Theory Comput Syst (2018) 62:854–898
DOI 10.1007/s00224-017-9770-0

# Document Spanners: From Expressive Power to Decision Problems

**Dominik D. Freydenberger**[1] · **Mario Holldack**[2]

Published online: 22 May 2017


**Abstract** We examine *document spanners*, a formal framework for information extraction that was introduced by Fagin, Kimelfeld, Reiss, and Vansummeren (PODS 2013, JACM 2015). A document spanner is a function that maps an input string to a relation over *spans* (intervals of positions of the string). We focus on document spanners that are defined by *regex formulas*, which are basically regular expressions that map matched subexpressions to corresponding spans, and on *core spanners*, which extend the former by standard algebraic operators and string equality selection. First, we compare the expressive power of core spanners to three models – namely, *patterns*, *word equations*, and a rich and natural subclass of *extended regular expressions* (regular expressions with a repetition operator). These results are then used to analyze the complexity of query evaluation and various aspects of static analysis of core spanners. Finally, we examine the relative succinctness of different kinds of representations of core spanners and relate this to the simplification of core spanners that are extended with difference operators.

**Keywords** Information extraction · Document spanners · Regular expressions · Xregex · Patterns · Word equations · Decision problems · Descriptional complexity

This article is part of the Topical Collection on *Special Issue on Database Theory*

An preliminary version of this article appeared as [14]. Dominik D. Freydenberger was supported by Deutsche Forschungsgemeinschaft (DFG) under grant FR 3551/1-1.

✉ Dominik D. Freydenberger
ddfy@ddfy.de

[1] Loughborough University, Loughborough, UK

[2] Goethe University, Frankfurt am Main, Germany

## 1 Introduction

Information Extraction (IE) is the task of automatically extracting structured information from texts. This paper examines *document spanners* (also called *spanners*), a formalization of the IE query language AQL, which is used in IBM's SystemT. Document spanners were introduced by Fagin et al. [7] in order to allow the theoretical examination of AQL, and were also used in [8].

A *span* is an interval on positions of a string $w$, and a *spanner* is a function that maps $w$ to a relation over spans of $w$. A central topic of [7] and of the present paper are *core spanners* (according to Fagin et al., this name was chosen because core spanners capture the core of AQL).

The primitive building blocks of core spanners are *regex formulas*, which are regular expressions with variables. Each of these variables corresponds to a subexpression, and whenever a regex formula $\alpha$ matches a string $w$, each variable is mapped to the span in $w$ that matches that subexpression. For example, consider the regex formula $\alpha := x\{\mathtt{aaa}\} \cdot \mathtt{a}^+ \cdot y\{\mathtt{a}^+\}$, with terminal $\mathtt{a}$, and variables $x$ and $y$. When $\alpha$ matches a string $w$, it maps $x$ to the span that contains the first three positions of $w$, and $y$ to a span from some position after the third to the last position of $w$. Hence, each match of $\alpha$ on $w$ determines a tuple of spans; and as there can be multiple matches of a regex formula to a string, this process creates a relation over spans of $w$. Core spanners are then defined by extending regex formulas with the relational operations projection, union, natural join, and string equality selection.

One of the two main topics of the present paper is the examination of decision problems for core spanners, in particular evaluation and static analysis. These results are mostly derived from the other main topic, the examination of the expressive power of core spanners in relation to three other models that use repetition operators, which act similar to the spanners' string equality selection.

We begin with comparing core spanners to *patterns*. A pattern is word that consists of variables and terminals, and generates the language of all words that can be obtained by substitution of the variables with arbitrary terminal words. For example, the pattern $\alpha = xx\mathtt{a}\mathtt{b}y$ (where $x$ and $y$ are variables, and $\mathtt{a}$ and $\mathtt{b}$ are terminals) generates the language of all words that have a prefix that consists of a square, followed by the word $\mathtt{ab}$. Although pattern languages have a simple definition, various decision problems for them are surprisingly hard. For example, their membership problem is NP-complete (cf. Angluin [1], Jiang et al. [24]), and their inclusion problem is undecidable (cf. Bremer and Freydenberger [4]). As we show that core spanners can recognize pattern languages, this allows us to conclude that evaluation of Boolean core spanners is NP-hard, and that spanner containment is undecidable.

Next, we consider *word equations*, which are equations of the form $\alpha = \beta$, where $\alpha$ and $\beta$ are patterns. Word equations can be used to define languages and word relations. We show that word equations with regular constraints can express all relations that are expressible with core spanners. By using an improved version of Makanin's algorithm (cf. Diekert [6]), this allows us to show that satisfiability and hierarchicality for core spanners can be decided in PSPACE. Moreover, using coding techniques from word equations, we show that two common relations from combinatorics on words can be selected with core spanners.

Finally, we examine the relation of core spanners to *xregexes* (also called *extended regular expressions*, *regexes*, or *regular expressions with backreferences* in literature). These are regular expressions that can use a repetition operator, that is available in most modern implementations for regular expressions (see, e. g., Friedl [17]) and that allows the definition of non-regular languages. For example, the xregex $x\{\Sigma^*\} \cdot \&x \cdot \&x$ generates all cubic words over $\Sigma$, as $x\{\Sigma^*\}$ generates some word $w$ which is stored in the variable $x$, and each occurrence of $\&x$ repeats that $w$. As a consequence of this increase in expressive power, many decision problems are harder for xregexes than for their "classical" counterparts. In particular, various problems of static analysis are undecidable (Freydenberger [12]).

But as shown by Fagin et al. [7], core spanners cannot define all languages that are definable by xregexes. Intuitively, the reason for this is that xregexes can use their repetition operators inside a Kleene star, which allows them to repeat an arbitrary word an unbounded number of times – for example, the xregex $x\{\Sigma^*\}\cdot\&x^+$ generates the language of all $w^n$, $n \geq 2$. In contrast to this, core spanners have to express repetitions with variables and string equality selections. Inspired by this observation, we introduce *variable-star free (or vstar-free) xregexes* as those xregexes that neither define nor use variables inside a Kleene star. We show that every vstar-free xregex can be converted into an equivalent core spanner. Since all undecidability results by Freydenberger [12] also apply to vstar-free xregexes, these undecidability results carry over to core spanners. This also has various consequences for the minimization and the relative succinctness of classes of spanner representations. We also show that complementing a core spanner can lead to a size increase that is not bounded by any recursive function (for basically all natural notions of size). Although this does not solve an open problem by Fagin et al. [7] on the simplification of core spanners with difference operators, it shows that if simplification is possible, it has to be non-computable. As a further contribution, we also develop tools to prove inexpressibility for vstar-free regular expressions and for core spanners.

As we shall see, many of the observed lower bounds hold even for comparatively restricted classes of core spanners (in particular, most of the results hold for spanners that do not use join). Hence, the authors consider it reasonable to expect that these results can be easily adapted to other information extraction languages that combine regular expressions with capture variables and a string equality operator.

In addition to regex formulas, Fagin et al. [7] also consider two types of automata as basic building blocks of spanner representations. While the present paper does not discuss these in detail, most of the results on spanner representations that are based on regex formulas can be directly converted to the respective class of spanner representations that are based on automata.

**Related Work** For an overview of related models, we refer to Fagin et al. [7]. In addition to this, we highlight connections to models with similar properties. In [7], Fagin et al. showed that there is a language that can be defined by xregexes, but not by core spanners. Furthermore, they compared the expressive power of core spanners and a variant of conjunctive regular path queries (CRPQs), a graph querying language. Barceló et al. [2] introduced extended CRPQs (ECRPQs), which can compare paths in the graph with regular relations. While there is no direct connection

between ECRPQs and core spanners, both models share the basic idea of combining regular languages with a comparison operator that can express string equality. As shown by Freydenberger and Schweikardt [16], ECRPQs have undecidability results that are comparable to those in the present paper, and to those for xregexes (cf. Freydenberger [12]). Furthermore, Barceló and Muñoz [3] have used word equations with regular constraints for variants of CRPQs.

Also note that Freydenberger [13] extends the results on the connection between word equations and core spanners from the present paper into a logic on words that has the same expressive power as core spanners.

**Structure of the Paper** In Section 2, we give definitions of xregexes and of core spanners. Section 3 compares the expressive power of core spanners to patterns, word equations, and vstar-free regular expressions. The results from this section are then used in Section 4 to examine the complexity of evaluation and static analysis of spanners. We also examine the consequences of these results to the relative succinctness of different spanner representations. Section 5 concludes the paper.

## 2 Preliminaries

Let $\mathbb{N}$ and $\mathbb{N}_{>0}$ be the sets of non-negative and positive integers, respectively. Let $\Sigma$ be a fixed finite alphabet of *(terminal) symbols*. Except when stated otherwise, we assume $|\Sigma| \geq 2$. We use $\varepsilon$ to denote the *empty word*. For every word $w \in \Sigma^*$ and every $a \in \Sigma$, let $|w|$ denote the length of $w$, and $|w|_a$ the number of occurrences of $a$ in $w$. A word $x \in \Sigma^*$ is a *subword* of a word $y \in \Sigma^*$ if there exist $u, v \in \Sigma^*$ with $y = uxv$. A word $x \in \Sigma^*$ is a *prefix* of a word $y \in \Sigma^*$ if there exists a $v \in \Sigma^*$ with $y = xv$, and a *proper prefix* if it is a prefix and $x \neq y$. For every $n \in \mathbb{N}$, an *n-ary word relation (over $\Sigma$)* is a subset of $(\Sigma^*)^n$.

### 2.1 Regexes (Extended Regular Expressions)

This section introduces the syntax and semantics of xregexes, which we shall also use for regex formulas in Section 2.2. We begin with the syntax, which follows the definition from [7].

**Definition 2.1** We fix an infinite set $X$ of *variables* and define the set $M$ of *meta symbols* as $M := \{\varepsilon, \emptyset, (, ), \{, \}, \cdot, \vee, ^*, \&\}$. Let $\Sigma$, $X$, and $M$ be pairwise disjoint. The set of *xregexes (extended regular expressions)* is defined as follows:

1. The symbols $\emptyset$ and $\varepsilon$, and every $a \in \Sigma$ are xregexes.
2. If $\alpha_1$ and $\alpha_2$ are xregex, then $(\alpha_1 \cdot \alpha_2)$ *(concatenation)*, $(\alpha_1 \vee \alpha_2)$ *(disjunction)*, and $(\alpha_1^*)$ *(Kleene star)* are xregexes.
3. For every $x \in X$ and every xregex $\alpha$ that contains neither $x\{\cdots\}$ nor $\&x$ as a subword, $x\{\alpha\}$ is an xregex *(variable binding)*.
4. For every $x \in X$, we have that $\&x$ is an xregex *(variable reference)*.

If a subword $\beta$ of an xregex $\alpha$ is an xregex itself, we call $\beta$ a *subexpression (of $\alpha$)*. The set of all subexpressions of $\alpha$ is denoted by $\mathsf{Sub}(\alpha)$, and the set of variables occurring in variable bindings in an xregex $\alpha$ is denoted by $\mathsf{Vars}(\alpha)$. If an xregex $\alpha$ contains neither variable references, nor variable bindings, we call $\alpha$ a *proper regular expression*.

In other words, we use the term "proper" to distinguish those expressions that are usually just called "regular expressions" from the more general extended regular expressions. We use the notation $\alpha^+$ as a shorthand for $\alpha \cdot \alpha^*$. Parentheses can be added freely. We may also omit parentheses and the concatenation operator, where we assume $*$ and $+$ are taking precedence over concatenation, and concatenation precedes disjunction. Furthermore, we use $\Sigma$ as a shorthand for the regular expression $\bigvee_{a \in \Sigma} a$.

Before introducing the semantics of xregexes formally, we give an intuitive explanation. An expression of the form $\alpha = x\{\beta\}$ matches the same strings as $\beta$, but $\alpha$ additionally stores the matched string in the variable $x$. Using a variable reference $\&x$, this string can then be repeated. For example, let $\alpha := (x\{\Sigma^*\} \cdot \&x)$. The subexpression $x\{\Sigma^*\}$ matches any string $w \in \Sigma^*$ and stores this match in $x$. The following variable reference $\&x$ repeats the stored $w$. Thus, $\alpha$ defines the (non-regular) *copy-language* $\{ww \mid w \in \Sigma^*\}$.

The following definition of the semantics of xregexes is based on the semantics by Freydenberger [12], which is an adaption of the semantics from Câmpeanu et al. [5] (the former uses variables, the latter backreferences). In comparison to [12], the case for Kleene star has been changed, in order to make the definition compatible with the parse trees for regex formulas from Fagin et al. [7].

**Definition 2.2** Let $\gamma$ be an xregex over $\Sigma$ and $X$. A *$\gamma$-parse tree* is a finite, directed, and ordered tree $T_\gamma$. Its nodes are labeled with tuples of the form $(w, \gamma') \in (\Sigma^* \times \mathsf{Sub}(\gamma))$. The root of every $\gamma$-parse tree $T_\gamma$ is labeled $(w, \gamma)$ with $w \in \Sigma^*$; and the following rules must hold for each node $v$ of $T_\gamma$:

1) If $v$ is labeled $(w, a)$ with $a \in (\Sigma \cup \{\varepsilon\})$, then $v$ is a leaf, and $w = a$.
2) If $v$ is labeled $(w, (\beta_1 \cdot \beta_2))$, then $v$ has exactly one left child $v_1$ and exactly one right child $v_2$ with respective labels $(w_1, \beta_1)$ and $(w_2, \beta_2)$, and $w = w_1 w_2$.
3) If $v$ is labeled $(w, (\beta_1 \vee \beta_2))$, then $v$ has a single child, labeled $(w, \beta_1)$ or $(w, \beta_2)$.
4) If $v$ is labeled $(w, \beta^*)$, then one of the following cases holds: (a) $w = \varepsilon$, and $v$ is a leaf, or (b) $w = w_1 w_2 \dots w_k$ for words $w_1, \dots, w_k \in \Sigma^+$ (with $k \geq 1$), and $v$ has $k$ children $v_1, \dots, v_k$ (ordered from left to right) that are labeled $(w_1, \beta)$, $\dots$, $(w_k, \beta)$.
5) If $v$ is labeled $(w, x\{\beta\})$, then $v$ has a single child, labeled $(w, \beta)$.
6) If $v$ is labeled $(w, \&x)$, let $\prec$ denote the post-order of the nodes of $T_\gamma$ (that results from a left-to-right, depth-first traversal). Then one of the following cases applies: (a) If there is no node $v'$ with $v' \prec v$ that is labeled $(w', x\{\beta'\}) \in \Sigma^* \times \mathsf{Sub}(\gamma)$, then $v$ is a leaf, and $w = \varepsilon$. (b) Otherwise, let $v'$ be the node with $v' \prec v$ that is $\prec$-maximal among nodes labeled $(w', x\{\beta'\})$. Then $v$ is a leaf, and $w = w'$.

