,i} \to \sigma_i$ for any $A \to \sigma_i \in P$,
  - $A_\epsilon \to \sigma$ for any $A \to \sigma \in P$ with $\sigma \in \Gamma_X$,
  - $A_\epsilon \to B_\epsilon C_\epsilon$ for any $A \to BC \in P$,
  - $A_{i,j} \to B_{i,j} C_\epsilon$ for any $1 \leq i \leq j \leq n$ and any $A \to BC \in P$,
  - $A_{i,j} \to B_\epsilon C_{i,j}$ for any $1 \leq i \leq j \leq n$ and any $A \to BC \in P$,
  - $A_{i,j} \to B_{i,i'} C_{i'+1,j}$ for any $1 \leq i \leq i' < j \leq n$ and $A \to BC \in P$.

We eliminate useless non-terminals from $G_{\mathbf{d}}$ and by a slight abuse of notation refer to the result as $G_{\mathbf{d}}$ from now on. The intuition behind this construction is that if the subscript of a non-terminal is i, j then this non-terminal produces a ref-word that clr maps to $\sigma_i \cdots \sigma_j$, and if it is ϵ then it produces a ref-word that consists only of variable operations.

► **Example 18.** Figure 6 presents a possible parse-tree of a grammar $G_{\mathbf{d}}$. ◄

We establish the following connection between G and $G_{\mathbf{d}}$.

► **Lemma 19.** *For every extraction grammar G in CNF, every document $\mathbf{d} := \sigma_1 \cdots \sigma_n$, every non-terminal A of G, and any ref-word $\mathbf{r} \in (\Sigma \cup \Gamma_X)^*$ with $\mathsf{clr}(\mathbf{r}) = \sigma_i \cdots \sigma_j$ the following holds: $A \Rightarrow_G^* \mathbf{r}$ if and only if $A_{i,j} \Rightarrow_{G_{\mathbf{d}}}^* \mathbf{r}$*

This allows us to conclude the following straightforward corollary.

► **Corollary 20.** *For every extraction grammar G in CNF and for every document $\mathbf{d}$, it holds that $\mathsf{Ref}(G, \mathbf{d}) = \mathcal{L}(G_{\mathbf{d}})$.*

We note that adjusting our extraction grammar to $\mathbf{d}$ is somewhat similar to the CYK algorithm [23] and therefore it is valid on extraction grammars G in CNF. For a similar reason, we obtain the following complexity which is cubic in $|\mathbf{d}|$.

► **Proposition 21.** *For every extraction grammar G in CNF and for every document $\mathbf{d}$, it holds that $G_{\mathbf{d}}$ can be constructed in $O(|\mathbf{d}|^3 |G|)$.*

Can the complexity of the adjustment be improved? We leave this as an open question. We note, however, that it might be possible to use similar ideas used by Earley's algorithm [11] to decrease the complexity of this step.

4.2 Constructing the Decorated Grammar

The goal of this step of the preprocessing is to encode the information on the produced variable operations within the terminals and non-terminals. We obtain from $G_{\mathbf{d}}$, constructed in the previous step, a new grammar, namely $\textsc{decorGrmr}(G_{\mathbf{d}})$, that produces *decorated words* over the alphabet $\{(\mathbf{x}, i, \mathbf{y}) \mid \mathbf{x}, \mathbf{y} \subseteq \Gamma_X, 1 \leq i \leq n\}$. A terminal $(\mathbf{x}, i, \mathbf{y})$ indicates that $\mathbf{x}$ and $\mathbf{y}$ are variable operations that occur right before and right after σ_i, respectively. (Notice that $\mathbf{x}, \mathbf{y}$ does not necessarily contain all of these variable operations as some of the variable operations that appear, e.g., after i, can be contained in $\mathbf{x}'$ in case $(\mathbf{x}', i+1, \mathbf{y}')$ is the terminal that appears right after $(\mathbf{x}, i, \mathbf{y})$.) This information is propagated also to the non-terminals such that a non-terminal with a superscript $\mathbf{x}, \mathbf{y}$ indicates that $\mathbf{x}$ and $\mathbf{y}$ are variable operations at the beginning and end, respectively, of the sub decorated word produced by this non-terminal. Non-terminals with subscript ϵ are those that produce sequences of variable operations.

To define $\textsc{decorGrmr}(G_{\mathbf{d}})$, we need G to be functional. The following key observation is used in the formal definition of $\textsc{decorGrmr}(G_{\mathbf{d}})$ and is based on the functionality of G.

▶ **Proposition 22.** *For every functional extraction grammar G and every non-terminal A of G there is a set $\mathbf{x}_A \subseteq \Gamma_X$ of variable operations such that for every ref-word $\mathbf{r}$ where $A \Rightarrow^* \mathbf{r}$ the variable operations that appear in $\mathbf{r}$ are exactly those in $\mathbf{x}_A$. Computing all sets $\mathbf{x}_A$ can be done in $O(|G|)$.*

In other words, for functional extraction grammars, the information on the variable operations is stored implicitly in the non-terminals. The grammar $\textsc{decorGrmr}(G_{\mathbf{d}})$ is defined in three steps.

Step 1. We set the following production rules for all subsets $\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{w} \subseteq \Gamma_X$ that are pairwise disjoint:

- $A^{\emptyset,\emptyset}_{i,i} \to \sigma_i$ for any rule $A_{i,i} \to \sigma_i$ in $G_{\mathbf{d}}$,
- $A_\epsilon \to \epsilon$ for any rule $A_\epsilon \to \tau$ in $G_{\mathbf{d}}$ (with $\tau \in \Gamma_X$),
- $A_\epsilon \to B_\epsilon C_\epsilon$ for any rule $A_\epsilon \to B_\epsilon C_\epsilon$ in $G_{\mathbf{d}}$,
- $A^{\mathbf{x},\mathbf{y}\cup\mathbf{x}_C}_{i,j} \to B^{\mathbf{x},\mathbf{y}}_{i,j} C_\epsilon$ for any rule $A_{i,j} \to B_{i,j} C_\epsilon$ in $G_{\mathbf{d}}$ and $\mathbf{x} \cap \mathbf{x}_C = \mathbf{y} \cap \mathbf{x}_C = \emptyset$,
- $A^{\mathbf{x}\cup\mathbf{x}_B,\mathbf{y}}_{i,j} \to B_\epsilon C^{\mathbf{x},\mathbf{y}}_{i,j}$ for any rule $A_{i,j} \to B_\epsilon C_{i,j}$ in $G_{\mathbf{d}}$ and $\mathbf{x} \cap \mathbf{x}_B = \mathbf{y} \cap \mathbf{x}_B = \emptyset$,
- $A^{\mathbf{x},\mathbf{w}}_{i,j} \to B^{\mathbf{x},\mathbf{y}}_{i,i'} C^{\mathbf{z},\mathbf{w}}_{i'+1,j}$ for any rule $A_{i,j} \to B_{i,i'} C_{i'+1,j}$ in $G_{\mathbf{d}}$ and pairwise disjoint $\mathbf{x}, \mathbf{y}, \mathbf{z}, \mathbf{w}$,

with $\mathbf{x}_B$ and $\mathbf{x}_C$ defined as in Proposition 22.

Step 2. We process the resulting grammar by three standard operations [23] in the following order: (i) we eliminate useless non-terminals (i.e., those that do not produce a terminal string or are not reachable from the initial non-terminal), (ii) we eliminate epsilon-productions, and (iii) we eliminate unit productions. We elaborate on (iii) as it is important for the sequel. To eliminate unit productions we compute for each non-terminal the set of non-terminals that are reachable from it by unit productions only. That is, we say that a non-terminal B is *reachable* form non-terminal A if there is a sequence of unit productions of the form $A_1 \to A_2, \cdots A_{n-1} \to A_n$ with $A_1 = A$ and $A_n = B$. We then replace every production $B \to \alpha$ which is not a unit production with $A \to \alpha$, and after that discard all unit productions.

Step 3. The last step of the construction is adding a fresh start symbol S and adding the production rules $S \to S^{\mathbf{x},\mathbf{y}}_{1,n}$ for any non-terminal of the form $S^{\mathbf{x},\mathbf{y}}_{1,n}$. We also replace each production of the form $A^{\mathbf{x},\mathbf{y}}_{i,i} \to \sigma_i$ with $A^{\mathbf{x},\mathbf{y}}_{i,i} \to (\mathbf{x}, i, \mathbf{y})$. This can be viewed as a 'syntactic sugar' since it is only intended to help us formulate easily the connection between the grammar G and $\textsc{decorGrmr}(G_{\mathbf{d}})$.

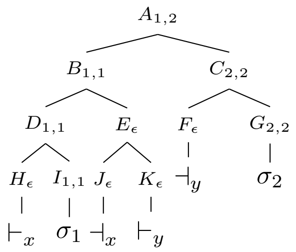

Figure 6 After the adjustment to $\mathbf{d}$

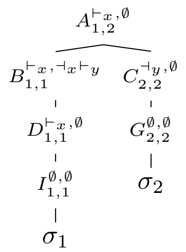

Figure 7 Before step 2 (*iii*)

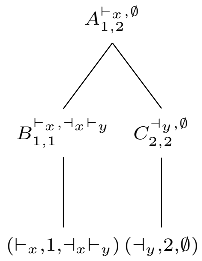

Figure 8 After step 3

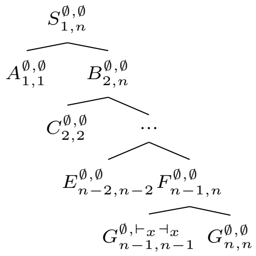

Figure 9 Non-stable non-terminals

▶ **Example 23.** Figures 7 and 8 illustrate the different steps in the construction of the decorated grammar $\textsc{decorGrmr}(G_\mathbf{d})$. For simplicity, we present the superscripts as pairs of sequences (each represent elements in the set) separated by commas ','. ◀

Note that by a simple induction it can be shown that the resulting grammar does no longer contain non-terminals of the form A_ϵ. We denote the resulting grammar and its set of non-terminals by $\textsc{decorGrmr}(G_\mathbf{d})$ and V^{DEC}, respectively.

The (X, d)-mapping μ^w that corresponds with $w := (\mathbf{x}_1, 1, \mathbf{y}_1)\cdots(\mathbf{x}_n, n, \mathbf{y}_n)$ (which is a decorated word produced by $\textsc{decorGrmr}(G_\mathbf{d})$) is defined by $\mu^w(x) = [i, j\rangle$ where $\vdash_x \in \mathbf{x}_i \cup \mathbf{y}_{i-1}$ and $\dashv_x \in \mathbf{x}_j \cup \mathbf{y}_{j-1}$ where $\mathbf{y}_0 = \mathbf{x}_{n+1} = \emptyset$. We say that a decorated word w is *valid* if $\mu^w(x)$ is well-defined for every $x \in X$.

▶ **Proposition 24.** *For every functional extraction grammar G in CNF and for every document $\mathbf{d}$, if G is unambiguous then $\textsc{decorGrmr}(G_\mathbf{d})$ is unambiguous.*

This allows us to establish the following connection between $\textsc{decorGrmr}(G_\mathbf{d})$ and $[\![G]\!](\mathbf{d})$.

▶ **Lemma 25.** *For every functional unambiguous extraction grammar G in CNF and for every document $\mathbf{d}$, every decorated word produced by $\textsc{decorGrmr}(G_\mathbf{d})$ is valid and*

$$[\![G]\!](\mathbf{d}) = \{\mu^w \mid S \Rightarrow^*_{\textsc{decorGrmr}(G_\mathbf{d})} w\}.$$

Finally, combining Proposition 24 and Lemma 25 leads to the following direct corollary.

▶ **Corollary 26.** *For every functional unambiguous extraction grammar G in CNF and for every document $\mathbf{d}$, enumerating mappings in $[\![G]\!](\mathbf{d})$ can be done by enumerating parse-trees of decorated words in $\{w \mid S \Rightarrow^*_{\textsc{decorGrmr}(G_\mathbf{d})} w\}$.*

To summarize the complexity of constructing $\textsc{decorGrmr}(G_\mathbf{d})$ we have:

▶ **Proposition 27.** *For every functional unambiguous extraction grammar G in CNF and for every document $\mathbf{d}$, $\textsc{decorGrmr}(G_\mathbf{d})$ can be constructed in $O(|G_\mathbf{d}|\, 5^{2k}) = O(|\mathbf{d}|^3 |G|\, 5^{2k})$ where k is the number of variables associated with G.*

5 Enumeration Algorithm

Our enumerating algorithm builds recursively the parse-trees of the decorated grammar $\textsc{decorGrmr}(G_\mathbf{d})$. Before presenting it, we discuss some of the main ideas that allow us to obtain a constant delay between every two consecutive outputs.

5.1 Stable non-terminals

The non-terminals of $\textsc{decorGrmr}(G_{\mathbf{d}})$ are decorated with superscripts and subscripts that give extra information that can be exploited in the process of the derivation.

▶ **Example 28.** Figure 10 presents a partial parse-tree (without the leaves and the first production) for a decorated word in $\textsc{decorGrmr}(G_{\mathbf{d}})$. Notice that the variable operations that appear in the subtrees rooted in the non-terminal $C_{1,3}^{\vdash_x \vdash_y, \dashv_y}$ are only those indicated in its superscript. That is, there are no variable operations that occur between positions $1, 2$ and $2, 3$. ◀

Motivated by this, we say that a non-terminal $A_{i,j}^{\mathbf{x},\mathbf{y}}$ of $\textsc{decorGrmr}(G_{\mathbf{d}})$ is *stable* if $\mathbf{x}_A = \mathbf{x} \cup \mathbf{y}$.

▶ **Lemma 29.** *For every functional extraction grammar G in CNF and for every document $\mathbf{d}$, the set of stable non-terminals of $\textsc{decorGrmr}(G_{\mathbf{d}})$ is computable in $O(|G_{\mathbf{d}}|5^{2k})$ where k is the number of variables G is associated with.*

Therefore, while constructing the parse-trees of $\textsc{decorGrmr}(G_{\mathbf{d}})$ whenever we reach a stable non-terminal we can stop since its subtree does not affect the mapping.

5.2 The Jump Function

If G is associated with k variables, there are exactly $2k$ variable operations in each ref-word produced by G. Hence, we can bound the number of non-stable non-terminals in a parse-tree of $\textsc{decorGrmr}(G_{\mathbf{d}})$. Nevertheless, the depth of a non-stable non-terminal can be linear in $|\mathbf{d}|$ as the following example suggests.

▶ **Example 30.** Consider the non-stable non-terminal $F_{n-1,n}^{\emptyset,\emptyset}$ in the partial parse tree in Figure 9 of the decorated word $(\emptyset, 1, \emptyset) \cdots (\emptyset, n-1, \vdash_x \dashv_x)(\emptyset, n, \emptyset)$. Observe that the depth of this non-terminal is linear in n. ◀

Since we want the delay of our algorithm to be independent of $|\mathbf{d}|$, we skip parts of the parse tree in which no variable operation occurs. This idea somewhat resembles an idea that was implemented by Amarilli et al. [2] in their constant delay enumeration algorithm for regular spanners represented as vset-automata. There, they defined a function that 'jumps' from one state to the other if the path from the former to the latter does not contain any variable operation. We extend this idea to extraction grammars by defining the notion of skippable productions. Intuitively, when we focus on a non-terminal in a parse tree, the corresponding mapping is affected by either the left subtree of this non-terminal, or by its right subtree, or by the production applied on the non-terminal itself (or by any combination of the above). If the mapping is affected exclusively by the left (right, respectively) subtree then we can skip the production and move to check the left (right, respectively) subtree, and do so recursively until we reach a production for which this is no longer the case.

Formally, a *skippable* production rule is of the form $A_{i,j}^{\mathbf{x},\mathbf{y}} \to B_{i,i'}^{\mathbf{x},\mathbf{z}} C_{i'+1,j}^{\mathbf{z}',\mathbf{y}}$ where (a) $A_{i,j}^{\mathbf{x},\mathbf{y}}$ is non-stable, (b) $\mathbf{z} = \mathbf{z}' = \emptyset$, and (c) exactly one of $B_{i,i'}^{\mathbf{x},\mathbf{z}}, C_{i'+1,j}^{\mathbf{z}',\mathbf{y}}$ is stable. Intuitively, (a) assures that the parse tree rooted in $A_{i,j}^{\mathbf{x},\mathbf{y}}$ affects the mapping, (b) assures that the production applied on $A_{i,j}^{\mathbf{x},\mathbf{y}}$ does not affect the mapping and (c) assures that exactly one subtree of $A_{i,j}^{\mathbf{x},\mathbf{y}}$ (either the one rooted at $B_{i,i'}^{\mathbf{x},\mathbf{z}}$ if $C_{i'+1,j}^{\mathbf{z}',\mathbf{y}}$ is stable, or the one rooted at $C_{i'+1,j}^{\mathbf{z}',\mathbf{y}}$ if $B_{i,i'}^{\mathbf{x},\mathbf{z}}$ is stable) affects the mapping. We then say that a skippable production rule ρ *follows* a skippable production rule ρ' if the non-stable non-terminal in the right-hand side of ρ' is the non-terminal in the left-hand side of ρ. The function $\textsc{jump}\colon V^{\text{DEC}} \to 2^{V^{\text{DEC}}}$ is defined by $B \in \textsc{jump}(A_{i,j}^{\mathbf{x},\mathbf{y}})$ if there is a sequence of skippable production rules $\rho_1, \cdots, \rho_m$ such that:

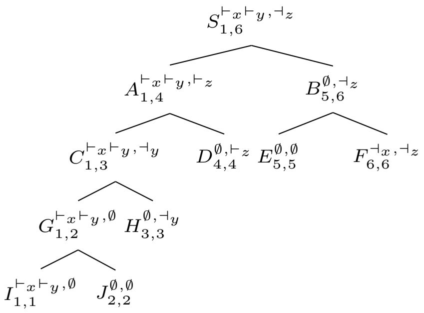

Figure 10 DECORGRMR($G_{\mathbf{d}}$) Parse tree

procedure ENUMERATE(α, map)
if $\alpha = \epsilon$ **then**
  output map
denote α *by* $A \cdot \alpha'$;
foreach $B \in$ JUMP(A) **do**
  foreach (β, map$'$) $\in$ APPLYPROD(B) **do**
    ENUMERATE($\beta \cdot \alpha'$, map $\cup$ map$'$);

Figure 11 Main enumeration algorithm

- ρ_ι follows $\rho_{\iota-1}$ for every ι,
- the left-hand side of ρ_1 is $A_{i,j}^{\mathbf{x},\mathbf{y}}$,
- the non-stable non-terminal in the right-hand side of ρ_m is B,
- there is a production rule that is not skippable whose left-hand side is B.

▶ **Example 31.** In the decorated grammar whose (one of its) parse tree appears in Figure 9 it holds that $F_{n-1,n}^{\emptyset,\emptyset} \in$ JUMP($S_{1,n}^{\emptyset,\emptyset}$). ◀

The acyclic nature of the decorated grammar (that is, the fact that a non-terminal cannot be produced from itself) enables us to obtain the following upper bound for the computation of the JUMP function.