If the root of a $\gamma$-parse tree $T_\gamma$ is labeled $(w, \gamma)$, we call $T_\gamma$ a *$\gamma$-parse tree for $w$*. If the context is clear, we omit $\gamma$ and call $T_\gamma$ a parse tree.

There is no parse tree for $\emptyset$, and references to unbound variables (i. e., variables that were not assigned a value with a variable binding operator) default to $\varepsilon$. For an example of a parse tree, see Fig. 1.

We use parse trees to define the semantics of xregexes:

**Definition 2.3** An xregex $\gamma$ *recognizes* the language $\mathcal{L}(\gamma)$ of all $w \in \Sigma^*$ for which there exists a $\gamma$-parse tree $T_\gamma$ with $(w, \gamma)$ as root label.

*Example 2.4* Consider the xregexes $\alpha := x\{\Sigma^+\} \cdot (\&x)^+$, $\beta := x\{\Sigma^+\} \cdot \&x \cdot x\{\Sigma^+\} \cdot \&x$, and $\gamma := x\{aa^+\} \cdot (\&x)^+$ for some $a \in \Sigma$.

Then $\mathcal{L}(\alpha) = \{w^n \mid w \in \Sigma^+, n \geq 2\}$, $\mathcal{L}(\beta) = \{x_1 x_1 x_2 x_2 \mid x_1, x_2 \in \Sigma^+\}$, and $\mathcal{L}(\gamma) = \{a^n \mid n \geq 2, n \text{ is not prime}\}$.

## 2.2 Document Spanners

Let $w := a_1 a_2 \cdots a_n$ be a word over $\Sigma$, with $n \in \mathbb{N}$ and $a_1, \ldots, a_n \in \Sigma$. A *span of* $w$ is an interval $[i, j\rangle$ with $1 \leq i \leq j \leq n + 1$ and $i, j \in \mathbb{N}$. For each span $[i, j\rangle$ of $w$, we define a subword $w_{[i,j\rangle} := a_i \cdots a_{j-1}$. In other words, each span describes a subword of $w$ by its bounding indices. Two spans $[i, j\rangle$ and $[i', j'\rangle$ of $w$ are equal if and only if $i = i'$ and $j = j'$. These spans *overlap* if $i \leq i' < j$ or $i' \leq i < j'$, and are *disjoint*, otherwise. The span $[i, j\rangle$ *contains* the span $[i', j'\rangle$ if $i \leq i' \leq j' \leq j$. The *set of all spans of* $w$ is denoted by $\mathsf{Spans}(w)$.

*Example 2.5* Let $w := \mathtt{aabbcabaa}$. As $|w| = 9$, both $[1, 3\rangle$ and $[8, 10\rangle$ are spans of $w$, but $[10, 11\rangle$ is not. Although $w_{[1,3\rangle} = w_{[8,10\rangle} = \mathtt{aa}$, the first two spans are not equal. Likewise, the two spans $[3, 3\rangle$ and $[5, 5\rangle$ are not equal, even though $w_{[3,3\rangle} = w_{[5,5\rangle} = \varepsilon$. The whole word $w$ is described by the span $[1, 10\rangle$.

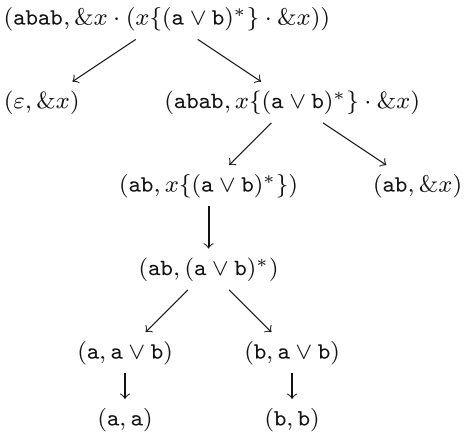



**Fig. 1** An $\alpha$-parse tree for $w$, where $\alpha := \&x \cdot (x\{(\mathtt{a} \vee \mathtt{b})^*\} \cdot \&x)$ and $w := \mathtt{abab}$. For these choices of $\alpha$ and $w$, this is the only possible parse tree

**Definition 2.6** Let $\mathsf{SVars}$ be a fixed, infinite set of *span variables*, where $\Sigma$ and $\mathsf{SVars}$ are disjoint. Let $V \subset \mathsf{SVars}$ be a finite subset of $\mathsf{SVars}$, and let $w \in \Sigma^*$. A $(V, w)$-*tuple* is a function $\mu\colon V \to \mathsf{Spans}(w)$, that maps each variable in $V$ to a span of $w$. If context allows, we write $w$-tuple instead of $(V, w)$-tuple. A set of $(V, w)$-tuples is called a $(V, w)$-*relation*.

As $V$ and $\mathsf{Spans}(w)$ are finite, every $(V, w)$-relation is finite by definition. Our next step is the definition of document spanners, which map words $w$ to $(V, w)$-relations:

**Definition 2.7** Let $V$ and $\Sigma$ be alphabets of variables and symbols, respectively. A *(document) spanner* is a function $P$ that maps every word $w \in \Sigma^*$ to a $(V, w)$-relation $P(w)$. Let $V$ be denoted by $\mathsf{SVars}(P)$. A spanner $P$ is $n$-*ary* if $|\mathsf{SVars}(P)| = n$, and *Boolean* if $\mathsf{SVars}(P) = \emptyset$. For all $w \in \Sigma^*$, we say $P(w) = \texttt{True}$ and $P(w) = \texttt{False}$ instead of $P(w) = \{()\}$ and $P(w) = \emptyset$, respectively.

A $w$-tuple $\mu \in P(w)$ is *hierarchical* if for all $x, y \in \mathsf{SVars}(P)$ at least one of the following holds: (1) The span $\mu(x)$ contains $\mu(y)$, (2) the span $\mu(y)$ contains $\mu(x)$, or (3) the spans $\mu(x)$ and $\mu(y)$ are disjoint. A spanner $P$ is *hierarchical* if, for every $w \in \Sigma^*$, every $\mu \in P(w)$ is hierarchical.

A spanner $P$ is *total on* $w$ if $P(w)$ contains all $w$-tuples over $\mathsf{SVars}(P)$. Let $Y \subset \mathsf{SVars}$ be a finite set of variables. The *universal spanner over* $Y$ is denoted by $\Upsilon_Y$. It is the unique spanner $P'$ such that $\mathsf{SVars}(P') = Y$ and $P'$ is total on every $w \in \Sigma^*$. Furthermore, a spanner $P$ is *hierarchical total on* $w$ if $P(w)$ is exactly the set of all hierarchical $w$-tuples over $\mathsf{SVars}(P)$; and the *universal hierarchical spanner* over a set $Y$ is the unique spanner $\Upsilon_Y^{\mathbf{H}}$ that is hierarchical total on every $w \in \Sigma^*$.

For two spanners $P_1$ and $P_2$, we write $P_1 \subseteq P_2$ if $P_1(w) \subseteq P_2(w)$ for every $w \in \Sigma^*$, and $P_1 = P_2$ if $P_1(w) = P_2(w)$ for every $w \in \Sigma^*$.

Hence, a spanner can be understood as a function that maps a word $w$ to a set of functions, each of which assigns spans of $w$ to the variables of the spanner. As Boolean spanners are functions that map words to truth values, they can be interpreted as characteristic functions of languages. For every Boolean spanner $P$, we define the *language recognized by* $P$ as $\mathcal{L}(P) := \{w \in \Sigma^* \mid P(w) = \texttt{True}\}$. We extend this to arbitrary spanners $P$ by $\mathcal{L}(P) := \{w \in \Sigma^* \mid P(w) \neq \emptyset\}$.

**Definition 2.8** A *regex formula* is an xregex $\alpha$ over $\Sigma$ and $X := \mathsf{SVars}$ such that $\alpha$ does not contain any variable references, and for every $\beta \in \mathsf{Sub}(\alpha)$ with $\beta = \gamma^*$, no subexpression of $\gamma$ may be a variable binding.

In other words, a regex formula is a proper regular expression that is extended with variable binding operators, but these operators may not occur inside a Kleene star. We define $\mathsf{SVars}(\gamma) := \mathsf{Vars}(\gamma)$ for all regex formulas $\gamma$.

To define the semantics of regex formulas, we use the definition of parse trees for xregexes, see Definition 2.2. Intuitively, the goal of this definition is that each occurrence of a variable $x$ in a $\gamma$-parse tree is matched to the corresponding span.

Here, two problems can arise. Firstly, a variable might not occur in the parse tree; for example, when matching the regex formula $(x\{\mathtt{a}\} \vee \mathtt{bb})$ to the word $\mathtt{bb}$. Secondly, a variable might be defined too often, as e. g. in the regex formula $x\{\Sigma^+\} \cdot x\{\Sigma^+\}$. In order to avoid such problems, we introduce the notion of a functional regex formula.

**Definition 2.9** Let $\gamma$ be a regex formula. We call $\gamma$ *functional* if for every $w \in \Sigma^*$ and every $\gamma$-parse tree $T_\gamma$ for $w$, for each variable $x \in \mathsf{SVars}(\gamma)$, there exactly one node of $T_\gamma$ has a label of the form $(v, x\{\beta\})$, where $v$ is a subword of $w$ and $\beta$ is a sub-regex formula of $\gamma$. The class of all functional regex formulas is denoted by $\mathsf{RGX}$.

As shown in Proposition 3.5 in Fagin et al. [7], functionality has a straightforward syntactic characterization: Basically, variables may not be redeclared, variables may not be used inside of Kleene stars, and if variables are used in a disjunction, each side of a disjunction has to bind exactly the same variables. Consider the following example:

*Example 2.10* The regex formula $\gamma_1 := (x\{\mathtt{a}\} \vee x\{\mathtt{b}\})$ is functional even though it contains two occurrences of variable definitions for $x$. There are just two $\gamma_1$-parse trees, both of which only contain one node labeled $(c, x\{c\})$, where $c \in \{\mathtt{a}, \mathtt{b}\}$. As a trivial case, even $\gamma_2 := x\{\emptyset\}$ is functional (as no $\gamma_2$-parse tree exists). Furthermore, the regex formulas $\gamma_3 := x\{(\mathtt{a} \vee \mathtt{b})^*\} \cdot x\{\mathtt{b}^+\}$ and $\gamma_4 := \mathtt{a}^* \vee x\{\mathtt{b}\}$ are not functional. Finally, $\gamma_5 := x\{\mathtt{a}\}^*$ is not a regex formula at all.

For functional regex formulas, we use parse trees to define the semantics:

**Definition 2.11** Let $\gamma$ be a functional regex formula and let $T$ be a $\gamma$-parse tree for a word $w \in \Sigma^*$. For every node $v$ of $T$, the subtree that is rooted at $v$ naturally maps to a span $p(v)$ of $w$. As $\gamma$ is functional, for every $x \in \mathsf{SVars}(\gamma)$, exactly one node $v_x$ of $T$ has a label that contains $x$. We define $\mu^T : \mathsf{SVars}(\gamma) \to \mathsf{Spans}(w)$ by $\mu^T(x) := p(v_x)$. Each $\gamma \in \mathsf{RGX}$ defines a spanner $[\![\gamma]\!]$ by

$$[\![\gamma]\!](w) := \{\mu^T \mid T \text{ is a } \gamma\text{-parse tree for } w\}$$

for each $w \in \Sigma^*$.

*Example 2.12* Assume that $\mathtt{a}, \mathtt{b} \in \Sigma$. We define the regex formula

$$\alpha := \Sigma^* \cdot x\{\mathtt{a} \cdot y\{\Sigma^*\} \cdot (z\{\mathtt{a}\} \vee z\{\mathtt{b}\})\} \cdot \Sigma^*.$$

Let $w := \mathtt{baaba}$. Then $[\![\alpha]\!](w)$ consists of $([2, 4\rangle, [3, 3\rangle, [3, 4\rangle)$, $([2, 5\rangle, [3, 4\rangle, [4, 5\rangle)$, $([2, 6\rangle, [3, 5\rangle, [5, 6\rangle)$, $([3, 5\rangle, [4, 4\rangle, [4, 5\rangle)$, and $([3, 6\rangle, [4, 5\rangle, [5, 6\rangle)$.

For every $w \in \Sigma^*$, a spanner $P$ defines a $(V, w)$-relation $P(w)$. In order to construct more sophisticated spanners, we introduce spanner operators.

**Definition 2.13** Let $P$, $P_1$, $P_2$ be spanners and let $w \in \Sigma^*$. The algebraic operators *union*, *projection*, *natural join* and *selection* are defined as follows.

*Union:* Two spanners $P_1$ and $P_2$ are *union compatible* if $\mathsf{SVars}(P_1) = \mathsf{SVars}(P_2)$, and their *union* $(P_1 \cup P_2)$ is defined by $\mathsf{SVars}(P_1 \cup P_2) := \mathsf{SVars}(P_1) = \mathsf{SVars}(P_2)$ and $(P_1 \cup P_2)(w) := P_1(w) \cup P_2(w)$ for every $w \in \Sigma^*$.

*Projection:* Let $Y \subseteq \mathsf{SVars}(P)$. The *projection* $\pi_Y P$ is defined by $\mathsf{SVars}(\pi_Y P) := Y$ and $\pi_Y P(w) := P(w)|_Y$ for all $w \in \Sigma^*$, where $P(w)|_Y$ is the restriction of all $w$-tuples in $P(w)$ to $Y$.

*Natural join:* Let $V_i := \mathsf{SVars}(P_i)$ for $i \in \{1, 2\}$. The *(natural) join* $(P_1 \bowtie P_2)$ of $P_1$ and $P_2$ is defined by $\mathsf{SVars}(P_1 \bowtie P_2) := \mathsf{SVars}(P_1) \cup \mathsf{SVars}(P_2)$ and, for all $w \in \Sigma^*$, we define $(P_1 \bowtie P_2)(w)$ as the set of all $(V_1 \cup V_2, w)$-tuples $\mu$ for which there exist $(V_i, w)$-tuples $\mu_1$ and $\mu_2$ with $\mu(w)|_{V_1} = \mu_1(w)$ and $\mu(w)|_{V_2} = \mu_2(w)$.

*Selection:* Let $R \subseteq (\Sigma^*)^k$ be a $k$-ary relation over $\Sigma^*$. The *selection operator* $\zeta^R$ is parameterized by $k$ variables $x_1, \ldots, x_k \in \mathsf{Vars}(P)$, written as $\zeta^R_{x_1,\ldots,x_k}$. The *selection* $\zeta^R_{x_1,\ldots,x_k} P$ is defined by $\mathsf{SVars}(\zeta^R_{x_1,\ldots,x_k} P) := \mathsf{SVars}(P)$ and, for all $w \in \Sigma^*$, we define $\zeta^R_{x_1,\ldots,x_k} P(w)$ as the set of all $\mu \in P(w)$ for which $\big(w_{\mu(x_1)}, \ldots, w_{\mu(x_k)}\big) \in R$.

Like [7], we mostly consider the *string equality selection* operator $\zeta^=$. Hence, unless otherwise noted, the term "selection" refers to selection by the $n$-ary string equality relation. Note that unlike selection (which compares strings), join requires that the spans are identical.

The join $P_1 \bowtie P_2$ of two spanners $P_1$ and $P_2$ is equivalent to the intersection $P_1 \cap P_2$ if $\mathsf{SVars}(P_1) = \mathsf{SVars}(P_2)$, and to the Cartesian Product $P_1 \times P_2$ if $\mathsf{SVars}(P_1)$ and $\mathsf{SVars}(P_2)$ are disjoint. Hence, if applicable, we write $\cap$ and $\times$ instead of $\bowtie$.

For convenience, we may add and omit parentheses. We assume there is an order of precedence with projection and selection ranking over join ranking over union, e.g. we may write $\pi_Y \zeta^=_{x,y} P_1 \cup P_2 \bowtie P_3$ instead of $(\pi_Y \zeta^=_{x,y} P_1 \cup (P_2 \bowtie P_3))$, where projection and selection are applied to $P_1$, and the result is united with the join of $P_2$ and $P_3$.

*Example 2.14* Let $P_1 := \zeta^=_{x,y} [\![x\{\Sigma^*\}y\{\Sigma^*\}]\!]$ and $P_2 := \zeta^=_{x,y,z} [\![x\{\Sigma^*\}y\{\Sigma^*\}z\{\Sigma^*\}]\!]$. Then $\mathcal{L}(P_1) = \{ww \mid w \in \Sigma^*\}$, and the variables $x$ and $y$ refer to the span of the first and second occurrence of $w$, respectively. Analogously, $\mathcal{L}(P_2) = \{w^3 \mid \in \Sigma^*\}$ (and $z$ refers to the third occurrence of $w$). Assume that we want to construct a spanner for the language $\{w^n \mid w \in \Sigma^*, n \in \{2, 3\}\}$. As $P_1$ and $P_2$ are not union compatible, we cannot simply define $P_1 \cup P_2$. Union compatibility can be achieved by projecting $P_2$ onto the set of common variables (i. e., $\pi_{\{x,y\}} P_2$).

**Definition 2.15** A *spanner algebra* is a finite set of spanner operators. If $\mathsf{O}$ is a spanner algebra, then $\mathsf{RGX}^{\mathsf{O}}$ denotes the set of all *spanner representations* that can be constructed by (repeated) combination of the symbols for the operators from $\mathsf{O}$ with regex formulas from $\mathsf{RGX}$. For each operator $o \in \mathsf{O}$ and each spanner representation of the form $o\rho$ (if $o$ is unary) or $\rho_1\ o\ \rho_2$ (if $o$ is binary), we define $[\![o\rho]\!] := o[\![\rho]\!]$ or $[\![\rho_1\ o\ \rho_2]\!] := [\![\rho_1]\!]\ o\ [\![\rho_2]\!]$, respectively. Furthermore, $[\![\mathsf{RGX}^{\mathsf{O}}]\!]$ is the closure of $[\![\mathsf{RGX}]\!]$ under the spanner operators in $\mathsf{O}$.

We define $\mathcal{L}(\rho) := \mathcal{L}([\![\rho]\!])$ for every spanner representation $\rho$. Fagin et al. [7] refer to $[\![\mathsf{RGX}]\!]$ as the class of *hierarchical regular spanners* and to $[\![\mathsf{RGX}^{\{\pi,\cup,\bowtie\}}]\!]$ as the class of *regular spanners*. In addition to (hierarchical) regular spanners, Fagin et al. also introduced the so-called *core spanners*, which are obtained by combining regex formulas with the four algebraic operators projection, selection, union, and join – in other words, the class of core spanners is the class $[\![\mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}]\!]$. Analogously, $\mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$ is the class of *core spanner representations*.

## 3 Expressibility Results

### 3.1 Pattern Languages

We begin our examination of the expressive power of core spanners by comparing them to one of the simplest mechanisms with repetition operators:

**Definition 3.1** Let $X$ be an infinite variable alphabet that is disjoint from $\Sigma$. A *pattern* is a word $\alpha \in (\Sigma \cup X)^+$ that generates the language

$$\mathcal{L}(\alpha) := \{\sigma(\alpha) \mid \sigma \text{ is a pattern substitution}\},$$

where a *pattern substitution* is a homomorphism $\sigma\colon (\Sigma \cup X)^* \to \Sigma^*$ with $\sigma(a) = a$ for all $a \in \Sigma$. We denote the set of all variables in $\alpha$ by $\mathsf{Vars}(\alpha)$.

Intuitively, a pattern $\alpha$ generates exactly those words that can be obtained by replacing the variables in $\alpha$ with terminal words homomorphically (i. e., multiple occurrences of the same variable have to be replaced in the same way). This type of pattern languages is also called *erasing pattern language* (cf. Jiang et al. [24]).

*Example 3.2* Let $x, y \in X$ and $\mathtt{a}, \mathtt{b} \in \Sigma$. The patterns $\alpha := xx$ and $\beta := x\mathtt{a}y\mathtt{b}x$ generate the languages $\mathcal{L}(\alpha) = \{ww \mid w \in \Sigma^*\}$ and $\mathcal{L}(\beta) = \{v\mathtt{a}w\mathtt{b}v \mid v, w \in \Sigma^*\}$.

From every pattern $\alpha$, we can straightforwardly construct an xregex for $\mathcal{L}(\alpha)$. A similar observation holds for core spanners:

**Theorem 3.3** *There is an algorithm that, given a pattern $\alpha$, computes in polynomial time $\rho_\alpha \in \mathsf{RGX}^{\{\zeta^=\}}$ such that $\mathcal{L}(\rho_\alpha) = \mathcal{L}(\alpha)$.*

*Proof* Let $\alpha = \alpha_1 \cdots \alpha_n$ with $n \in \mathbb{N}_{>0}$ and $\alpha_1, \ldots, \alpha_n \in (\Sigma \cup X)$. We rewrite $\alpha$ into a regex formula $\hat{\alpha}$, by replacing the $i$-th occurrence of a variable $x$ with a binding $x_i\{\Sigma^*\}$. More formally, we define $\hat{\alpha} := \hat{\alpha}_1 \cdots \hat{\alpha}_n$, where for each $i \in \{1, \ldots, n\}$, the regex formula $\hat{\alpha}_i$ is defined as follows:

1. If $\alpha_i$ is a terminal (i. e., there is an $a \in \Sigma$ with $\alpha_i = a$), let $\hat{\alpha}_i := a$.
2. If $\alpha_i$ is the $j$-th occurrence of a variable $x \in X$ in $\alpha$, let $\hat{\alpha}_i := x_j\{\Sigma^*\}$.

Hence, no variable occurs twice in $\hat{\alpha}$; and as $\hat{\alpha}$ contains no disjunctions on variables, $\hat{\alpha}$ is functional.

We now define $S$ to be a sequence of selections; where $S$ contains exactly the selections $\zeta^{=}_{x_1,\ldots,x_k}$ for each $x \in \mathsf{Vars}(\alpha)$ with $|\alpha|_x = k$ and $k \geq 2$. In other words, for each $x$ that occurs more than once in $\alpha$, we include a selection of all $x_i$.

Finally, we define $\rho_\alpha := S\hat{\alpha}$. It is easy to see that $\mathcal{L}(\rho_\alpha) = \mathcal{L}(\alpha)$: For every $w \in \mathcal{L}(\alpha)$, we can use a pattern substitution $\sigma$ with $\sigma(\alpha)$ to construct a corresponding $w$-tuple $\mu$ for $\rho_\alpha$. Likewise, for every $w \in \mathcal{L}(\rho_\alpha)$, there exists a corresponding $w$-tuple $\mu$ from which we can reconstruct a pattern substitution $\sigma$ with $\sigma(\alpha) = w$: By the construction of $\rho_\alpha$, for each pair of variables $x_i, x_j$ in $\hat{\alpha}$, the words $w_{\mu(x_i)}$ and $w_{\mu(x_j)}$ must be identical. This allows us to define $\sigma(x) := w_{\mu(x_1)}$. □

*Example 3.4* Let $x, y, z \in X$, $\mathtt{a}, \mathtt{b} \in \Sigma$, and define the pattern $\alpha := x\mathtt{a}yy\mathtt{b}xzx$. The construction in the proof of Theorem 3.3 leads to the spanner representation $\zeta^{=}_{x_1,x_2,x_3}\zeta^{=}_{y_1,y_2}\gamma$, where $\gamma = x_1\{\Sigma^*\}\cdot\mathtt{a}\cdot y_1\{\Sigma^*\}\cdot y_2\{\Sigma^*\}\cdot\mathtt{b}\cdot x_2\{\Sigma^*\}\cdot z_1\{\Sigma^*\}\cdot x_3\{\Sigma^*\}$.

While the construction in the proof of Theorem 3.3 is so simple that it might not seem noteworthy, it will prove quite useful: In contrast to their simple definition, many canonical decision problems for them are surprisingly hard. Via Theorem 3.3, the corresponding lower bounds also apply to spanners, as we discuss in Sections 4.1 and 4.2.

### 3.2 Word Equations and Existential Concatenation Formulas

In this section, we introduce word equations, which are equations of patterns (cf. Definition 3.1) and can be used to define languages and relations, cf. Karhumäki et al. [26]:

**Definition 3.5** A *word equation* is a pair $\eta := (\eta_L, \eta_R)$ of patterns $\eta_L$ and $\eta_R$. A pattern substitution $\sigma$ is a *solution* of $\eta$ if $\sigma(\eta_L) = \sigma(\eta_R)$. We define $\mathsf{Vars}(\eta) := \mathsf{Vars}(\eta_L) \cup \mathsf{Vars}(\eta_R)$. For $k \geq 1$, a relation $R \subseteq (\Sigma^*)^k$ is defined by a word equation $\eta := (\eta_L, \eta_R)$ if there exist variables $x_1, \ldots, x_k \in \mathsf{Vars}(\eta)$ such that $R = \{(\sigma(x_1), \ldots, \sigma(x_k)) \mid \sigma \text{ is a solution of } \eta\}$.

We also write $(\eta_L, \eta_R)$ as $\eta_L = \eta_R$. As we shall see just after the next definition both sides of the equation may have common variables. The following relations are well known examples of relations that are definable by word equations:

**Definition 3.6** Over $\Sigma^*$, we define relations

$$R_{\text{com}} := \{(x, y) \mid x, y \in \{u\}^* \text{ for some } u \in \Sigma^*\},$$
$$R_{\text{cyc}} := \{(x, y) \mid x \text{ is a cyclic permutation of } y\}.$$

As shown in Lothaire [30], the relation $R_{\text{com}}$ is defined by the equation $xy = yx$, and $R_{\text{cyc}}$ is defined by the equation $xz = zy$.

Let $R$ be a $k$-ary string relation, and let $C$ be a class of spanners. We say that $R$ is *selectable* by $C$, if for every spanner $P \in C$ and every sequence of variables $\mathbf{x} = (x_1, \ldots, x_k)$ with $x_1, \ldots, x_k \in \mathsf{SVars}(P)$, the spanner $\zeta^R_{\mathbf{x}} P$ is also in $C$.

**Proposition 3.7** *The relations $R_{com}$ and $R_{cyc}$ are selectable by core spanners.*

*Proof* Both parts of the proof use a technique from [7]. Let $\mathbf{x} = x_1, ..., x_k$ be a sequence of distinct span variables ($k \geq 1$), and let $X := \{x_1, \ldots, x_k\}$. The spanner $\zeta^R_{\mathbf{x}} \Upsilon_X$ is called the *$R$-restricted universal spanner* over $\mathbf{x}$, and is denoted by $\Upsilon^R_{\mathbf{x}}$. According to Proposition 4.15 in [7], in order to show that a $R$ is selectable by core spanners, it suffices to show that $\Upsilon^R_{\mathbf{x}}$ is a core spanner for every $\mathbf{x} \in \mathsf{SVars}^k$.