▶ **Lemma 32.** *For every functional unambiguous extraction grammar G in CNF and for every document $\mathbf{d}$, the* JUMP *function is computable in $O(|\mathbf{d}|^5 3^{4k} |G|^2)$ where k is the number of variables G is associated with.*

The proof of this lemma relies on the acyclic nature of DECORGRMR($G_{\mathbf{d}}$). Lemmas 29 and 32 imply that we can find the non-stable non-terminals as well as compute the JUMP function as part of the quintic preprocessing. It is important to note that if we can reduce the complexity of computing the JUMP function then we can reduce the preprocessing time.

5.3 The Algorithm

Our main enumeration algorithm is presented in Figure 11 and outputs $(X, \mathbf{d})$-mappings μ represented as sets of pairs $(\vdash_x, i), (\dashv_x, j)$ whenever $\mu(x) = [i, j\rangle$. The procedure APPLYPROD is called with a non-terminal $A_{i,j}^{\mathbf{x},\mathbf{y}}$ that is (a) non-stable and (b) appears at the left-hand side of at least one rule that is not skippable. The procedure APPLYPROD is an iterator that outputs with constant delay all those pairs (β, map) for which there exists a not skippable rule of the form $A_{i,j}^{\mathbf{x},\mathbf{y}} \to B_{i,i'}^{\mathbf{x},\mathbf{z}} C_{i'+1,j}^{\mathbf{z}',\mathbf{y}}$ such that the following hold:

- map $= \{(\tau, i'+1) \mid \tau \in \mathbf{z} \cup \mathbf{z}'\}$, and
- β is the concatenation of the non-stable terminals amongst $B_{i,i'}^{\mathbf{x},\mathbf{z}}$ and $C_{i'+1,j}^{\mathbf{z}',\mathbf{y}}$.

Notice that since $A_{i,j}^{\mathbf{x},\mathbf{y}}$ is non-stable and since $A_{i,j}^{\mathbf{x},\mathbf{y}} \to B_{i,i'}^{\mathbf{x},\mathbf{z}} C_{i'+1,j}^{\mathbf{z}',\mathbf{y}}$ is not skippable, it holds that either $\mathbf{z} \neq \emptyset$ or $\mathbf{z}' \neq \emptyset$ (or both). Thus, the returned map is not empty which implies that every call to this procedure adds information on the mapping, and thus the number of calls is bounded. Notice also that β is the concatenation of the non-terminals among $B_{i,i'}^{\mathbf{x},\mathbf{z}}$ and $C_{i'+1,j}^{\mathbf{z}',\mathbf{y}}$ that affect the mapping.

▶ **Example 33.** The procedure APPLYPROD applied on $S_{1,6}^{\vdash_x\vdash_y,\dashv_z}$ from Figure 10 adds the pair $(\vdash_z, 5)$ to map; When applied on $A_{1,4}^{\vdash_x\vdash_y,\vdash_z}$, it adds the pair $(\dashv_y, 4)$ to map; When applied on $B_{5,6}^{\emptyset,\dashv_z}$, it adds the pair $(\dashv_x, 6)$ to map. ◀

The recursive procedure ENUMERATE outputs mappings as a set of pairs of the form (γ, i) with $\gamma \in \Gamma_X$ a variable operation and $1 \leq i \leq n$. The main enumeration algorithm calls the recursive procedure ENUMERATE with pairs $(S_{1,n}^{\mathbf{x},\mathbf{y}}, \mathsf{map})$ where $S_{1,n}^{\mathbf{x},\mathbf{y}}$ is a non-terminal in DECORGRMR$(G_{\mathbf{d}})$, and map is the set containing pairs $(\tau, 1)$ for any $\tau \in \mathbf{x}$, and $(\tau, n+1)$ for any $\tau \in \mathbf{y}$. The recursive procedure ENUMERATE gets a pair (α, map) as input where α is a (possibly empty) sequence of non-stable non-terminals and map is a set of pairs of the above form. It recursively constructs an output mapping by applying derivations on the non-stable non-terminals (by calling APPLYPROD) while skipping the skippable productions (by using JUMP). We assume that ENUMERATE has $O(1)$ access to everything computed in the preprocessing stage, that is, the grammar DECORGRMR$(G_{\mathbf{d}})$, the JUMP function, and the sets of stable and non-stable non-terminals.

▶ **Theorem 34.** *For every functional unambiguous extraction grammar G in CNF and for every document $\mathbf{d}$, the main enumerating algorithm described above enumerates the mappings in $[\![G]\!](\mathbf{d})$ (without repetitions) with delay of $O(k)$ between each two consecutive mappings where k is the number of variables G is associated with.*

Had G been ambiguous, the complexity guarantees on the delay would not have held.

Finally, we remark that the proof of Theorem 17 follows from Corollary 25, Proposition 27, Lemma 29, Lemma 32, and Theorem 34.

6 Conclusion

In this paper we propose a new grammar-based language for document spanners, namely extraction grammars. We compare the expressiveness of context-free spanners with previously studied classes of spanners and present a pushdown model for these spanners. We present an enumeration algorithm for unambiguous grammars that outputs results with a constant delay after quintic preprocessing in data complexity. We conclude by suggesting several future research directions.

To reach a full understanding of the expressiveness of context-free spanners, one should characterize the string relations that can be expressed with context-free spanners. This can be done by understanding the expressiveness of context-free grammars enriched with string equality selection. We note that there are some similarities between recursive Datalog over regex formulas [35] and extraction grammars. Yet, with the former we reach the full expressiveness of polynomial time spanners (data complexity) whereas with the latter we cannot express string equality. Understanding the connection between these two formalisms better can be a step in understanding the expressive power of extraction grammars.

Regarding our enumeration complexity, it might be possible to decrease the preprocessing complexity by using other techniques to compute the jump function. Another direction is to find restricted classes of extraction grammars that are more expressive than regular spanners yet allow linear time preprocessing (similarly to [2]).

It can be interesting to examine more carefully whether the techniques used here for enumerating the derivations can be applied also for enumerating queries on trees, or enumerating queries beyond MSO on strings. This connects to a recent line of work on efficient enumeration algorithms for monadic-second-order queries on trees [3]. Can our techniques be used to obtain efficient evaluation for more expressive queries?

Acknowledgments

The author is grateful to Arnaud Durand, Michael Kaminski, Benny Kimelfeld, and Leonid Libkin for the discussions and their useful comments, and to Dominik D. Freydenberger for the helpful references.

References

1 Jitendra Ajmera, Hyung-Il Ahn, Meena Nagarajan, Ashish Verma, Danish Contractor, Stephen Dill, and Matthew Denesuk. A CRM system for social media: challenges and experiences. In *WWW*, pages 49–58. ACM, 2013.

2 Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-delay enumeration for nondeterministic document spanners. In *ICDT*, pages 22:1–22:19, 2019.

3 Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Enumeration on trees with tractable combined complexity and efficient updates. In *PODS*, pages 89–103, 2019.

4 Edward Benson, Aria Haghighi, and Regina Barzilay. Event discovery in social media feeds. In *ACL*, pages 389–398. The Association for Computer Linguistics, 2011.

5 J. Richard Büchi and Lawrence H. Landweber. Definability in the monadic second-order theory of successor. *J. Symb. Log.*, 34(2):166–170, 1969.

6 Laura Chiticariu, Rajasekar Krishnamurthy, Yunyao Li, Sriram Raghavan, Frederick Reiss, and Shivakumar Vaithyanathan. SystemT: An algebraic approach to declarative information extraction. In *ACL*, pages 128–137, 2010.

7 Noam Chomsky. On certain formal properties of grammars. *Information and Control*, 2(2):137–167, 1959.

8 Johannes Doleschal, Benny Kimelfeld, Wim Martens, Yoav Nahshon, and Frank Neven. Split-correctness in information extraction. In *PODS*, pages 149–163, 2019.

9 Johannes Doleschal, Benny Kimelfeld, Wim Martens, and Liat Peterfreund. Weight annotation in information extraction. In *ICDT*, volume 155 of *LIPIcs*, pages 8:1–8:18. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2020.

10 Pál Dömösi. Unusual algorithms for lexicographical enumeration. *Acta Cybernetica*, 14(3):461–468, 2000.

11 Jay Earley. An efficient context-free parsing algorithm. *Communications of the ACM*, 13(2):94–102, 1970.

12 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Document spanners: A formal approach to information extraction. *J. ACM*, 62(2):12, 2015.

13 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Declarative cleaning of inconsistencies in information extraction. *ACM Trans. Database Syst.*, 41(1):6:1–6:44, 2016.

14 Fernando Florenzano, Cristian Riveros, Martín Ugarte, Stijn Vansummeren, and Domagoj Vrgoc. Constant delay algorithms for regular document spanners. In *PODS*, pages 165–177. ACM, 2018.

15 Dominik D. Freydenberger. A logic for document spanners. In *ICDT*, volume 68 of *LIPIcs*, pages 13:1–13:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

16 Dominik D. Freydenberger. A logic for document spanners. *Theory Comput. Syst.*, 63(7):1679–1754, 2019.

17 Dominik D. Freydenberger and Mario Holldack. Document spanners: From expressive power to decision problems. In *ICDT*, volume 48 of *LIPIcs*, pages 17:1–17:17. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2016.

18 Dominik D. Freydenberger and Mario Holldack. Document spanners: From expressive power to decision problems. *Theory Comput. Syst.*, 62(4):854–898, 2018.