$R_{\text{cyc}}$: Note that for all $x, y \in \Sigma^*$, the word $x$ is a cyclic permutation of $y$ (and vice versa) if and only if there exist $u, v \in \Sigma^*$ with $x = uv$ and $y = vu$ (see e. g. Lothaire [30]). Hence we can define the core spanner $P_{\text{cyc}} := \pi_{\{x,y\}} \hat{P}$, where

$$\hat{P} := \zeta^=_{u_1,u_2} \zeta^=_{v_1,v_2} [\![\alpha_x \times \alpha_y]\!],$$

and the regex formulas $\alpha_x$ and $\alpha_y$ are defined as

$$\begin{aligned} \alpha_x &:= \Sigma^* x \left\{u_1\{\Sigma^*\} \cdot v_1\{\Sigma^*\}\right\} \Sigma^*, \\ \alpha_y &:= \Sigma^* y \left\{v_2\{\Sigma^*\} \cdot u_2\{\Sigma^*\}\right\} \Sigma^*. \end{aligned}$$

In order to prove that $P_{\text{cyc}} = \Upsilon^{R_{\text{cyc}}}_{x,y}$, we first observe that, for every $w \in \Sigma^*$ and every $\mu \in P_{\text{cyc}}(w)$, there exists a $\hat{\mu} \in \hat{P}(w)$ with $\mu(x) = \hat{\mu}(x)$ and $\mu(y) = \hat{\mu}(y)$. The selections enforce $u := w_{\hat{\mu}(u_1)} = w_{\hat{\mu}(u_2)}$ and $v := w_{\hat{\mu}(v_1)} = w_{\hat{\mu}(v_2)}$. Hence, $w_{\mu(x)} = uv$ and $w_{\mu(y)} = vu$, which means that $(w_{\mu(x)}, w_{\mu(y)}) \in R_{\text{cyc}}$, and $\mu \in \Upsilon^{R_{\text{cyc}}}_{x,y}(w)$. For the other direction, we can show analogously that every $\mu \in \Upsilon^{R_{\text{cyc}}}_{x,y}(w)$ can be extended into a $\hat{\mu} \in \hat{P}(w)$, which then proves $\mu \in P_{\text{cyc}}(w)$.

$R_{\text{com}}$: This proof relies on another fact from combinatorics on words. For all $x, y \in \Sigma^*$, the equation $xy = yx$ holds if and only if $(x, y) \in R_{\text{com}}$ (again, see Lothaire [30]). We define a core spanner $P_{\text{com}} := \pi_{\{x,y\}} \hat{P}$, where

$$\hat{P} := \zeta^=_{r_1,r_2,r_3,r_4} \zeta^=_{x,x_2} \zeta^=_{y,y_2} \zeta^=_{\hat{x},\hat{x}_2} \zeta^=_{\hat{y},\hat{y}_2} [\![\alpha_1 \times \alpha_2 \times \alpha_3 \times \alpha_4]\!],$$

and the regex formulas $\alpha_1, \ldots, \alpha_4$ are defined as

$$\begin{aligned} \alpha_1 &:= \Sigma^* x \left\{\hat{x}\{\Sigma^*\} \cdot r_1\{\Sigma^*\}\right\} \Sigma^*, \\ \alpha_2 &:= \Sigma^* x_2 \left\{r_2\{\Sigma^*\} \cdot \hat{x}_2\{\Sigma^*\}\right\} \Sigma^*, \\ \alpha_3 &:= \Sigma^* y \left\{\hat{y}\{\Sigma^*\} \cdot r_3\{\Sigma^*\}\right\} \Sigma^*, \\ \alpha_4 &:= \Sigma^* y_2 \left\{r_4\{\Sigma^*\} \cdot \hat{y}_2\{\Sigma^*\}\right\} \Sigma^*. \end{aligned}$$

In order to prove that $P_{\text{com}} = \Upsilon^{R_{\text{com}}}_{x,y}$, first assume that $\mu \in P_{\text{com}}(w)$ for some $w \in \Sigma^*$. Again, this means that there exists a $\hat{\mu} \in \hat{P}(w)$ with $\mu(x) = \hat{\mu}(x)$ and $\mu(y) = \hat{\mu}(y)$. In a slight abuse of notation, we identify the variables $x, \hat{x}, y, \hat{y}$ with the corresponding subwords of $w$. In other words, we define $x, \hat{x}, y, \hat{y} \in \Sigma^*$ by

$z := w_{\hat{\mu}(z)}$ for $z \in \{x, \hat{x}, y, \hat{y}\}$. Furthermore, let $r = w_{\hat{\mu}(r_1)}$. Due to the equality selections, we obtain the following word equations from $\alpha_1$ to $\alpha_4$:

$$x = \hat{x}r = r\hat{x},$$
$$y = \hat{y}r = r\hat{y}.$$

We explain this in detail for the first equation: First, note that due to the structure of $\alpha_1$, we know that $w_{\mu(x)} = w_{\mu(\hat{x})} \cdot w_{\mu(r_1)}$ holds. Likewise, the structure of $\alpha_2$ ensures that $w_{\mu(x_2)} = w_{\mu(r_2)} \cdot w_{\mu(\hat{x}_2)}$. Due to the selections $\zeta^{=}_{r_1,r_2,r_3,r_4}$, $\zeta^{=}_{x,x_2}$, and $\zeta^{=}_{\hat{x},\hat{x}_2}$, the latter can be expressed as $w_{\mu(x)} = w_{\mu(r_1)} \cdot w_{\mu(\hat{x})}$, and by combining the two equations while abusing the notation as explained above, we obtain $x = \hat{x}r = r\hat{x}$. The second equation is obtained analogously.

As $\hat{x}r = r\hat{x}$, there exists a word $u \in \Sigma^*$ with $r, \hat{x} \in \{u\}^*$. We choose the shortest $u$ for which $r \in \{u\}^*$. Then, due to $\hat{y}r = r\hat{y}$, we have that $\hat{y} \in \{u\}^*$ holds as well. This implies $x, y \in \{u\}^*$, $(w_{\mu(x)}, w_{\mu(y)}) \in R_{\text{com}}$, and $\mu \in \Upsilon^{R_{\text{com}}}_{x,y}(w)$. Again we can show analogously that every $\mu \in \Upsilon^{R_{\text{com}}}_{x,y}(w)$ can be extended into a $\hat{\mu} \in \hat{P}(w)$, which then proves $\mu \in P_{\text{com}}(w)$. □

In particular, this means that we can add $\zeta^{R_{\text{com}}}$ and $\zeta^{R_{\text{cyc}}}$ to core spanner representations, without leaving the class $[\![\mathsf{RGX}^{\{\pi,\zeta^{=},\cup,\bowtie\}}]\!]$.

*Example 3.8* Define $L_{\text{imp}} := \{w^n \mid w \in \Sigma^+, n \geq 2\}$ and $\rho := \zeta^{R_{\text{com}}}_{x,y}(x\{\Sigma^+\} \cdots y\{\Sigma^+\})$. Then $\mathcal{L}(\rho) = L_{\text{imp}}$.

This does not imply that $R_{\text{com}}$ can be used to select relations like $R_{\text{pow}} := \{(x, x^n) \mid n \geq 0\}$. For example, if $x := \texttt{abab}$, then $(x, y) \in R_{\text{com}}$ holds for all $y \in \{\texttt{ab}\}^*$. The authors conjecture that $R_{\text{pow}}$ is not selectable by core spanners.

Furthermore, the spanner that is constructed for $R_{\text{com}}$ in the proof of Proposition 3.7 is more complicated than the corresponding word equation $xy = yx$. In fact, we constructed both spanners not from the equations, but from a characterization of the solutions. This appears to be necessary, due the fact that spanners need to relate their variables to an input $w$, while word equations use their variables without such restrictions. We shall see in Theorem 3.13 that, if this is kept in mind, core spanners can be used to simulate word equations.

Before we consider this topic further, we examine how word equations can simulate spanners, as this shall provide useful insights on some question of static analysis in Section 4.2. One drawback of word equations is that they are unable to express many comparatively simple regular languages; like $A^*$ for any non-empty $A \subset \Sigma^*$ (cf. Karhumäki et al. [26]). In order to overcome this problem, we consider the following extension:

**Definition 3.9** Let $\eta := (\eta_L, \eta_R)$ be a word equation. A *regular constraints function*[1] is a function $\mathcal{C}$ that maps each $x \in \mathsf{Vars}(\eta)$ to a nondeterministic finite

[1] Following the terminology of [3]; literature also uses the term *rational constraints*.

automaton $\mathcal{C}(x)$. A solution $\sigma$ of $\eta$ is a *solution of $\eta$ under constraints $\mathcal{C}$* if $\sigma(x) \in \mathcal{L}(\mathcal{C}(x))$ holds for every $x \in \mathsf{Vars}(\eta)$.

Hence, regular constraints restrict the possible substitutions of a variable $x$ to a regular language $\mathcal{L}(\mathcal{C}(x))$.

A syntactic extension of word equations is EC, the *existential theory of concatenation*, which is obtained by extending word equations with $\vee$, $\wedge$, and existential quantification over variables. For example, $R_{\text{cyc}}$ is expressed by the EC-formula

$$\varphi_{\text{cyc}}(x, y) := \exists z\colon (xz = zy).$$

Using appropriate coding techniques, one can transform every EC-formula into an equivalent word equation (see Diekert [6]). Although the transformations given in [6] can result in an exponential blowup, satisfiability of word equations and of EC-formulas can still be decided in PSPACE.

Like word equations, these formulas can be further extended by adding regular constraints. For each variable $x$ and each nondeterministic finite automaton (NFA) $A$, the *(regular) constraint* $L_A(x)$ is satisfied for a solution $\sigma$ if $\sigma(x) \in \mathcal{L}(A)$. We call the resulting class of formulas $\mathsf{EC}^{\mathsf{reg}}$, the *existential theory of concatenation with regular constraints*.

*Example 3.10* Let $A$ be an NFA with $\mathcal{L}(A) = \{\mathtt{ab}^i\mathtt{a} \mid i \geq 1\}$, and define the $\mathsf{EC}^{\mathsf{reg}}$-formula $\varphi(x, y) := \exists z\colon (L_A(z) \wedge (\exists z_1, z_2\colon x = z_1 z z_2) \wedge (\exists z_1, z_2\colon y = z_1 z z_2))$.

Then $\varphi$ expresses the relation of all $(x, y)$ that have a common subword $z$ from $\mathcal{L}(A)$.

Note that we intentionally use $L_A(x)$ for constraint symbols instead of $\mathcal{C}$, to emphasize the following distinction in the use of constraints: In word equations, every variable $x$ is constrained to one language $L(\mathcal{C}(x))$. In contrast to this, an $\mathsf{EC}^{\mathsf{reg}}$-formula can use multiple constraint symbols for one variable (e. g., in the form of $L_A(x) \wedge L_{A'}(x)$), or none at all.

Using the same techniques as for EC, one can transform $\mathsf{EC}^{\mathsf{reg}}$-formulas into equivalent word equations with regular constraints. Again, the construction can result in an exponential blowup, but satisfiability of $\mathsf{EC}^{\mathsf{reg}}$-formulas can still be decided in PSPACE (cf. Diekert [6]).

In order to simulate core spanners with $\mathsf{EC}^{\mathsf{reg}}$-formulas, we introduce the following definition:

**Definition 3.11** Let $P$ be a core spanner with $\mathsf{SVars}(P) = \{x_1, \ldots, x_n\}$, $n \geq 0$, and let $\varphi(x_w, x_1^P, x_1^C, \ldots x_n^P, x_n^C)$ be an $\mathsf{EC}^{\mathsf{reg}}$-formula. We say that $\varphi$ *realizes* $P$ if, for all $w, w_1^P, w_1^C, \ldots, w_n^P, w_n^C \in \Sigma^*$, we have that $\varphi(w, w_1^P, w_1^C, \ldots, w_n^P, w_n^C) = \mathtt{True}$ holds if and only if there is a $\mu \in P(w)$ with $w_k^P = w_{[1,i_k\rangle}$ and $w_k^C = w_{[i_k,j_k\rangle}$ for each $1 \leq k \leq n$, where $[i_k, j_k\rangle = \mu(x_k)$.

This definition uses the fact that spans are always defined in relation to a word $w$. Note that every span $[i, j\rangle \in \mathsf{Spans}(w)$ is characterized by the words $w_{[1,i\rangle}$ and

$w_{[i,j\rangle}$. Hence, if $\mu \in [\![\rho]\!](w)$, the $\mathsf{EC}^{\mathsf{reg}}$-formula models $\mu(x_k) = [i_k, j_k\rangle$ by mapping $x_w$ to $w$, $x_k^P$ to $w_{[1,i_k\rangle}$, and $x_k^C$ to $w_{[i_k,j_k\rangle}$. In the naming of the variables, $C$ stands for *content*, and $P$ for *prefix*. This allows us to model spanners in $\mathsf{EC}^{\mathsf{reg}}$-formulas:

**Theorem 3.12** *There is an algorithm that, given $\rho \in \mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$, computes in polynomial time an $\mathsf{EC}^{\mathsf{reg}}$-formula $\varphi_\rho$ that realizes $[\![\rho]\!]$.*

*Proof* Before presenting the construction that is the main part of proof, we briefly consider a technical detail of functional regex formulas. On an intuitive level, functional regex formulas guarantee that in each parse tree, every variable is assigned exactly once (hence, $x\{\mathtt{a}\} \cdot x\{\mathtt{a}\}$ is not functional). Consequently, it seems reasonable to conjecture that, if a functional regex formula contains a subformula of the form $\alpha_1 \cdot \alpha_2$, then $\mathsf{SVars}(\alpha_1) \cap \mathsf{SVars}(\alpha_2) = \emptyset$ must hold.

While this conjecture is true for regex formulas that do not contain $\emptyset$, it does not hold in general. For example, consider $\alpha := \alpha_1 \cdot \alpha_2$ with $\alpha_1 := x\{\mathtt{a}\}$ and $\alpha_2 := (x\{\emptyset\} \vee \mathtt{b})$. Then $x \in \mathsf{SVars}(\alpha_1) \cap \mathsf{SVars}(\alpha_2)$, but as $x\{\emptyset\}$ can never be part of the label of a parse tree, the regex formula $\alpha$ is functional.

In order to exclude these fringe cases and simplify the construction of $\mathsf{EC}^{\mathsf{reg}}$-formulas, we introduce the following concept: A regex formula $\alpha$ is *$\emptyset$-reduced* if $\alpha = \emptyset$, or if $\alpha$ does not contain any occurrence of $\emptyset$. Using simple rewrite rules, we can observe the following. □

*Claim 1* There is an algorithm that, given a regex formula $\alpha$, computes in polynomial time an $\emptyset$-reduced regex formula $\alpha_R$ with $[\![\alpha_R]\!] = [\![\alpha]\!]$.

*Proof* In order to compute $\alpha_R$, it suffices to rewrite $\alpha$ according to the following rewrite rules:

1. $\emptyset^* \to \varepsilon$,
2. $(\hat{\alpha} \vee \emptyset) \to \hat{\alpha}$ and $(\emptyset \vee \hat{\alpha}) \to \hat{\alpha}$ for all regex formulas $\hat{\alpha}$,
3. $(\hat{\alpha} \cdot \emptyset) \to \emptyset$ and $(\emptyset \cdot \hat{\alpha}) \to \emptyset$ for all regex formulas $\hat{\alpha}$,
4. $x\{\emptyset\} \to \emptyset$ for all variables $x$.

As $\emptyset$ is never part of a parse tree, we can observe that for all regex formulas $\alpha$ and $\beta$, where $\beta$ is obtained by applying any number of these rewrite rules, $[\![\beta]\!] = [\![\alpha]\!]$ holds. Furthermore, one can use these rules to convert $\alpha$ into an equivalent and $\emptyset$-reduced $\alpha_R$ in polynomial time: If $\alpha$ is stored in a tree structure, it suffices to apply all applicable rules in bottom-up manner. □ (Claim 1)

This allows us to proceed to the main part of the proof. Recall that our goal is a procedure that, given a $\rho \in \mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$ with $\mathsf{SVars}(\rho) = \{x_1, \ldots, x_n\}$, constructs an $\mathsf{EC}^{\mathsf{reg}}$-formula $\varphi_\rho(x_w, x_1^P, x_1^C, \ldots x_n^P, x_n^C)$ such that for all $w$, $w_1^P$, $w_1^C$, …, $w_n^P$, $w_n^C \in \Sigma^*$, we have that $\varphi_\rho(w, w_1^P, w_1^C, \ldots, w_n^P, w_n^C) = \mathtt{True}$ holds if and only if there is some $\mu \in P(w)$ with $w_k^P = w_{[1,i_k\rangle}$ and $w_k^C = w_{[i_k,j_k\rangle}$ for each $1 \le k \le n$, where $[i_k, j_k\rangle = \mu(x_k)$.

In fact, this $\mu$ is always uniquely defined by $w$, the $w_k^P$, and the $w_k^C$. Based on this, we introduce some notation that simplifies our reasoning. Given $w \in \Sigma^*$ and $\mu \in P(w)$, we define the $(2n+1)$-tuple $\mathbf{w}_\mu := (w, w_1^P, w_1^C, \ldots, w_n^P, w_n^C)$ by $w_k^P := w_{[1,i_k\rangle}$ and $w_k^C := w_{[i_k,j_k\rangle}$ as in the previous paragraph. For the other direction, we say that a $(2n+1)$-tuple $\mathbf{w} = (w, w_1^P, w_1^C, \ldots, w_n^P, w_n^C)$ over $\Sigma^*$ is *spanner compatible* if, for all $1 \leq k \leq n$ the concatenated word $w_k^P \cdot w_k^C$ is a prefix of $w$. In this case, we define $\mu_\mathbf{w}$ through $\mu_\mathbf{w}(x_k) = [i_k, j_k\rangle$ with $i_k := |w_k^P| + 1$ and $j_k := |w_k^P w_k^C| + 1$ for $1 \leq k \leq n$. Note that these are one-to-one conversions if $w$ is fixed: Every $\mu$ defines its unique spanner compatible $\mathbf{w}_\mu$, and every spanner compatible $\mathbf{w}$ defines its unique $\mu_\mathbf{w}$. We can now rephrase Definition 3.11 using this terminology, and observe that $\varphi_\rho$ realizes $[\![\rho]\!]$ if and only if the following two statements hold:

1. For all $\mathbf{w} \in (\Sigma^*)^{2n+1}$, we have that $\varphi_\rho(\mathbf{w}) = \texttt{True}$ implies that $\mathbf{w}$ is spanner compatible and $\mu_\mathbf{w} \in P(w)$.
2. If $\mu \in P(w)$, then $\varphi_\rho(\mathbf{w}_\mu) = \texttt{True}$.

We now proceed to the most complicated part of this proof, the construction of $\mathsf{EC}^{\mathsf{reg}}$-formulas from regex formulas. (The following sub-proof is rather lengthy, as it contains the full induction for the correctness proof. The main part of the proof continues on page 17).

*Claim 2* There is an algorithm that, given a functional regex formula $\rho \in \mathsf{RGX}$, constructs in polynomial time an $\mathsf{EC}^{\mathsf{reg}}$-formula $\varphi_\rho$ that realizes $[\![\rho]\!]$.

*Proof* Due to Claim 1, we can assume without loss of generality that $\rho$ is $\emptyset$-reduced. We define $\varphi_\rho$ recursively as follows:

1. If $\rho$ does not contain any variables (i. e., $n = 0$), $\rho$ is a proper regular expression. Using canonical transformation techniques, we can construct in polynomial time a non-deterministic finite automaton $A$ with $\mathcal{L}(A) = \mathcal{L}(\rho)$, and we define
$$\varphi_\rho(x_w) := L_A(x_w).$$
Then $\varphi_\rho$ realizes $[\![\rho]\!]$, as $\varphi_\rho(w) = \texttt{True}$ holds if and only if $w \in \mathcal{L}(A) = \mathcal{L}(\rho)$, which holds if and only if $\mu_\mathbf{w} \in [\![\rho]\!](w)$.
2. If $\rho$ contains variables, we assume that $\mathsf{SVars}(\rho) = \{x_1, \ldots, x_n\}$ with $n \geq 1$. By definition of regex formulas, no variable of $\rho$ may occur inside of a Kleene star. Hence, we can distinguish three cases:
   (a) $\rho = \rho_1 \vee \rho_2$, where $\rho_1, \rho_2$ are functional regex formulas with $\mathsf{SVars}(\rho_1) = \mathsf{SVars}(\rho_2) = \mathsf{SVars}(\rho)$. We define
$$\varphi_\rho\left(x_w, x_1^P, x_1^C, \ldots, x_n^P, x_n^C\right) := \left(\varphi_{\rho_1}\left(x_w, x_1^P, x_1^C, \ldots, x_n^P, x_n^C\right) \vee \varphi_{\rho_2}\left(x_w, x_1^P, x_1^C, \ldots, x_n^P, x_n^C\right)\right).$$
   The intuition behind this formula should be clear; we proceed directly to proving the correctness. Assume that $\varphi_{\rho_1}$ and $\varphi_{\rho_2}$ realize $[\![\rho_1]\!]$ and $[\![\rho_1]\!]$,

respectively. We choose any $w \in \Sigma^*$. To show the direction from logic to spanners, we extend $w$ into a tuple $\mathbf{w}$. By definition, $\varphi_\rho(\mathbf{w}) = \texttt{True}$ holds if and only if $\varphi_{\rho_i}(\mathbf{w}) = \texttt{True}$ for an $i \in \{1, 2\}$. As $\varphi_{\rho_i}$ realizes $[\![\rho_i]\!]$, the tuple $\mathbf{w}$ is spanner compatible, and $\mu_{\mathbf{w}} \in [\![\rho_i]\!](w)$ holds. For the other direction, we proceed analogously: If $\mu \in [\![\rho_i]\!](w)$, then $\varphi_{\rho_i}(\mathbf{w}_\mu) = \texttt{True}$; hence, $\varphi_\rho(\mathbf{w}_\mu) = \texttt{True}$. We conclude that $\varphi_\rho$ realizes $[\![\rho]\!]$.

(b) $\rho = \rho_1 \cdot \rho_2$, where $\rho_1, \rho_2$ are functional regex formulas with $\mathsf{SVars}(\rho_1) \cup \mathsf{SVars}(\rho_2) = \mathsf{SVars}(\rho)$ and $\mathsf{SVars}(\rho_1) \cap \mathsf{SVars}(\rho_2) = \emptyset$. Without loss of generality, we can assume

$$\begin{aligned}\mathsf{SVars}(\rho_1) &= \{x_1, \ldots, x_m\},\\ \mathsf{SVars}(\rho_2) &= \{x_{m+1}, \ldots, x_n\}\end{aligned}$$

with $0 \leq m \leq n$. We define

$$\varphi_\rho\left(x_w, x_1^P, x_1^C, \ldots, x_n^P, x_n^C\right) :=$$
$$\exists y_1, y_2, z_{m+1}^P, \ldots, z_n^P : \varphi_I\left(x_w, x_1^P, x_1^C, \ldots, x_n^P, x_n^C, y_1, y_2, z_{m+1}^P, \ldots, z_n^P\right),$$

where

$$\varphi_I(x_w, x_1^P, x_1^C, \ldots, x_n^P, x_n^C, y_1, y_2, z_{m+1}^P, \ldots, z_n^P) :=$$
$$\left((x_w = y_1 \cdot y_2) \wedge \varphi_{\rho_1}\left(y_1, x_1^P, x_1^C, \ldots, x_m^P, x_m^C\right)\right.$$
$$\left.\wedge \varphi_{\rho_2}\left(y_2, z_{m+1}^P, x_{m+1}^C, \ldots, z_n^P, x_n^C\right) \wedge \bigwedge_{m+1 \leq i \leq n}\left(x_i^P = y_1 \cdot z_i^P\right)\right).$$

The idea behind this formula is as follows: As $\rho = \rho_1 \cdot \rho_2$, whenever $[\![\rho]\!](w) \neq \emptyset$ holds, $w$ can be decomposed into $w = w_1 \cdot w_2$, where $w_1$ is parsed in $\rho_1$, and $w_2$ in $\rho_2$. We store these words in the variables $y_1$ and $y_2$, respectively. For all variables in $\mathsf{SVars}(\rho_1)$, the spans of the $\mu \in [\![\rho_1]\!](w_1)$ are also spans in $w$ (as $w_1$ is a prefix of $w$). Hence, we can use the results from $\rho_1$ unchanged. On the other hand, $[\![\rho_2]\!](w_2)$ determines spans in relation to $w_2$. Hence, each span $[i, j\rangle \in \mathsf{Spans}(w_2)$ corresponds to the span $[i + c, j + c\rangle \in \mathsf{Spans}(w)$, where $c := |w_1|$. The variables $z_i^P$ represent the start of the span with respect to $y_2$, and the conjunction of the equations $(x_i^P = y_1 \cdot z_i^P)$ converts these starts into spans with respect to $x_w$.

The correctness proof is a little lengthy, but straightforward. Assume that $\varphi_{\rho_1}$ and $\varphi_{\rho_2}$ realize $[\![\rho_1]\!]$ and $[\![\rho_2]\!]$. Assume that $\varphi_\rho(\mathbf{w}) = \texttt{True}$ for some tuple $\mathbf{w} = (w, w_1^P, w_1^C, \ldots, w_n^P, w_n^C)$. By definition of $\varphi_\rho$, the tuple $\mathbf{w}$ can be extended into $\mathbf{w}' = (w, w_1^P, w_1^C, \ldots, w_n^P, w_n^C, u_1, u_2, v_{m+1}^P, \ldots, v_n^P)$ with $\varphi_I(\mathbf{w}') = \texttt{True}$. By observing the structure of $\varphi_I$, we obtain:

i. $w = u_1 \cdot u_2$,

ii. $w_i^P = u_1 \cdot v_i^P$ for $m+1 \leq i \leq n$,
iii. $\varphi_{\rho_1}(\mathbf{u}_1) = \texttt{True}$ and $\varphi_{\rho_2}(\mathbf{u}_2) = \texttt{True}$, where

$$\mathbf{u}_1 := \left(u_1, w_1^P, w_1^C, \ldots, w_m^P, w_m^C\right),$$
$$\mathbf{u}_2 := \left(u_2, v_{m+1}^P, w_{m+1}^C, \ldots, v_n^P, w_n^C\right).$$

From this and our initial assumption, we can conclude that $\mathbf{w}$ is spanner compatible, and that $\mu_{\mathbf{u}_1} \in [\![\rho_1]\!](u_1)$ and $\mu_{\mathbf{u}_2} \in [\![\rho_2]\!](u_2)$ must hold. Thus, there exits corresponding parse trees $T_1$ and $T_2$ with respective root labels $(u_1, \rho_1)$ and $(u_2, \rho_2)$. We combine these into a new parse tree $T$ by adding a new root node $(w, \rho_1 \cdot \rho_2)$ that has $T_1$ as left and $T_2$ as right child. As described in Definition 2.11, this tree $T$ defines the $w$-tuple

$$\mu^T(x_k) = \begin{cases} [\, i_k, j_k\rangle & \text{if } 1 \leq k \leq m \text{ and } \mu_1(x_k) = [\, i_k, j_k\rangle, \\ [\, i_k + |u_1|, j_k + |u_1|\rangle & \text{if } m+1 \leq k \leq n \text{ and } \mu_2(x_k) = [\, i_k, j_k\rangle. \end{cases}$$

In other words, for the variables $x_1$ to $x_m$, the $w$-tuple $\mu^T$ simulates $\mu_1$ in $u_1$, the left part of $w$; and for the variables $x_{m+1}$ to $x_n$, it simulates $\mu_2$ in $u_2$, the right part of $w$. Hence, all spans for the latter variables are shifted by $|u_1|$. Using the equalities $w_i^P = u_1 \cdot v_i^P$ from above, we obtain $\mu^T = \mu_{\mathbf{w}}$, which concludes this direction of the correctness proof. The other direction proceeds analogously: Given $\mu \in [\![\rho]\!]$, we can use the corresponding parse tree $T$ to factorize $w$ into $u_1$ and $u_2$. We then shift the spans of the variables $x_{m+1}$ to $x_n$ by $|u_1|$, and use this to obtain $\mathbf{u}_2$ with $\varphi_{\rho_2}(\mathbf{u}_2) = \texttt{True}$. No effort is necessary for $\mathbf{u}_1$, and we can then combine $\mathbf{u}_1$ and $\mathbf{u}_2$ into a tuple $\mathbf{w}$ with $\varphi_\rho(\mathbf{w}) = \texttt{True}$ and $\mathbf{w} = \mathbf{w}_\mu$. Thus, $\varphi_\rho$ realizes $[\![\rho]\!]$.