19 Dominik D. Freydenberger, Benny Kimelfeld, and Liat Peterfreund. Joining extractions of regular expressions. In *PODS*, pages 137–149, 2018.

20 Dominik D. Freydenberger and Liat Peterfreund. Finite models and the theory of concatenation. *CoRR*, abs/1912.06110, 2019. URL: `http://arxiv.org/abs/1912.06110`.

21 Dominik D. Freydenberger and Sam M. Thompson. Dynamic complexity of document spanners. In *ICDT*, volume 155, pages 11:1–11:21, 2020.

22 John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation - international edition (2. ed)*. Addison-Wesley, 2003.

23 John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to automata theory, languages, and computation, 3rd Edition*. Pearson international edition. Addison-Wesley, 2007.

24 Donald E. Knuth. Semantics of context-free languages. *Mathematical Systems Theory*, 2(2):127–145, 1968.

25 Donald E. Knuth. Correction: Semantics of context-free languages. *Mathematical Systems Theory*, 5(1):95–96, 1971.

26 Clemens Lautemann, Thomas Schwentick, and Denis Thérien. Logics for context-free languages. In *CSL*, volume 933 of *Lecture Notes in Computer Science*, pages 205–216. Springer, 1994.

27 Yaoyong Li, Kalina Bontcheva, and Hamish Cunningham. SVM based learning system for information extraction. In *Deterministic and Statistical Methods in Machine Learning, First International Workshop, Sheffield, UK, September 7-10, 2004, Revised Lectures*, pages 319–339, 2004.

28 Yunyao Li, Frederick Reiss, and Laura Chiticariu. SystemT: A declarative information extraction system. In *ACL*, pages 109–114. ACL, 2011.

29 Erkki Mäkinen. On lexicographic enumeration of regular and context-free languages. *Acta Cybernetica*, 13(1):55–61, 1997.

30 Francisco Maturana, Cristian Riveros, and Domagoj Vrgoč. Document spanners for extracting incomplete information: Expressiveness and complexity. In *PODS*, pages 125–136, 2018.

31 Andrea Moro, Marco Tettamanti, Daniela Perani, Caterina Donati, Stefano F Cappa, and Ferruccio Fazio. Syntax and the brain: disentangling grammar by selective anomalies. *Neuroimage*, 13(1):110–118, 2001.

32 Takashi Nagashima. A formal deductive system for CFG. *Hitotsubashi journal of arts and sciences*, 28(1):39–43, 1987.

33 Yoav Nahshon, Liat Peterfreund, and Stijn Vansummeren. Incorporating information extraction in the relational database model. In *WebDB*, page 6. ACM, 2016.

34 Liat Peterfreund, Dominik D. Freydenberger, Benny Kimelfeld, and Markus Kröll. Complexity bounds for relational algebra over document spanners. In *PODS*, pages 320–334. ACM, 2019.

35 Liat Peterfreund, Balder ten Cate, Ronald Fagin, and Benny Kimelfeld. Recursive programs for document spanners. In *ICDT*, volume 127 of *LIPIcs*, pages 13:1–13:18. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2019.

36 Christopher De Sa, Alexander Ratner, Christopher Ré, Jaeho Shin, Feiran Wang, Sen Wu, and Ce Zhang. Deepdive: Declarative knowledge base construction. *SIGMOD Record*, 45(1):60–67, 2016.

37 Markus L. Schmid. Characterising REGEX languages by regular languages equipped with factor-referencing. *Inf. Comput.*, 249:1–17, 2016.

38 Rania A Abul Seoud, Abou-Bakr M Youssef, and Yasser M Kadah. Extraction of protein interaction information from unstructured text using a link grammar parser. In *2007 International Conference on Computer Engineering & Systems*, pages 70–75. IEEE, 2007.

39 Warren Shen, AnHai Doan, Jeffrey F. Naughton, and Raghu Ramakrishnan. Declarative information extraction using Datalog with embedded extraction predicates. In *VLDB*, pages 1033–1044, 2007.

40 Jason M Smith and David Stotts. SPQR: Flexible automated design pattern extraction from source code. In *18th IEEE International Conference on Automated Software Engineering*, pages 215–224. IEEE, 2003.

41 Leslie G. Valiant. General context-free recognition in less than cubic time. *J. Comput. Syst. Sci.*, 10(2):308–315, 1975. doi:10.1016/S0022-0000(75)80046-8.

42 Huang Wen-Ji. Enumerating sentences of context free language based on first one in order. *Journal of Computer Research and Development*, 41(1):9–14, 2004.

43 Virginia Vassilevska Williams. Multiplying matrices faster than coppersmith-winograd. In *STOC*, pages 887–898. ACM, 2012.

44 Hua Xu, Shane P. Stenner, Son Doan, Kevin B. Johnson, Lemuel R. Waitman, and Joshua C. Denny. MedEx: a medication information extraction system for clinical narratives. *JAMIA*, 17(1):19–24, 2010.

45 Akane Yakushiji, Yuka Tateisi, Yusuke Miyao, and Jun-ichi Tsujii. Event extraction from biomedical papers using a full parser. In *Biocomputing 2001*, pages 408–419. World Scientific, 2000.

46 Huaiyu Zhu, Sriram Raghavan, Shivakumar Vaithyanathan, and Alexander Löser. Navigating the intranet with high precision. In *WWW*, pages 491–500. ACM, 2007.

A Appendix for Section 2

▶ **Proposition 10.** *Every extraction grammar G can be converted into an equivalent functional extraction grammar G' in $O(|G|^2 + 3^{2k}|G|)$ time where k is the number of variables G is associated with.*

Proof. Let $G := (X, V, \Sigma, P, S)$ be an extraction grammar. We assume that G is in CNF since if it is not then we can convert it into one in $O(|G|^2)$ with the standard algorithm presented, e.g., in [23]. Note that the resulting grammar is $O(|G|)$. We define an extraction grammar G' that is associated with X as well as follows. Its non-terminals are $X \times 2^{\Gamma_X}$, its terminals are Σ, the start variable is (S, Γ_X), and its production rules are as follows.

- $(A, X_1) \to (B, X_2)(C, X_3)$ whenever $A \to BC \in P$, and $X_1 = X_2 \cup X_3$, and $X_2 \cap X_3 = \emptyset$, and for any $x \in X$ if $\dashv_x \in X_2$ then $\vdash_x \notin X_3$;
- $(A, \emptyset) \to \sigma$ whenever $A \to \sigma \in P$ with $\sigma \in \Sigma$; and
- $(A, \{\tau\}) \to \tau$ whenever $A \to \tau \in P$ with $\tau \in \Gamma_X$.

A straightforward induction shows that G' is functional and that it is equivalent to G. The number of ways to choose two disjoint subsets of Γ_X is 3^{2k} and therefore complexity of the construction is $O(|G|^2 + 3^{2k}|G|)$ where k is the number of variables associated with G. Notice that G' is in CNF since G is in CNF. Notice also that if G is in CNF to begin with, the complexity reduces to $O(3^{2k}|G|)$. ◀

▶ **Proposition 12.** *In Proposition 10, if G is unambiguous then so is G'.*

Proof. Assume that G is unambiguous. It suffices to show that converting it to CNF G'' results in an unambiguous extraction grammar and that the conversion to G' preserve unambiguity (that is, if G'' is unambiguous then so is G'). This is straightforward since if we assume that G' is not unambiguous than we can easily show that so is G'' (by translating G''s derivations to G'' by omitting the second coordinate). So it is left to show that the conversion to CNF preserves unambiguity. For that, we repeat here the details of the algorithm presented in [22, Section 7.1.5] for converting a grammar G to CNF. The algortihm consists of three steps:

1. We omit ϵ-productions, unit productions and useless symbols.
2. We arrange that all bodies of length 2 or more consists only of variables;
3. We break bodies of length 3 or more into a cascade of productions, each with a body consisting of two variables.

In step 1 we do not add ambiguity as we reduce productions. In step 2 we add productions of the form $A \to \sigma$ but these do not affect unambiguity as these As are fresh non-terminals and for each σ we have a unique A. In step 3 we replace rules of the form $A \to A_1 \cdots A_n$ into the cascade $A \to A_1B_1, B_1 \to A_2B_2, \cdots B_{n-2} \to A_{n-1}A_n$ where all B_is are fresh non-terminals. Thus, this step do not affect unambiguity. ◀

B Appendix for Section 3

▶ **Proposition 13.** *The class of spanners definable by regular extraction grammars is equal to the class of regular spanners.*

Proof. Let us consider the regular spanner represented as the vset-automaton A. Since every regular language is also a context-free language we can construct a pushdown automaton A'

such that $\mathcal{R}(A') = \mathcal{R}(A)$. Thus, $[\![A]\!] = [\![A']\!]$

For the second direction we have to show that every spanner described by a right-regular extraction grammar is a regular spanner. Let $G = (V, \Sigma, P, S)$ be a right-regular extraction grammar. We use a similar construction used for converting a right regular grammar into a non-deterministic automaton while treating the variable operations as terminal symbols. The states of the automaton are a new final state q_f along with a state q_A for every non-terminal $A \in V$ with the state q_S as the initial state. The transition function δ is the following: for every $A \to \sigma B$ we set $\delta(q_A, \sigma) = q_B$ and for $A \to \sigma$ we set $\delta(q_A, \sigma) = q_f$. It holds that $\mathcal{R}(G) = \mathcal{L}(A)$ and in particular, for every $\mathbf{d}$ we have that $[\![G]\!](\mathbf{d}) = [\![A]\!](\mathbf{d})$. ◀

▶ **Example 35.** Let us consider the following Boolean core spanner given as the expression $\pi_\emptyset\, \zeta^{=}_{x,y}\big(\vdash_x \Sigma^* \dashv_x \vdash_y \Sigma^* \dashv_y\big)$. It can be shown that the spanner $[\![\zeta^{=}_{x,y}\big(\vdash_x \Sigma^* \dashv_x \vdash_y \Sigma^* \dashv_y\big)]\!]$ is evaluated to a non-empty $(\{x, y\}, \mathbf{d})$-relation on $\mathbf{d}$ if and only if $\mathbf{d}$ is of the form ww for some $w \in \Sigma^*$ (and in this case it contains a single $(\{x, y\}, \mathbf{d})$-mapping). For instance, on $\mathbf{d}_1 := \mathtt{abab}$ we have that $[\![\zeta^{=}_{x,y}\big(\vdash_x \Sigma^* \dashv_x \vdash_y \Sigma^* \dashv_y\big)]\!](\mathbf{d}_1)$ contains exactly one $(\{x, y\}, \mathbf{d}_1)$-mapping μ_1 that is defined such that $\mu_1(x) = [1, 3\rangle$ and $\mu_1(y) = [3, 5\rangle$. On the other hand, on $\mathbf{d}_2 := \mathtt{aba}$ and on $\mathbf{d}_3 := \mathtt{abaa}$ it holds that $[\![\zeta^{=}_{x,y}\big(\vdash_x \Sigma^* \dashv_x \vdash_y \Sigma^* \dashv_y\big)]\!](\mathbf{d}_i)$ for $i = 2, 3$ is empty. ◀

▶ **Proposition 14.** *The classes of core spanners and generalized core spanners are each incomparable with the class of context-free spanners.*

Proof. The proof of this proposition is based partly on the following observation which, in turn, is based on the definition: A language L is *definable* by a Boolean spanner S if for every word $w \in \Sigma^*$ it holds that $S(w) \neq \emptyset$ if only if $w \in L$.

▷ Observation 36. Boolean extraction grammars capture the context-free languages.

This observation is straightforward as every extraction grammar that is associated with an empty set of variables can be interpreted as a context-free grammar. We now prove the proposition: To show that each of the class of core spanners and generalized spanners is not contained in the class of context-free spanners it suffices to show that there is a that is not context-free. Indeed, the language $\{ww \mid w \in \Sigma^*\}$ which, as shown in Example 35, is definable by a Boolean core spanner is not a context-free language [23] and hence by Observation 36 is not definable by any extraction spanner. For the other direction, the language $\{\mathtt{a}^n\mathtt{b}^n | n \geq 1\}$ is context free and hence by Observation 36 is definable by an extraction grammar, nevertheless, this language is not accepted by any Boolean core spanner [12] and not by any generalized core spanner [20]. ◀

▶ **Theorem 15.** *The class of context-free spanners is closed under union and projection, and not closed under natural join and difference.*

Proof. We show each of the claims separately:

Closure under projection and union. Let $G_X := (V, \Sigma, P, S)$ be an extraction grammar and let $X' \subseteq X$. We define a morphism $\mathsf{clr}_X : (\Sigma \cup V \cup \Gamma_X)^* \to (\Sigma \cup V \cup \Gamma_Y)^*$ where $Y := X \setminus X'$ by $\mathsf{clr}_X(\sigma) = \sigma$ for $\sigma \in \Sigma$, and $\mathsf{clr}_X(\gamma) = \gamma$ for $\gamma \in \Gamma_Y$ and $\mathsf{clr}_X(\gamma) = \epsilon$ for $\gamma \in \Gamma_{X'}$. We define the grammar $G' = (V', \Sigma', P', S')$ where $V' := V, \Sigma' := \Sigma, S' := S$ and P' is defined as follows: for every $A \to \alpha \in P$ we have $A \to \mathsf{clr}_X(\alpha) \in P'$. It is straightforward that $[\![G']\!] = [\![\pi_{X'} G_X]\!]$. Closure under union can be obtained by the same technique used for unifying grammars. Let G_1, G_2 be two extraction grammars with $\mathsf{Vars}(G_1) = \mathsf{Vars}(G_2)$. We can construct a grammar G with $\mathcal{R}(G) = \mathcal{R}(G_1) \cup \mathcal{R}(G_2)$. Therefore, for every $\mathbf{d}$ it holds that $\mathsf{Ref}(G, \mathbf{d}) = (\mathcal{R}(G_1) \cup \mathcal{R}(G_2)) \cap \mathsf{Ref}(\mathbf{d})$ and thus $[\![G]\!] = [\![G_1]\!] \cup [\![G_2]\!]$.

Non-closure under natural join and difference. Non-closure properties are immediate consequence of the case for context-free languages since non-deterministic automata and push-down automata can be viewed as Boolean vset-automata and Boolean pushdown automata, respectively. This is due to the fact that the definitions of Boolean pushdown automaton and non-deterministic automaton coincide, as well as those of Boolean vset-automaton and non-deterministic automaton. ◀

▶ **Proposition 16.** *For every extraction grammar G and every document $\mathbf{d}$ it holds that $[\![G]\!](\mathbf{d})$ can be computed in $O(|G|^2 + |\mathbf{d}|^{2k+3}\, k^3\, |G|)$ time where G is associated with k variables.*

Proof. This proposition relies on the celebrated Cocke-Younger-Kasami (CYK) parsing algorithm for context-free grammars [23]. Given a context-free grammar G in Chomsky Normal Form [7], as will be defined shortly, and a string w, the CYK algorithm determines whether G produces w (i.e., w is a word in the language defined by G.) A context-free grammar G is in Chomsky Normal Form (CNF) if all of its production rules are of the form $A \to BC$ where A, B, C are non-terminals or of the form $A \to \sigma$ where A is a non-terminal and σ is a terminal. The algorithm that computes $[\![G]\!](\mathbf{d})$ operates as follows: it iterates over all of the ref-words $\mathbf{r}$ that are (1) valid for the variables $\mathsf{Vars}(G)$ whose operations appear in G and (2) mapped by clr into $\mathbf{d}$. For each such ref-word, it uses CYK algorithm to determine whether it is produced by G by treating G as a standard CFG over the extended alphabet $\Sigma \cup \Gamma_X$, and after converting G into CNF. We convert G into CNF in $O(|G|^2)$. Note that there are $O(|d|^{2k})$ valid ref-words, and each is represented by a ref-word of length $O(|\mathbf{d}| + 2k)$. For every such ref-word we use CYK to check whether it belongs to G in $O((|\mathbf{d}| + 2k)^3|G|)$. Since we repeat the process for every ref-word the total complexity is $O(|G|^2 + |d|^{2k}(|\mathbf{d}| + 2k)^3|G|)$. ◀

C Appendix for Section 4

▶ **Lemma 19.** *For every extraction grammar G in CNF, every document $\mathbf{d} := \sigma_1 \cdots \sigma_n$, every non-terminal A of G, and any ref-word $\mathbf{r} \in (\Sigma \cup \Gamma_X)^*$ with $\mathsf{clr}(\mathbf{r}) = \sigma_i \cdots \sigma_j$ the following holds: $A \Rightarrow_G^* \mathbf{r}$ if and only if $A_{i,j} \Rightarrow_{G_{\mathbf{d}}}^* \mathbf{r}$*

Proof. The proof is by induction on $j - i$. In the base case we have $j = i$ and then $\mathbf{r} = \sigma_i$. By definition we have $A \Rightarrow_G^* \sigma_i$ if and only if $A_{i,i} \Rightarrow_{G_{\mathbf{d}}}^* \sigma_i$. For the induction step: If $A \Rightarrow_G^* \mathbf{r}$ with $\mathbf{r} = \sigma_i \cdots \sigma_j$ then since G is in CNF we have $A \Rightarrow_G^* BC \Rightarrow_G^* \mathbf{r}_1\mathbf{r}_2$ where $\mathsf{clr}(\mathbf{r}_1) = \sigma_i \cdots \sigma_{i+\ell}$ and $\mathsf{clr}(\mathbf{r}_2) = \sigma_{i+\ell+1} \cdots \sigma_j$. We can use the induction hypothesis to obtain that $B_{i,i+\ell} \Rightarrow_{G_{\mathbf{d}}}^* \mathbf{r}_1$ and that $C_{i+\ell+1,j} \Rightarrow_{G_{\mathbf{d}}}^* \mathbf{r}_2$. By definition of $G_{\mathbf{d}}$ we have that $A_{i,j} \Rightarrow_{G_{\mathbf{d}}}^* B_{i,i+\ell}C_{i+\ell+1,j}$ and thus we can conclude that $A_{i,j} \Rightarrow_{G_{\mathbf{d}}}^* \mathbf{r}$. For the other direction, let us assume that $A_{i,j} \Rightarrow_{G_{\mathbf{d}}}^* \mathbf{r}$. By definition $G_{\mathbf{d}}$ is in CNF as well and also by definition we have that $A_{i,j} \Rightarrow_{G_{\mathbf{d}}}^* B_{i,i+\ell}C_{i+\ell+1,j}$ where $A \Rightarrow_G^* BC$. Thus, we obtain $A_{i,j} \Rightarrow_{G_{\mathbf{d}}}^* B_{i,i+\ell}C_{i+\ell+1,j} \Rightarrow_{G_{\mathbf{d}}}^* \mathbf{r}_1\mathbf{r}_2 = \mathbf{r}$. By induction hypothesis we have $B \Rightarrow_G^* \mathbf{r}_1$ and $C \Rightarrow_G^* \mathbf{r}_2$. Combining the above we get $A \Rightarrow_G^* BC \Rightarrow_G^* \mathbf{r}_1\mathbf{r}_2 = \mathbf{r}$. ◀