(c) $\rho = x\{\hat{\rho}\}$ for some $x \in \{x_1, \ldots, x_n\}$, and $\hat{\rho}$ is a functional regex formula with $\mathsf{SVars}(\hat{\rho}) = \mathsf{SVars}(\rho) \setminus \{x\}$. Without loss of generality, let $x = x_1$. We define

$$\varphi_\rho(x_w, x_1^P, x_1^C, \ldots, x_n^P, x_n^C) := \left(\left(x_1^P = \varepsilon\right) \wedge \left(x_1^C = x_w\right) \wedge \varphi_{\hat{\rho}}\left(x_w, x_2^P, x_2^C, \ldots, x_n^P, x_n^C\right)\right).$$

The formula uses the fact that in this case, for each $\mu \in [\![\rho]\!](w)$, we have that $\mu(x_1) = [1, |w| + 1\rangle$ must hold. This is encoded by $x_1^P = \varepsilon$ and $x_1^C = w$. For the correctness proof, assume that $\varphi_{\hat{\rho}}$ realizes $[\![\hat{\rho}]\!]$. Going from logic to spanners, assume that $\mathbf{w} = (w, w_1^P, w_1^C, \ldots, w_n^P, w_n^C)$ and $\varphi_\rho(\mathbf{w}) = \texttt{True}$. Due to the structure of the formula, we know that $w_1^P = \varepsilon$, $w_1^C = w$, and $\varphi_{\hat{\rho}}(\hat{\mathbf{w}}) = \texttt{True}$ for $\hat{\mathbf{w}} = (w, w_2^P, w_2^C, \ldots, w_n^P, w_n^C)$. As $\varphi_{\hat{\rho}}$ realizes $[\![\hat{\rho}]\!]$, we know that $\hat{\mathbf{w}}$ is spanner compatible, and $\mu_{\hat{\mathbf{w}}} \in [\![\hat{\rho}]\!](w)$. Due to this and the definition of $\rho$, we observe $\mu \in [\![\rho]\!](w)$ for the $w$-tuple

$$\mu(x_k) := \begin{cases} [\, 1, |w| + 1\rangle & \text{if } k = 1, \\ \mu_{\hat{\mathbf{w}}}(x_k) & \text{if } k > 1. \end{cases}$$

As $\mu = \mu_{\mathbf{w}}$, we conclude this direction of the proof. For the other direction, let $\mu \in [\![\rho]\!](w)$. By definition, $\mu(x_1) = [1, |w| + 1\rangle$ and $\hat{\mu} \in [\![\hat{\rho}]\!]$ for $\hat{\mu} = \mu|_{\{x_2, \ldots, x_n\}}$. Due to our initial assumption, $\varphi_{\hat{\rho}}(\mathbf{w}_{\hat{\mu}}) = \texttt{True}$

must hold. Note that $\mathbf{w}_{\hat{\mu}} = (w, w_2^P, w_2^C, \ldots, w_n^P, w_n^C)$, and let $\mathbf{w} := (w, \varepsilon, w, w_2^P, w_2^C, \ldots, w_n^P, w_n^C)$. Then $\varphi_\rho(\mathbf{w}) = \texttt{True}$; and as $\mathbf{w} = \mathbf{w}_\mu$, this concludes this direction. Thus, $\varphi_\rho$ realizes $[\![\rho]\!]$.

Finally, note that the size of $\varphi_\rho$ is polynomial in the size of $\rho$. More importantly, the construction of $\varphi_\rho$ follows the syntax of $\rho$, and does not requires expensive additional computations. Hence, $\varphi_\rho$ can be computed in polynomial time. □ (Claim 2)

Using Claim 2, we have the conversion for RGX, the class of (functional) regex formulas. As final step of the proof, we extended this to all core spanner representations (i. e., the full class $\mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$). Consider an arbitrary core spanner representations $\rho \in \mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$ with $\mathsf{SVars}(\rho) = \{x_1, \ldots, x_n\}$, $n \geq 0$. We distinguish the following cases:

1. $\rho$ is a regex formula. This case is covered in Claim 2.
2. $\rho = \pi_Y \hat{\rho}$, with $Y = \mathsf{SVars}(\rho)$ and $\mathsf{SVars}(\hat{\rho}) \supseteq \mathsf{SVars}(\rho)$. Assume without loss of generality that $\mathsf{SVars}(\hat{\rho}) = \{x_1, \ldots, x_{n+m}\}$ with $m \geq 0$. We define

$$\varphi_\rho\left(x_w, x_1^P, x_1^C, \ldots, x_n^P, x_n^C\right) := \exists x_{n+1}^P, x_{n+1}^C, \ldots, x_{n+m}^P, x_{n+m}^C \colon \varphi_{\hat{\rho}}\left(x_w, x_1^P, x_1^C, \ldots, x_{n+m}^P, x_{n+m}^C\right)$$

   Regarding the correctness, assume that $\varphi_{\hat{\rho}}$ realizes $[\![\hat{\rho}]\!]$. Hence, if $\hat{\mu} \in [\![\hat{\rho}]\!](w)$, we have $\varphi_{\hat{\rho}}(\mathbf{w}_{\hat{\mu}}) = \texttt{True}$. This means that for $\mu := \hat{\mu}|_Y$, we know that $\varphi_\rho(\mathbf{w}_\mu) = \texttt{True}$ holds as well. Likewise, if $\varphi_\rho(\mathbf{w}) = \texttt{True}$, there exists an extension $\hat{\mathbf{w}}$ of $\mathbf{w}$ with $\mu_{\hat{\mathbf{w}}} \in [\![\hat{\rho}]\!](w)$. As $\hat{\mathbf{w}}$ is spanner compatible, so is $\mathbf{w}$. Thus, we observe $\mu_{\mathbf{w}} = \mu_{\hat{\mathbf{w}}}|_Y$ and $\mu_{\mathbf{w}} \in [\![\rho]\!](w)$. Hence, $\varphi_\rho$ realizes $[\![\rho]\!]$.
3. $\rho = \zeta^=_{\mathbf{x}} \hat{\rho}$, with $\mathbf{x} \in (\mathsf{SVars}(\hat{\rho}))^m$, $2 \leq m \leq n$, and $\mathsf{SVars}(\rho) = \mathsf{SVars}(\hat{\rho})$. Assume without loss of generality that $\mathbf{x} = (x_1, \ldots, x_m)$. We define

$$\varphi_\rho\left(x_w, x_1^P, x_1^C, \ldots, x_n^P, x_n^C\right) := \left(\varphi_{\hat{\rho}}\left(x_w, x_1^P, x_1^C, \ldots, x_n^P, x_n^C\right) \wedge \bigwedge_{2 \leq i \leq m} \left(x_1^C = x_i^C\right)\right).$$

   Recall that $\zeta^=_{x_i,x_j}$ only checks whether $w_{\mu(x_i)} = w_{\mu(x_j)}$ holds, not whether $\mu(x_i) = \mu(x_j)$. This is equivalent to checking whether $x_i^C = x_j^C$ holds.

   We only proof the correctness for $m = 2$, the other cases proceed analogously (or by reducing them to this binary case). Assume that $\varphi_{\hat{\rho}}$ realizes $[\![\hat{\rho}]\!]$. Let $\mu \in [\![\rho]\!](w)$. Then $w_{\mu(x_1)} = w_{\mu(x_2)}$ and $\mu \in [\![\hat{\rho}]\!](w)$ hold by definition. The latter implies $\varphi_{\hat{\rho}}(\mathbf{w}) = \texttt{True}$. Together with the former and the structure of $\varphi_\rho$, we conclude $\varphi_\rho(\mathbf{w}) = \texttt{True}$.

   For the other direction, let $\varphi_\rho(\mathbf{w}) = \texttt{True}$. By the structure of $\varphi_\rho$, we know that $\varphi_{\hat{\rho}}(\mathbf{w}) = \texttt{True}$ and $w_1^C = w_2^C$. As $\varphi_{\hat{\rho}}$ realizes $[\![\hat{\rho}]\!]$, we have that $\mathbf{w}$ is spanner compatible, and $\mu_{\mathbf{w}} \in [\![\hat{\rho}]\!](w)$. Due to $w_1^C = w_2^C$, this implies $\mu_{\mathbf{w}} \in [\![\rho]\!](w)$ and concludes the proof that $\varphi_\rho$ realizes $[\![\rho]\!]$.

4. $\rho = (\rho_1 \cup \rho_2)$, with $\mathsf{SVars}(\rho_1) = \mathsf{SVars}(\rho_2) = \mathsf{SVars}(\rho)$. Let

$$\varphi_\rho\left(x_w, x_1^P, x_1^C, \ldots, x_n^P, x_n^C\right) := \left(\varphi_{\rho_1}\left(x_w, x_1^P, x_1^C, \ldots, x_n^P, x_n^C\right) \vee \varphi_{\rho_2}\left(x_w, x_1^P, x_1^C, \ldots, x_n^P, x_n^C\right)\right).$$

In this case, the construction and the correctness proof are identical to case 2a (disjunction) in the proof of Claim 2.

5. $\rho = (\rho_1 \bowtie \rho_2)$ with $\mathsf{SVars}(\rho) = \mathsf{SVars}(\rho_1) \cup \mathsf{SVars}(\rho_2)$. We assume without loss of generality that $\mathsf{SVars}(\rho_1) = \{x_1, \ldots, x_l\}$ and $\mathsf{SVars}(\rho_2) = \{x_m, \ldots, x_n\}$ with $0 \leq l \leq n$, $1 \leq m \leq n+1$, and $m \leq l+1$. Note that this implies $\mathsf{SVars}(\rho_1) \cap \mathsf{SVars}(\rho_2) = \{x_m, \ldots, x_l\}$, and $\mathsf{SVars}(\rho_1) \cap \mathsf{SVars}(\rho_2) = \emptyset$ if and only if $m = l+1$. We define

$$\varphi_\rho\left(x_w, x_1^P, x_1^C, \ldots, x_n^P, x_n^C\right) := \left(\varphi_{\rho_1}\left(x_w, x_1^P, x_1^C, \ldots, x_l^P, x_l^C\right) \wedge \varphi_{\rho_2}\left(x_w, x_m^P, x_m^C, \ldots, x_n^P, x_n^C\right)\right).$$

The definition of $\bowtie$ requires that $\mu \in [\![\rho]\!](w)$ holds if and only if there are $\mu_1 \in [\![\rho_1]\!](w)$ and $\mu_2 \in [\![\rho_2]\!](w)$ with $\mu_1(x_i) = \mu_2(x_i)$ for all $i \in \{m, \ldots, l\}$. For each of these variables $x_i$, we have that $\varphi_{\rho_1}$ and $\varphi_{\rho_2}$ model the span with the same variables $x_i^P$ and $x_i^C$.

To prove the correctness, assume that $\varphi_{\rho_1}$ and $\varphi_{\rho_2}$ realize $[\![\rho_1]\!]$ and $[\![\rho_2]\!]$, respectively. Let $\mu \in [\![\rho]\!](w)$. Then there exist $\mu_1 \in [\![\rho_1]\!](w)$ and $\mu_2 \in [\![\rho_2]\!](w)$ with $\mu_1 = \mu|_{\{x_1,\ldots,x_l\}}$ and $\mu_2 = \mu|_{\{x_m,\ldots,x_n\}}$, which implies $\mu_1(x_k) = \mu_2(x_k)$ for $m \leq k \leq l$. Now, in order to talk about the components of $\mathbf{w}_{\mu_1}$ and $\mathbf{w}_{\mu_2}$, we name the components of the tuples as $\mathbf{w}_{\mu_1} = (w, w_1^P, w_1^C, \ldots, w_l^P, w_l^C)$ and $\mathbf{w}_{\mu_2} = (w, w_m^P, w_m^C, \ldots, w_n^P, w_n^C)$. As $\mu_1$ and $\mu_2$ agree on their common variables, we can combine this to $\mathbf{w} := (w, w_1^P, w_1^C, \ldots, w_n^P, w_n^C) = \mathbf{w}_\mu$. As each $\varphi_{\rho_i}$ realizes $[\![\rho_i]\!]$, we know that $\varphi_{\rho_i}(\mathbf{w}_{\mu_i}) = \texttt{True}$. Hence, $\varphi_\rho(\mathbf{w}_\mu) = \varphi_\rho(\mathbf{w}) = \texttt{True}$. This concludes this direction.

For the other direction, assume that $\varphi_\rho(\mathbf{w}) = \texttt{True}$. Due to the structure of the formula, this implies $\varphi_{\rho_i}(\mathbf{w}_i) = \texttt{True}$, where $\mathbf{w}_1 := (w, w_1^P, w_1^C, \ldots, w_l^P, w_l^C)$ and $\mathbf{w}_2 := (w, w_m^P, w_m^C, \ldots, w_n^P, w_n^C)$. As $\varphi_{\rho_i}$ realizes $[\![\rho_i]\!]$, we know that $\mathbf{w}_i$ is spanner compatible, and $\mu_{\mathbf{w}_i} \in [\![\rho_i]\!](w)$. Due to the former, $\mathbf{w}$ is also spanner compatible. Due to the latter, we know that $\mu_{\mathbf{w}} \in [\![\rho]\!](w)$, as $\mu_{\mathbf{w}}(x_k) = \mu_{\mathbf{w}_1}(x_k) = \mu_{\mathbf{w}_2}(x_k)$ for all $m \leq k \leq l$. Hence, $\varphi_\rho$ realizes $[\![\rho]\!]$.

The formula $\varphi_\rho$ can be derived from $\rho$ without requiring further computation, and its size is polynomial in the size of $\rho$. Hence, $\varphi_\rho$ can be constructed in polynomial time.

As we shall see in Section 4.2, this result allows us to find upper bounds on two problems from the static analysis of spanners. We now examine how spanners can simulate word equations (and, thereby, also $\mathsf{EC}^{\mathsf{reg}}$-formulas). As discussed above, spanners need to relate their variables to an input word. Hence, we only state the following result, which is a weaker form of simulation than for the other direction:

**Theorem 3.13** *Every word equation $\eta := (\eta_L, \eta_R)$ with regular constraints $\mathcal{C}$ can be converted effectively into a $\rho \in \mathsf{RGX}^{\{\zeta^{=}, \times\}}$ with $\mathsf{SVars}(\rho) \supseteq \mathsf{Vars}(\eta)$ such that for all $w \in \Sigma^*$, there is a solution $\sigma$ of $\eta$ under constraints $\mathcal{C}$ with $w = \sigma(\eta_L) = \sigma(\eta_R)$ if and only if there is a $\mu \in [\![\rho]\!](w)$ with $\sigma(x) = w_{\mu(x)}$ for all $x \in \mathsf{Vars}(\eta)$.*

*Proof* As each of the two sides of a word equation is a pattern, we can transform those into regex formulas by using the a slightly adapted version of the conversion procedure from the proof of Theorem 3.3. Only two changes are made. Firstly, instead of binding a variable $x$ to some $\Sigma^*$, we respect the constraints by using a regular expression for the language $\mathcal{L}(\mathcal{C}(x))$. Secondly, in order to ensure $\mathsf{SVars}(\rho) \supseteq \mathsf{Vars}(\eta)$, the first occurrence of a variable $x$ is not represented by $x_1$, but by $x$.

Assume that $\eta_L = \alpha_1 \cdots \alpha_m$ and $\eta_R = \alpha_{m+1} \cdots \alpha_n$ with $m, n \in \mathbb{N}$, $m+1 \leq n$, and $\alpha_1, \ldots, \alpha_n \in (\Sigma \cup X)$. We construct regex formulas $\hat{\eta}_L := \hat{\alpha}_1 \cdots \hat{\alpha}_m$ and $\hat{\eta}_R := \hat{\alpha}_{m+1} \cdots \hat{\alpha}_n$, where for each position in $1 \leq i \leq n$, we define $\hat{\alpha}_i$ as follows:

1. If $\alpha_i$ is a terminal (i. e., there is an $a \in \Sigma$ with $\alpha_i = a$), let $\hat{\alpha}_i := a$.
2. If $\alpha_i$ is a variable (i. e., there is an $x \in X$ with $\alpha_i = x$), let $\gamma$ be a regular expression with $\mathcal{L}(\gamma) = \mathcal{L}(\mathcal{C}(x))$. Furthermore, let $j := |\alpha_1 \cdots \alpha_i|_x$.
   (a) If $j = 1$, define $\hat{\alpha}_i := x\{\gamma\}$
   (b) If $j \geq 2$, define $\hat{\alpha}_i := x_j\{\gamma\}$ (where $x_j \in \mathsf{SVars}$ is a new variable).

This ensures that $\mathsf{SVars}(\hat{\eta}_L)$ and $\mathsf{SVars}(\hat{\eta}_R)$ are disjoint. We then construct a sequence $S$ of string equality selections appropriately: For every $x \in \mathsf{Vars}(\eta)$ with $k := |\eta_L \eta_R|_x \geq 2$, the sequence $S$ includes a selection $\zeta^{=}_{x, x_2, \ldots, x_k}$.

Finally, we define $\rho := S(\hat{\eta}_L \times \hat{\eta}_R)$.

In order to prove that this construction is correct, we show that for all $w \in \Sigma^*$, $\mu \in [\![\rho]\!](w)$ holds if and only if there is a solution $\sigma$ of $\eta$ under constraints $\mathcal{C}$ with

1. $w = \sigma(\eta_L) = \sigma(\eta_R)$, and
2. $\sigma(x) = w_{\mu(x)}$ for all $x \in \mathsf{Vars}(\eta)$.

We begin with the *if*-direction. Assume that $\sigma$ is a solution of $\eta$ under constraints $\mathcal{C}$. Let $w := \sigma(\eta_L)$ (which implies $w = \sigma(\eta_R)$, as $\sigma$ is a solution of $\eta$). We use this to define a $w$-tuple $\mu$ as follows: Due to our construction, each variable $\hat{x} \in \mathsf{SVars}(\rho)$ corresponds to a uniquely defined $\alpha_i$ with $\alpha_i = x$. If $1 \leq i \leq m$, then $\hat{x}$ occurs in $\hat{\eta_L}$, and if $m+1 \leq i \leq n$, then $\hat{x}$ occurs in $\hat{\eta_R}$. We now define $\mu(\hat{x}) := [l, r\rangle$, where the choice of $l$ and $r$ depends on this distinction:

- If $\hat{x}$ occurs in $\hat{\eta_L}$, let $l := |\sigma(\alpha_1 \cdots \alpha_{i-1})| + 1$ and $r := |\sigma(\alpha_1 \cdots \alpha_i)| + 1$,
- If $\hat{x}$ occurs in $\hat{\eta_R}$, let $l := |\sigma(\alpha_{m+1} \cdots \alpha_{i-1})| + 1$ and $r := |\sigma(\alpha_{m+1} \cdots \alpha_i)| + 1$.

Either way, we know that $w_{\mu(\hat{x})} = \sigma(x)$ holds, which implies $w_{\mu(\hat{x})} \in \mathcal{L}(\mathcal{C}(x))$. Analogously, we can use $\sigma$ to construct parse trees for $(w, \hat{\eta}_L)$ and $(w, \hat{\eta}_R)$. This allows us to conclude $\mu \in [\![\hat{\eta}_L \times \hat{\eta}_R]\!](w)$. Furthermore, for every selection $\zeta^{=}_{x, x_2, \ldots, x_k}$ in $S$, we know from the construction that $x$ and all $x_i$ ($1 \leq i \leq k$) refer to the same $x \in \mathsf{Vars}(\eta)$, which means that $w_{\mu(x)} = w_{\mu(x_i)} = \sigma(x)$ holds. Hence, for each of these selections, $\mu \in [\![\hat{\eta}_L \times \hat{\eta}_R]\!](w)$ implies $\mu \in [\![\zeta^{=}_{x, x_2, \ldots, x_k}(\hat{\eta}_L \times \hat{\eta}_R)]\!](w)$.

Thus, $\mu \in [\![S(\hat{\eta}_L \times \hat{\eta}_R)]\!](w)$, which is equivalent to $\mu \in [\![\rho]\!](w)$ and concludes this direction of the proof.

For the *only if*-direction, assume that $\mu \in [\![\rho]\!](w)$. We now define a pattern substitution $\sigma$ by $\sigma(a) := a$ for all $a \in \Sigma$, and $\sigma(x) := w_{\mu(x)}$ for all $x \in \mathsf{Vars}(\eta)$. By our construction, $\mu(x)$ is derived from $x\{\gamma\}$, where $\mathcal{L}(\gamma) = \mathcal{L}(\mathcal{C}(x))$ must hold, which means that $w_{\mu(x)} \in \mathcal{L}(\mathcal{C}(x))$, and hence $\sigma(x) \in \mathcal{L}(\mathcal{C}(x))$. All that remains to be shown is that $\sigma(\eta_L) = \sigma(\eta_R) = w$. In order to prove this, we first define $\hat{w}_L = \hat{w}_1 \cdots \hat{w}_m$ and $\hat{w}_R = \hat{w}_{m+1} \cdots \hat{w}_n$, where the $\hat{w}_i$ with $1 \leq i \leq n$ are defined as follows:

1. If $\alpha_i = a \in \Sigma$, let $\hat{w}_i := a$. Then $\hat{w}_i = \hat{\alpha}_i$ and $\hat{w} = \sigma(\alpha_i)$ hold by definition.
2. If $\alpha_i = x \in X$, let $j := |\alpha_1 \cdots \alpha_i|_x$. We distinguish two cases.
   (a) If $j = 1$, let $\hat{w}_i = w_{\mu(x)}$. Then $\sigma(\alpha_i) = \hat{w}_i$ holds by definition.
   (b) If $j \geq 2$, let $\hat{w}_i = w_{\mu(x_j)}$. Observe that $S$ contains the selection $\zeta^{=}_{x,x_2,\ldots,x_k}$. Hence, $w_{\mu(x_j)} = w_{\mu(x)}$ holds, which implies $\sigma(\alpha_i) = \hat{w}_i$.

Now note that the $\hat{w}_i$ correspond to the labels of the parse trees that have root labels $(w, \hat{\eta}_L)$ and $(w, \hat{\eta}_R)$. Hence, $\hat{w}_L = w$ and $\hat{w}_R = w$ must hold. Furthermore, we have $\hat{w}_i = \sigma(\alpha_i)$ for all $1 \leq i \leq m$. This allows us to conclude

$$\begin{aligned} \sigma(\eta_L) &= \sigma(\alpha_1 \cdots \alpha_m) & \sigma(\eta_R) &= \sigma(\alpha_{m+1} \cdots \alpha_n) \\ &= \hat{w}_1 \cdots \hat{w}_m = \hat{w}_L, & &= \hat{w}_{m+1} \cdots \hat{w}_n = \hat{w}_R. \end{aligned}$$

We observe $\sigma(\eta_L) = \sigma(\eta_R) = w$, which concludes this direction of the proof. □

While this form of simulation is weaker (as $w$ has to be present), it still shows that the constructed spanner is satisfiable if and only if the word equation (with constraints) is satisfiable. Furthermore, the computed $(V, w)$-relations encode solutions of the equation.

*Example 3.10* Let $\mathtt{a}, \mathtt{b} \in \Sigma$ and define $\eta := (xy, yx)$ with $\mathcal{L}(\mathcal{C}(x)) = \mathcal{L}(\mathtt{aab}^+)$ and $\mathcal{L}(\mathcal{C}(y)) = \Sigma^+$. The construction from the proof of Theorem 3.13 results in

$$\rho := \zeta^{=}_{x,x_2}\zeta^{=}_{y,y_2}(\hat{\eta}_L \times \hat{\eta}_R),$$

where $\hat{\eta}_L := x\{\mathtt{aab}^+\} \cdot y\{\Sigma^+\}$ and $\hat{\eta}_R := y_2\{\Sigma^+\} \cdot x_2\{\mathtt{aab}^+\}$.

The only reason that this construction is not necessarily possible in polynomial time is that regular constraints are specified with NFAs, while core spanners use regular expressions, which can lead to an exponential increase in the size.

There is a similar construction that does not use the join operator: By adding new variables $z_1, z_2$, we can construct

$$\hat{\rho} := \zeta^{=}_{x,x_2}\zeta^{=}_{y,y_2}\zeta^{=}_{z_1,z_2}(z_1\{\hat{\eta}_L\}z_2\{\hat{\eta}_R\}),$$

which behaves almost like $\rho$; the only difference that the solution is encoded in $w = \sigma(\eta_L \cdot \eta_R)$, instead of $\sigma(\eta_L)$.

### 3.3 Xregexes

As shown by Fagin et al. [7], there are languages that are recognized by xregexes, but not by core spanners. In order to prove this, [7] introduced the so-called "uniform-0-chunk"-language $L_{\text{uzc}}$: Assuming $0, 1 \in \Sigma$, $L_{\text{uzc}}$ is defined as the language of all $w = s_1 \cdot t \cdot s_2 \cdot t \cdots s_{n-1} \cdot t \cdot s_n$, where $n > 0$, $s_1, \ldots, s_n \in \{1\}^+$, and $t \in \{0\}^+$. Then $\mathcal{L}(\alpha_{\text{uzc}}) = L_{\text{uzc}}$ holds for the xregex $\alpha_{\text{uzc}} := 1^+ \cdot x\{0^*\} \cdot (1^+ \cdot \&x)^* \cdot 1^+$, but no core spanner recognizes $L_{\text{uzc}}$.

Considering that the syntax of regex formulas does not allow the use of variables inside a Kleene star (or plus), this inexpressibility result might be considered expected, as $\alpha_{\text{uzc}}$ has an occurrence of $\&x$ inside a Kleene star. This raises the question whether xregexes that restrict variables in a similar manner can still recognize languages that core spanners cannot. In order to examine this question, we define the following subclass of xregexes:

**Definition 3.15** An xregex $\alpha$ is *variable star-free* (short: *vstar-free*) if, for every $\beta \in \mathsf{Sub}(\alpha)$ with $\beta = \gamma^*$, no subexpression of $\gamma$ is a variable binding or a variable reference. We denote the class of all vstar-free xregexes by $\mathsf{vsfXR}$.