▶ **Proposition 21.** *For every extraction grammar G in CNF and for every document $\mathbf{d}$, it holds that $G_{\mathbf{d}}$ can be constructed in $O(|\mathbf{d}|^3|G|)$.*

Proof. Each production rule in G has at most $|\mathbf{d}|^3$ variations in $G_{\mathbf{d}}$. Thus, the total time required to produce $G_{\mathbf{d}}$ is $O(|\mathbf{d}|^3|G|)$. ◀

▶ **Proposition 22.** *For every functional extraction grammar G and every non-terminal A of G there is a set $\mathbf{x}_A \subseteq \Gamma_X$ of variable operations such that for every ref-word $\mathbf{r}$ where $A \Rightarrow^* \mathbf{r}$ the variable operations that appear in $\mathbf{r}$ are exactly those in $\mathbf{x}_A$. Computing all sets $\mathbf{x}_A$ can be done in $O(|G|)$.*

Proof. Assume by contradiction that this is not the case. That is, there are $\mathbf{r}, \mathbf{s}$ such that $A \Rightarrow^* \mathbf{r}$ and $A \Rightarrow^* \mathbf{s}$. Since G does not contain any unreachable non-terminal and since every non-terminal of G is reachable from the initial non-terminal there are the following leftmost derivations:

$$S \Rightarrow^* \mathbf{r}' A\gamma \Rightarrow^* \mathbf{r}'\mathbf{r}\mathbf{r}''$$

and

$$S \Rightarrow^* \mathbf{s}' A\delta \Rightarrow^* \mathbf{s}'\mathbf{s}\mathbf{s}''$$

where $\mathbf{r}', \mathbf{s}', \mathbf{r}'', \mathbf{s}'' \in (\Sigma \cup \Gamma_X)^*$ and $\gamma, \delta \in (V \cup \Sigma \cup \Gamma_X)^*$. Recall that both $\mathbf{r}'\mathbf{r}\mathbf{r}''$ and $\mathbf{s}'\mathbf{s}\mathbf{s}''$ are valid and thus if the variable operations that occur in $\mathbf{r}$ and $\mathbf{s}$ are different then so do the variable operations that appear in $\mathbf{r}'\mathbf{r}''$ and in $\mathbf{s}'\mathbf{s}''$. Thus, we can replace $\mathbf{r}$ with $\mathbf{s}$ and obtain a ref-word that is not valid. This contradicts the fact that G is functional.

To compute the sets $\mathbf{x}_A$, we view the non-terminals as nodes and the transitions as edges such that $A \to BC$ implies that there is an edge from A to B, and an edge from A to C. We then run a BFS on this graph and view the output sequence of non-terminals as a topological order. We then iterate over the non-terminals in an inverse order and accumulate for each the set of variable operations that consists of its descendants. This whole process can be done in $O(|G|)$. ◀

▶ **Proposition 24.** *For every functional extraction grammar G in CNF and for every document $\mathbf{d}$, if G is unambiguous then* DECORGRMR$(G_{\mathbf{d}})$ *is unambiguous.*

Proof. It is straightforward that if G is unambiguous then so is $G_{\mathbf{d}}$ for any document $\mathbf{d}$. Assume by contradiction that DECORGRMR$(G_{\mathbf{d}})$ is not unambiguous. That is, there is a decorated word that has two parse-trees. By the definition of DECORGRMR$(G_{\mathbf{d}})$ it follows that if it is the case then $G_{\mathbf{d}}$ has two different parse-trees to the same ref-words which leads to the desired contradiction. ◀

▶ **Lemma 25.** *For every functional unambiguous extraction grammar G in CNF and for every document $\mathbf{d}$, every decorated word produced by* DECORGRMR$(G_{\mathbf{d}})$ *is valid and*

$$[\![G]\!](\mathbf{d}) = \{\mu^w \mid S \Rightarrow^*_{\text{DECORGRMR}(G_{\mathbf{d}})} w\}.$$

Proof. Let w be a decorated word such that $S \Rightarrow^*_G (\mathbf{x}_1, 1, \mathbf{y}_1) \cdots (\mathbf{x}_n, n, \mathbf{y}_n)$. Assume by contradiction that there is a variable x such that $\mu^w(x)$ is not well-defined. It follows from the construction of DECORGRMR$(G_{\mathbf{d}})$ that either the variable operation $\vdash_x$ or $\dashv_x$ appears more than once in a ref-word produced by G or that the variable operation $\vdash_x$ occurs after $\dashv_x$ in a ref-word produced by G. That is a contradiction to the fact that G is functional.

Before moving on to the second part of the theorem, we present some definitions. We define a function f from the possible parse-trees of $G_{\mathbf{d}}$ to the possible parse-trees of $\text{DECORGRMR}(G_{\mathbf{d}})$ with $\mathbf{d} := \sigma_1 \cdots \sigma_n$. We first define for every σ_i the non-terminal associated with it as the non-terminal that is the root of the maximal subtree with leaf σ_i and no other leaf from Σ. The function f replaces for every i the subtree rooted in the non-terminal $A_{i,i}$ associated with σ_i with the subtree $A_{i,i}^{\mathbf{x},\mathbf{y}} \to (\mathbf{x}, i, \mathbf{y})$ where $A_{i,i} \Rightarrow^*_{G_{\mathbf{d}}} x_1 \cdots x_\ell \sigma_i y_1 \cdots y_{\ell'}$ and $\mathbf{x} = \{x_1, \cdots, x_\ell\}$ and $\mathbf{y} = \{y_1, \cdots, y_{\ell'}\}$. By a slight abuse of notation and since we are dealing with unambiguous grammars we refer to parse-trees as the word they produce and vice-versa. Note that

Claim 1. if $\mathbf{r} \in \mathcal{R}(G_{\mathbf{d}})$ then $S \Rightarrow^*_{\text{DECORGRMR}(G_{\mathbf{d}})} f(\mathbf{r})$.

This claim can be showed by a simple induction using the definition of $\text{DECORGRMR}(G_{\mathbf{d}})$.

Claim 2. if $S \Rightarrow^*_{\text{DECORGRMR}(G_{\mathbf{d}})} (\mathbf{x}_1, 1, \mathbf{y}_1) \cdots (\mathbf{x}_n, n, \mathbf{y}_n)$ with then there is a $\mathbf{r}$ such that $\mathbf{r} \in \mathcal{R}(G_{\mathbf{d}})$ and the image under f of the parse-tree of $\mathbf{r}$ is w.

This claim can be proved by a simple induction using the definition of $\text{DECORGRMR}(G_{\mathbf{d}})$.

Claim 3. For every $\mathbf{r}$ it holds that $\mu^{\mathbf{r}} = \mu^{\mathbf{r}'}$ where $\mathbf{r}'$ is the image under f of the parse tree of $\mathbf{r}$.

This claim is a direct consequence of the definitions.

Moving back to the proof of the theorem, which is showing that

$$[\![G]\!](\mathbf{d}) = \{\mu^w \mid S \Rightarrow^*_{\text{DECORGRMR}(G_{\mathbf{d}})} w\}.$$

If $\mu \in [\![G]\!](\mathbf{d})$ then there is a ref-word $\mathbf{r}$ in $\mathcal{R}(G)$ such that $\mu = \mu^{\mathbf{r}}$. By claim 1, $S \Rightarrow^*_{\text{DECORGRMR}(G_{\mathbf{d}})} f(\mathbf{r})$. Note also that $\mathsf{clr}^{\text{DEC}}(f(\mathbf{r})) = \mathbf{d}$. Finally, by Claim 3 we can conclude the claim.

If $\mu \in \{\mu^w \mid S \Rightarrow^*_{\text{DECORGRMR}(G_{\mathbf{d}})} w\}$ then there is a decorated word w such that $\mu = \mu^w$. Claim 2 implies that there exists a $\mathbf{r} \in \mathcal{R}(G)$ such that $f(\mathbf{r}) = w$. Then, by claim 3 we conclude that $\mu^{\mathbf{r}} = \mu^w$ which completes the proof. ◀

▶ **Proposition 27.** *For every functional unambiguous extraction grammar G in CNF and for every document $\mathbf{d}$, $\text{DECORGRMR}(G_{\mathbf{d}})$ can be constructed in $O(|G_{\mathbf{d}}|\, 5^{2k}) = O(|\mathbf{d}|^3 |G|\, 5^{2k})$ where k is the number of variables associated with G.*

Proof. In Step 1 of the construction we create several copies of each production of $G_{\mathbf{d}}$. Notice that there are 5^{2k} four pairwise disjoint subsets of Γ_X (which is equivalent to the number of mappings from Γ_X to a domain of size 5 which indicates to which subset an elements belong). The size of the grammar after step 1 is $O(|G_{\mathbf{d}}| 5^{2k})$ and that is also the time complexity required for constructing it. Step 2 consists of several linear scans of the resulting grammar and hence can be done in $O(|G_{\mathbf{d}}| 5^{2k})$ time. Finally, in step 3 we remove the unit productions. For this purpose we have to compute for each non-terminal the set of its reachable non-terminal using only unit productions. The naive solution would require a quadratic computation but here we have the subscripts that give us extra information and it suffices to check for a non-terminal $A_{i,j}$ only those non-terminals that have similar subscript. Since for each i, j there are at most $O(|G| 3^{2k})$ unit productions, we can do the computation in $O(|\mathbf{d}|^2 |G| 3^{2k})$. Thus, we obtain the following complexity $O(|G_{\mathbf{d}}|\, 5^{2k}) = O(|\mathbf{d}|^3 |G|\, 5^{2k})$. ◀

D Appendix for Section 5

▶ **Lemma 29.** *For every functional extraction grammar G in CNF and for every document* $\mathbf{d}$*, the set of stable non-terminals of* $\textsc{decorGrmr}(G_{\mathbf{d}})$ *is computable in* $O(|G_{\mathbf{d}}|5^{2k})$ *where k is the number of variables G is associated with.*

Proof. Recall that in Proposition 22 we showed that we can compute the sets $\mathbf{x}_A$ for every non-terminal A in G in linear time in $|G|$. It is straightforward that the stable non-terminals of $\textsc{decorGrmr}(G_{\mathbf{d}})$ are exactly those of the form $A_{i,j}^{\mathbf{x},\mathbf{y}}$. We can scan the non-terminals of $\textsc{decorGrmr}(G_{\mathbf{d}})$ and for each such non-terminal $A_{i,j}^{\mathbf{x},\mathbf{y}}$ check in $O(1)$ whether $\mathbf{x}_a = \mathbf{x} \cup \mathbf{y}$. Therefore, the complexity of this step is $O(|G|) + O(|\textsc{decorGrmr}(G_{\mathbf{d}})|) = O(|\mathbf{d}|^3|G|\,5^{2k})$. ◀

▶ **Lemma 32.** *For every functional unambiguous extraction grammar G in CNF and for every document* $\mathbf{d}$*, the* $\textsc{jump}$ *function is computable in* $O(|\mathbf{d}|^5 3^{4k}|G|^2)$ *where k is the number of variables G is associated with.*

Proof. We claim that Algorithm 1 computes the jump function. The algorithm has access to the grammar $\textsc{decorGrmr}(G_{\mathbf{d}})$ as well as to its skippable productions (i.e., it can check in $O(1)$ whether a production is skippable or not). The algorithm returns a 5 dimensional array such that at the end of the run $\mathsf{jmp}[\mathsf{A}, \mathsf{i}, \mathsf{j}, \mathbf{x}, \mathbf{y}]$ will contain all of the non-terminals that are in the result of applying the $\textsc{jump}$ on $A_{i,j}^{\mathbf{x},\mathbf{y}}$. The algorithm uses an auxiliary array

Algorithm 1: Compute The Jump

```
procedure computeJump()
foreach non terminal A^{x,y}_{i,j} in decorGrmr(G_d) do
    reachable[A, i, j, x, y] := {A^{x,y}_{i,j}}
foreach non terminal A in decorGrmr(G_d) do
    insert A to remove
foreach production rule ρ in decorGrmr(G_d) do
    if ρ is non-skipabble with lefthand side A^{x,y}_{i,j} then
        remove := remove \ {A^{x,y}_{i,j}}
foreach non-terminal A^{x,y}_{i,j} in decorGrmr(G_d) do
    jmp[A, i, j, x, y] := {A^{x,y}_{i,j}} \ remove
initialize perform a topological sort on non-terminals;
foreach skippable ρ := A^{x,y}_{i,j} → B^{x,∅}_{i,ℓ} C^{∅,y}_{ℓ+1,j} in decorGrmr(G_d) in reverse order to
 topological sort do
    if B^{x,∅}_{i,ℓ} is non-stable then
        reachable[A, i, j, x, y] := reachable[A, i, j, x, y] ∪ reachable[B, i, ℓ, x, ∅];
        jmp[A, i, j, x, y] := reachable[A, i, j, x, y] \ remove;
    if C^{∅,y}_{ℓ+1,j} is non-stable then
        reachable[A, i, j, x, y] := reachable[A, i, j, x, y] ∪ reachable[C, ℓ + 1, j, ∅, y]
        jmp[A, i, j, x, y] := reachable[A, i, j, x, y] \ remove;
output jmp
```

$\mathsf{reachable}$ and an auxiliary set remove. The correctness of the algorithm is based on the following claims:

Claim 1: The array $\mathsf{reachable}$ stores in cell $\mathsf{reachable}[\mathsf{A}, \mathsf{i}, \mathsf{j}, \mathbf{x}, \mathbf{y}]$ all of the non-terminals B such that there is a sequence of skippable productions $\rho_1, \cdots, \rho_m$ such that:

- ρ_ι follows $\rho_{\iota-1}$ for every ι,
- the left-hand side of ρ_1 is $A_{i,j}^{\mathbf{x},\mathbf{y}}$,
- the non-stable non-terminal in the right-hand side of ρ_m is B

Claim 2: The set remove contains all of the non-terminals B for which there is no non-skippable production with B in the left-hand side. That is, all of the productions that have B in their left-hand side are skippable.

This two claims allows us to conclude the following. Since both $B \in \mathsf{reachable}[\mathsf{A}, \mathsf{i}, \mathsf{j}, \mathbf{x}, \mathbf{y}]$ and $B \notin \mathsf{remove}$, there is a sequence of skippable productions $\rho_1, \cdots, \rho_m$ such that:

- ρ_ι follows $\rho_{\iota-1}$ for every ι,
- the left-hand side of ρ_1 is $A_{i,j}^{\mathbf{x},\mathbf{y}}$,
- the non-stable non-terminal in the right-hand side of ρ_m is B,
- there is *no* production rule that is not skippable whose left-hand side is B.

This, in turn, completes the proof of correctness of the algorithm.

We shall now explain why both claims hold. While Claim 2 is derived directly from the way we initialize remove, the proof of Claim 1 is more involved as we now explain. Since $\textsc{decorGrmr}(G_{\mathbf{d}})$ produces only terminal decorated words of fixed length $|\mathbf{d}|$ (see Lemma 25) and since it is in CNF, we can conclude that the longest derivation sequence of $\textsc{decorGrmr}(G_{\mathbf{d}})$ is of length $O(|\mathbf{d}|)$. Therefore, the correctness of the claim is obtained from the fact that we iterate over the skippable derivation in a reverse order to the topological order.

Complexity. We assume that we can check in $O(1)$ whether a non-terminal is stable. To do so, we can run the algorithm described in the proof of Lemma 29 and store the data in a lookup table with the non-terminals as keys and the value is either 'stable' or 'non-stable'. The runtime of this procedure is $O(|G_{\mathbf{d}}|5^{2k})$.

We also assume that we can check in $O(1)$ whether a rule in $\textsc{decorGrmr}(G_{\mathbf{d}})$ is skippable. To do so, we can use the above look-up table and for each rule check whether all of the conditions in the definition hold in $O(1)$. We can store this information in a lookup table with the rules as keys and the value is either 'skippable' or 'non-skippable'. The runtime of this procedure is also $O(|G_{\mathbf{d}}|5^{2k})$.

The four first for each loops are used for initializations and require $O(|G_{\mathbf{d}}|5^{2k}) = O(|\mathbf{d}|^3|G|5^{2k})$ as the iterate through all production rules of $\textsc{decorGrmr}(G_{\mathbf{d}})$. We then preform a topological sort on the production rules in $O(|\mathbf{d}|^3|G|5^{2k})$.

The last for each loop iterates through all of the skippable rules of $\textsc{decorGrmr}(G_{\mathbf{d}})$ ordered by their head in a reverse order to that we have obtained in the topological sort we performed. There are at most $O(|\mathbf{d}|^3|G|3^{2k})$ such rules (since we their form is $A_{i,j}^{\mathbf{x},\mathbf{y}} \to B_{i,\ell}^{\mathbf{x},\emptyset} C_{\ell+1,j}^{\emptyset,\mathbf{y}}$). In each such iteration, we compute set unions and set difference. The size of the set is bounded by $O(|\mathbf{d}|^2 3^{2k}|G|)$. Thus, the total time complexity of the second for loop is $O(|\mathbf{d}|^5 3^{4k}|G|^2)$. And finally, the total complexity of the $\textsc{jump}$ is $O(|\mathbf{d}|^5 3^{4k}|G|^2)$. ◀

D.1 Proof of Theorem 34

▶ **Theorem 34.** *For every functional unambiguous extraction grammar G in CNF and for every document $\mathbf{d}$, the main enumerating algorithm described above enumerates the mappings* *in* $[\![G]\!](\mathbf{d})$ *(without repetitions) with delay of* $O(k)$ *between each two consecutive mappings where* k *is the number of variables* G *is associated with.*

In what follows, we present each $(X, \mathbf{d})$-mapping μ as the set $\{(\vdash_x, i), (\dashv_x, j) | \mu(x) := [i, j\rangle\}$ of pairs. Before proving the theorem itself we prove some lemmas.

We say that a mapping A is compatible with a set B of pairs of the form (τ, i) with $\tau \in \Gamma_X$ and $1 \leq i \leq n$ if $B \subseteq A$.

▶ **Lemma 37.** *The procedure* ENUMERATE$(S^{\mathbf{x},\mathbf{y}}_{1,n}, \mathsf{map})$ *where* $\mathsf{map} = \{(x, 1) \mid x \in \mathbf{x}\} \cup \{(y, n+1) \mid y \in \mathbf{y}\}$ *outputs every mapping* μ^w *that corresponds with a decorated word* w *produced by* DECORGRMR$(G, \mathbf{d})$ *and that is compatible with* map*.*

Proof. Let w be a decorated word such that $\gamma_1 \Rightarrow \cdots \Rightarrow \gamma_m \Rightarrow w$ in a leftmost derivation where every $\gamma_i \neq \gamma_j$ and $\gamma_1 = S^{\mathbf{x},\mathbf{y}}_{1,n}$. We recursively define $\mu^{w,j}$ as follows: $\mu^{w,m}$ is map. We denote the production rule applied in $\gamma_\ell \Rightarrow \gamma_{\ell+1}$ by $A^{\mathbf{x},\mathbf{y}}_{i,j} \rightarrow B^{\mathbf{x},\mathbf{z}}_{i,i'} C^{\mathbf{w},\mathbf{y}}_{i'+1,j}$ and $\mu^{w,\ell}$ is the union of $\mu^{w,\ell+1}$ with $\{(\tau, i'+1) | \tau \in \mathbf{z} \cup \mathbf{w}\}$. We say that $A^{\mathbf{x},\mathbf{y}}_{i,j} \rightarrow B^{\mathbf{x},\mathbf{z}}_{i,i'} C^{\mathbf{w},\mathbf{y}}_{i'+1,j}$ is *productive* if $\mu^{w,\ell}$ strictly contains $\mu^{w,\ell+1}$. We define the morphism NONSTABLE that maps stable non-terminal and non-terminals into ϵ and acts as the identity on the non-stable non-terminals.

We prove that if $\gamma_\ell \Rightarrow^* w$ where γ_ℓ is a sequence over the non-terminals and terminals then ENUMERATE(NONSTABLE$(\gamma_\ell), \emptyset)$ returns the mapping $\mu^{w,\ell}$. The proof is done by a nested induction: on $\ell - i$ for $i = 0, \cdots \ell - 1$ and an inner induction on the number of productive productions applied throughout $\gamma_\ell \Rightarrow^* w$. If there were no productive productions applied then $\mu^{w,\ell} = \emptyset$ and, in addition, this implies that NONSTABLE$(\gamma_\ell) = \epsilon$. Indeed ENUMERATE$(\epsilon, \emptyset)$ returns $\emptyset$ which completes the induction basis. For the induction step: Let us assume that there is one or more productive production applied in $\gamma_\ell \Rightarrow^* w$. This implies that there is at least one non-stable non-terminal in γ_ℓ. Let us find the left most non-stable non-terminal $A^{\mathbf{x},\mathbf{y}}_{i,j}$ in γ_ℓ and denote $\gamma_\ell = \alpha A \beta$. We can write

$$\alpha A \beta \Rightarrow^* \alpha' A^{\mathbf{x},\mathbf{y}}_{i,j} \beta' \Rightarrow \alpha' B^{\mathbf{x},\mathbf{z}}_{i,i'} C^{\mathbf{w},\mathbf{y}}_{i'+1,j} \beta' \Rightarrow^* w$$

where no productive production was applied here $\alpha A \beta \Rightarrow^* \alpha' A^{\mathbf{x},\mathbf{y}}_{i,j} \beta'$ and $A^{\mathbf{x},\mathbf{y}}_{i,j} \rightarrow B^{\mathbf{x},\mathbf{z}}_{i,i'} C^{\mathbf{w},\mathbf{y}}_{i'+1,j}$ is a productive production. We then denote $\gamma_\ell = \alpha A \beta$, $\gamma_{\ell'} = \alpha' A^{\mathbf{x},\mathbf{y}}_{i,j} \beta'$, and $\gamma_{\ell'+1} = \alpha' B^{\mathbf{x},\mathbf{z}}_{i,i'} C^{\mathbf{w},\mathbf{y}}_{i'+1,j} \beta'$. We distinguish between two cases:

- If A is a non-stable non-terminal then it holds that $A^{\mathbf{x},\mathbf{y}}_{i,j} \in$ JUMP(A) and then we consider the run ENUMERATE(NONSTABLE$(\alpha A \beta), \emptyset)$ in which we enter the outer for loop with $A^{\mathbf{x},\mathbf{y}}_{i,j}$ and choose from the output of APPLYPROD the pair (β', map') that corresponds with the iteration in APPLYPROD that matches $A^{\mathbf{x},\mathbf{y}}_{i,j} \rightarrow B^{\mathbf{x},\mathbf{z}}_{i,i'} C^{\mathbf{w},\mathbf{y}}_{i'+1,j}$. Note that in this case map' is the union $\{(\tau, i'+1) | x \in \mathbf{z} \cup \mathbf{w}\}$. We then continue with calling ENUMERATE(NONSTABLE$(\gamma_{\ell'} + 1), \mathsf{map}')$. By induction hypothesis and by the definition of ENUMERATE (and in particular the fact that it accumulates the output mapping throughout the run), it holds that ENUMERATE(NONSTABLE$(\alpha A \beta), \emptyset)$ returns $\mu^{w,\ell}$.
- If A is a stable non-terminal then it holds that $A^{\mathbf{x},\mathbf{y}}_{i,j} \in$ JUMP(A') where A' is the first non-stable non-terminal to the right of A. In this case, the proof continues similarly.

We can finally conclude the desired claim. ◀

▶ **Lemma 38.** *The procedure* ENUMERATE$(S^{\mathbf{x},\mathbf{y}}_{1,n})$ *outputs every mapping* μ^w *that corresponds with a decorated word* w *produced by* DECORGRMR$(G, \mathbf{d})$ *exactly once.*

Proof. Assume to the contrary that there is a mapping μ that is outputted twice. Let us consider the two different call stacks of the procedure ENUMERATE and let us examine the first point where they differ. That is, in one stack ENUMERATE(α, map) has called

ENUMERATE(α', map') and in the other stack ENUMERATE(α, map) has called the procedure ENUMERATE(α'', map''). This, in turn, implies that the APPLYPROD returned two pairs (β, map) and (β', map') that are different. Note that for any two pairs (β, map) and (β', map') in the output of APPLYPROD it holds that both $\mathsf{map} \neq \emptyset$ and $\mathsf{map}' \neq \emptyset$, and $\mathsf{map} \neq \mathsf{map}'$ since DECORGRMR($G_{\mathbf{d}}$) is unambiguous. This, combined with the fact that ENUMERATE accumulates its output mapping throughout the run, we conclude that the mappings in both runs would be different. That is a contradiction. ◀

▶ **Proposition 39.** *The procedure* APPLYPROD *is always called by* ENUMERATE *with a non-terminal A that is non-stable and appears in the lefthand side of some non-skippable production.*

Proof. This is straightforward from the definition of JUMP and the fact that APPLYPROD is called on a non-terminal in the image of JUMP. Note that JUMP never maps a non-terminal into the empty set. ◀

▶ **Proposition 40.** *The following hold:*

- *The procedure* APPLYPROD *always returns at least one pair (β, map) with $\mathsf{map} \neq \emptyset$.*
- *For every pair (β, map) that is in the output of* APPLYPROD *it holds that $\mathsf{map} \neq \emptyset$.*

Proof.
- Straightforward from Proposition 39.
- If APPLYPROD is called with $A_{i,j}^{\mathbf{x},\mathbf{y}}$ then by Proposition 39, it appears in the lefthand side of a non-skippable rule $A_{i,j}^{\mathbf{x},\mathbf{y}} \to B_{i,\ell}^{\mathbf{x},\mathbf{z}} C_{\ell+1,j}^{\mathbf{w},\mathbf{y}}$. Since the rule is not skippable it holds that $\mathbf{z} \cup \mathbf{w} \neq \emptyset$ which concludes the desired claim.

  This concludes the proof. ◀

▶ **Lemma 41.** *The procedure* ENUMERATE($S_{1,n}^{\mathbf{x},\mathbf{y}}$) *outputs with $O(k)$ delay between two consecutive mappings.*

Proof. The size of each mapping that is outputted by ENUMERATE is $O(k)$. Due to Proposition 40, before each recursive call to ENUMERATE the mapping grows in at least one pair. Therefore the stack call of ENUMERATE is at most $k+1$. Note that when the stack call of enum is of depth i then in the last call to ENUMERATE (the top of the stack) the first element passed to it contains at most $k-i$ non-stable non-terminals (since the mappings are always of fixed size k). ◀

Algorithm 2: Apply Production Algorithm

---

procedure $\textsc{applyProd}(A^{\mathbf{x},\mathbf{y}}_{i,j})$
initialize $\mathsf{map} = \emptyset$;
foreach *non-skippable production of the form* $A^{\mathbf{x},\mathbf{y}}_{i,j} \to B^{\mathbf{x},\mathbf{z}}_{i,\ell} C^{\mathbf{w},\mathbf{y}}_{\ell+1,j}$ **do**
    $\mathsf{map} = \emptyset$;
    foreach $x \in \mathbf{z} \cup \mathbf{w}$ **do**
        $\mathsf{map} = \mathsf{map} \cup \{(x, \ell+1)\}$
    if $B^{\mathbf{x},\mathbf{z}}_{i,\ell}$ *and* $C^{\mathbf{w},\mathbf{y}}_{\ell+1,j}$ *are non-stable* **then**
        $\beta = B^{\mathbf{x},\mathbf{z}}_{i,\ell} C^{\mathbf{w},\mathbf{y}}_{\ell+1,j}$
    if $B^{\mathbf{x},\mathbf{z}}_{i,\ell}$ *is non-stable and* $C^{\mathbf{w},\mathbf{y}}_{\ell+1,j}$ *is stable* **then**
        $\beta = B^{\mathbf{x},\mathbf{z}}_{i,\ell}$
    if $B^{\mathbf{x},\mathbf{z}}_{i,\ell}$ *is stable and* $C^{\mathbf{w},\mathbf{y}}_{\ell+1,j}$ *is non-stable* **then**
        $\beta = C^{\mathbf{w},\mathbf{y}}_{\ell+1,j}$
    if $B^{\mathbf{x},\mathbf{z}}_{i,\ell}$ *and* $C^{\mathbf{w},\mathbf{y}}_{\ell+1,j}$ *are stable* **then**
        $\beta = \epsilon$
    output (β, map)