As we shall see in Theorem 3.21 below, every language that is recognized by a vstar-free xregex is also recognized by a core spanner. While this observation might be considered not very surprising, its proof needs to deal with some technicalities. In particular, one needs to deal with expressions like $\alpha := x\{\Sigma^*\} \cdot (\&x \vee \&x\&x)$. A conversion in the spirit of Theorem 3.3 would need to replace the $\&x$ with distinct variables and ensure equality with selections; but as the disjunction contains subexpressions with distinct numbers of occurrences of $\&x$, we would not be able to ensure functionality of the resulting regex formula. We avoid these problems by working with the following syntactically restricted class of vstar-free xregexes:

**Definition 3.16** An $\alpha \in \mathsf{vsfXR}$ is an *xregex path* if, for every $\beta \in \mathsf{Sub}(\alpha)$ with $\beta = (\gamma_1 \vee \gamma_2)$, no subexpression of $\gamma_1$ or $\gamma_2$ is a variable binding or a variable reference. We denote the class of all xregex paths by $\mathsf{XRP}$.

Intuitively, an xregex path $\alpha \in \mathsf{XRP}$ can be understood as a concatenation $\alpha = \alpha_1 \cdots \alpha_n$, where each $\alpha_i$ is either a proper regular expression, a variable reference, or a variable binding of the form $\alpha_i = x\{\hat{\alpha}\}$, where $\hat{\alpha}$ is also an xregex path. By "multiplying out" disjunctions that contain variables, we can convert every vstar-free xregex into a disjunction of xregex paths.

**Lemma 3.17** *There is an algorithm that, given $\alpha \in \mathsf{vsfXR}$, computes $\alpha_1, \ldots, \alpha_n \in \mathsf{XRP}$ with $\mathcal{L}(\alpha) = \bigcup_{i=1}^{n} \mathcal{L}(\alpha_i)$.*

*Proof* If a vstar-free xregex $\alpha$ is not an xregex path, there exists at least one $x \in \mathsf{Vars}(\alpha)$ and at least one subexpression $\beta \in \mathsf{Sub}(\alpha)$ with $\beta \neq \alpha$ such that

1. $\beta$ is a disjunction; i. e., $\beta = (\gamma_1 \vee \gamma_2)$ for some $\gamma_1, \gamma_2 \in \mathsf{vsfXR}$,

2. $\beta$ contains a variable binding $x\{\cdots\}$ or a variable reference $\&x$.

We now rewrite $\alpha$ into two vstar-free xregexes $\alpha_1$ and $\alpha_2$, by replacing $\beta$ with $\gamma_1$ or $\gamma_2$, respectively. We observe that this rewriting step does not change the language: □

*Claim 1* $\mathcal{L}(\alpha) = \mathcal{L}(\alpha_1) \cup \mathcal{L}(\alpha_2)$

*Proof* If $w \in \mathcal{L}(\alpha)$, there exists an $\alpha$-parse tree $T$ for $w$; in other words, the root of $T$ is labelled with $(w, \alpha)$. Recall that $\alpha$ is vstar-free. Hence, we know that $T$ uses the occurrence of $\beta$ that was rewritten to create $\alpha_1$ and $\alpha_2$ at most once (in order to be able to use the occurrence multiple times, $\alpha$ would need to contain a star around $\beta$).

This allows us to distinguish two possibilities: If $T$ does not use this occurrence of $\beta$ at all, we can immediately transform $T$ into an $\alpha_i$-parse tree $T_i$ ($i \in \{1, 2\}$) by replacing the root label with $(w, \alpha_i)$, and changing all children accordingly. Hence, $w \in \mathcal{L}(\alpha_i)$ holds.

On the other hand, if $T$ uses this occurrence of $\beta$, then there exists a uniquely defined node $v$ in $T$ that is labeled with $(\hat{w}, \beta)$ for some word $\hat{w} \in \Sigma^*$. Furthermore, this node corresponds to the occurrence of $\beta$ that was rewritten in $\alpha_1$ and $\alpha_2$. By definition, $v$ has exactly one child $\hat{v}$ that is labeled with either $(\hat{w}, \gamma_i)$, where $i \in \{1, 2\}$. We rewrite $T$ into a $\alpha_i$-parse tree $T_i$ by removing $v$ (i. e., $\hat{v}$ replaces $v$), relabeling the root of $T$ to $(w, \alpha_i)$, and changing all labels between the root and $\hat{v}$ accordingly. As $T_i$ is a $\alpha_i$-parse tree for $w$, we have that $w \in \mathcal{L}(\alpha_i)$ holds. This proves $\mathcal{L}(\alpha) \subseteq \mathcal{L}(\alpha_1) \cup \mathcal{L}(\alpha_2)$.

In order to prove $\mathcal{L}(\alpha) \supseteq \mathcal{L}(\alpha_1) \cup \mathcal{L}(\alpha_2)$, we proceed analogously: If $w \in \mathcal{L}(\alpha_1) \cup \mathcal{L}(\alpha_2)$, we can transform a $\alpha_i$-parse tree for $w$ into an $\alpha$-parse tree by inserting a node $(\hat{w}, \beta)$ (if necessary), and changing the labels accordingly. □ (Claim 1)

Note that this equivalence relies on the fact that $\alpha$ is vstar-free, which implies that $\beta$ does not occur inside a Kleene star. For xregexes that are not vstar-free, we can only conclude $\mathcal{L}(\alpha) \supseteq \mathcal{L}(\alpha_1) \cup \mathcal{L}(\alpha_2)$. This is easily seen considering the example of $x\{\mathtt{a}\}y\{\mathtt{b}\}(\&x \vee \&y)^*$, which would be rewritten to $x\{\mathtt{a}\}(\&x)^*$ and $y\{\mathtt{b}\}(\&y)^*$.

We repeat this rewriting procedure on every created vstar-free xregex that is not an xregex path. This procedure terminates, as every rewriting removes a disjunction that contains at least one variable (binding or reference). Hence if $\alpha$ contains $k \in \mathbb{N}_{>0}$ disjunctions, this process results in xregex paths $\alpha_1, \ldots, \alpha_n$ for some $n \leq 2^k$, and $\mathcal{L}(\alpha) = \bigcup_{i=1}^{n} \mathcal{L}(\alpha_i)$.

*Example 3.18* Let $\alpha := x\{\Sigma^*\} \cdot \&x \cdot (x\{\Sigma^*\} \vee y\{\Sigma^*\}) \cdot (\&x \vee \&y) \cdot \&x$. Multiplying out the disjunctions, we obtain the following xregex paths:

$$\begin{aligned}
\alpha_1 &= x\{\Sigma^*\} \cdot \&x \cdot x\{\Sigma^*\} \cdot \&x \cdot \&x,\\
\alpha_2 &= x\{\Sigma^*\} \cdot \&x \cdot x\{\Sigma^*\} \cdot \&y \cdot \&x,\\
\alpha_3 &= x\{\Sigma^*\} \cdot \&x \cdot y\{\Sigma^*\} \cdot \&x \cdot \&x,\\
\alpha_4 &= x\{\Sigma^*\} \cdot \&x \cdot y\{\Sigma^*\} \cdot \&y \cdot \&x.
\end{aligned}$$

Then $\mathcal{L}(\alpha) = \bigcup_{i=1}^{4} \mathcal{L}(\alpha_i)$.

This transformation process might result in an exponential number of xregex paths; but as efficiency is not of concern right now, this is not a problem (the followup paper Freydenberger [13] shows that this blowup can be avoided with a more involved construction). Each of these xregex paths is then transformed into a functional regex formula:

**Lemma 3.19** *There is an algorithm that, given* $\alpha \in \mathsf{XRP}$*, computes* $\rho \in \mathsf{RGX}^{\{\pi,\zeta^{=}\}}$ *with* $\mathcal{L}(\rho) = \mathcal{L}(\alpha)$.

*Proof* Before we start with the proof, note that we can safely assume that $\alpha$ does not contain $\emptyset$: If $\emptyset$ occurs inside a Kleene star (or a disjunction), that Kleene star (or disjunction) cannot contain any variable bindings or references, as $\alpha$ is an xregex path. Hence, we can remove $\emptyset$ as in the proof of Theorem 3.12. All other occurrences of $\emptyset$ imply $\mathcal{L}(\alpha) = \emptyset$ – in this case, we are done.

Our goal is to rewrite the xregex path $\alpha$ into an equivalent core spanner of the form $\pi_{\emptyset} S \delta$, where $\delta$ is a regex formula, and $S$ is a sequence of string equality selections.

The main idea of the construction is quite straightforward: We basically replace each variable reference $\&x$ with a unique $x_i\{\Sigma^*\}$, and use a string equality $\zeta^{=}_{x,x_i}$ to connect $x_i$ with the appropriate binding. The only technical problem is that unlike regex formulas, xregexes allow variables to be bound multiple times. We solve this by using a unique variable for every occurrence of a variable binding in $\alpha$.

As explained above, the xregex path $\alpha$ can be understood as a concatenation $\alpha = \alpha_1 \cdots \alpha_n$, where each $\alpha_i$ is either a proper regular expression, a variable reference, or a variable binding of the form $\alpha_i = x\{\hat{\alpha}\}$, where $\hat{\alpha}$ is also an xregex path.

Now, if we choose any occurrence of a variable reference $\&x$ in $\alpha$, exactly one of the following two cases applies:

1. There is no binding $x\{\}$ in $\alpha$ that to the left of that occurrence of $\&x$, or
2. there is a binding $x\{\}$ in $\alpha$ that is to the left of that occurrence of $\&x$.

In the first case, this $\&x$ will always default to $\varepsilon$, which means that we can safely replace it with $\varepsilon$.

In the second case, we see that this $\&x$ will always refer to the variable binding $x\{\}$ that is closest to it to the left in $\alpha$. In other words, we can simply read $\alpha$ from left to right. All $\&x$ before the first binding for $x$ default to $\varepsilon$; and all $\&x$ after the first binding for $x$ refer to the most recent binding for $x$ (recall that, according to our definition of xregexes, no variable binding for a variable $x$ may contain another binding of $x$).

This allows us to rewrite $\alpha$ into an xregex path $\gamma$ with $\mathcal{L}(\gamma) = \mathcal{L}(\alpha)$ such that no occurrence of a variable reference $\&x$ in $\gamma$ refers to the default value $\varepsilon$, and every variable binding $x\{\cdots\}$ occurs at most once. This is done the following way: We read $\alpha$ from left to right. If we encounter a reference $\&x$ for which no binding has been seen, we replace it with $\varepsilon$. If we encounter a binding $x\{\}$ that has already been seen before, we replace it with a binding for a new variable $\hat{x}$, and all occurrences of $\&x$

are renamed to $\&\hat{x}$. (Of course, further occurrences of $x\{\}$ would require further new variables.) For example, the xregex path

$$\alpha_2 = x\{\Sigma^*\} \cdot \&x \cdot x\{\Sigma^*\} \cdot \&y \cdot \&x$$

from Example 3.18 would be rewritten to

$$\gamma_2 = x\{\Sigma^*\} \cdot \&x \cdot \hat{x}\{\Sigma^*\} \cdot \varepsilon \cdot \&\hat{x}.$$

After rewriting $\alpha$ to $\gamma$, the next step is to transform $\gamma$ into a regex formula $\delta$ by replacing all variable references in a manner that is similar to the proof of Theorem 3.3. More specifically, we construct $\delta$ by replacing, for each $x \in \mathsf{Vars}(\gamma)$, the $i$-th occurrence of $\&x$ in $\gamma$ with $x_i\{\Sigma^*\}$. Note that $\delta$ is functional: Each variable in $\mathsf{SVars}(\delta)$ appears exactly once in $\delta$; and as $\delta$ is also an xregex path, this implies that every $\delta$-parse tree contains every variable exactly once. (Recall that we assumed that $\alpha$ does not contain $\emptyset$; hence, neither do $\gamma$ and $\delta$.)

For every variable $x$ for which there occur references $\&x$ in $\gamma$, we define a selection $\zeta^{=}_{V_x}$, where $V_x := \{x\} \cup \{x_i \mid x_i \text{ occurs in } \delta\}$. We let $S$ denote a sequence of these selections (the order is irrelevant), and define the spanner representation $\rho := \pi_\emptyset S \delta$. As we simulate the behavior of each variable binding $x\{\cdots\}$ and its references $\&x$ using the selection $\zeta^{=}_{V_x}$, it is easy to see that $\mathcal{L}(\rho) = \mathcal{L}(\gamma)$ and, hence, $\mathcal{L}(\rho) = \mathcal{L}(\alpha)$. □

*Example 3.20* Consider the xregex path

$$\alpha := \&x \cdot x\{\Sigma^* \cdot y\{\Sigma^*\}\} \cdot \&x \cdot \&y \cdot y\{\Sigma^*\} \cdot \&x \cdot \&y.$$

The construction from the proof of Lemma 3.19 leads to the equivalent xregex path

$$\gamma := \varepsilon \cdot x\{\Sigma^* \cdot y\{\Sigma^*\}\} \cdot \&x \cdot \&y \cdot \hat{y}\{\Sigma^*\} \cdot \&x \cdot \&\hat{y},$$

from which we derive the functional regex formula

$$\delta := x\left\{\Sigma^* y\{\Sigma^*\}\right\} x_1\{\Sigma^*\} y_1\{\Sigma^*\} \hat{y}\{\Sigma^*\} x_2\{\Sigma^*\} \hat{y}_1\{\Sigma^*\},$$

which we use in the spanner representation $\rho := \pi_\emptyset \zeta^{=}_{x,x_1,x_2} \zeta^{=}_{y,y_1} \zeta^{=}_{\hat{y},\hat{y}_1} \delta$. Then $\mathcal{L}(\alpha) = \mathcal{L}(\rho)$.

As these spanner representations are Boolean, they are also union compatible. Hence, we can now combine Lemma 3.17 and Lemma 3.19 to observe the following.

**Theorem 3.21** *There is an algorithm that, given* $\alpha \in \mathsf{vsfXR}$, *computes* $\rho \in \mathsf{RGX}^{\{\pi,\zeta^{=},\cup\}}$ *with* $\mathcal{L}(\rho) = \mathcal{L}(\alpha)$.

In Section 4.2, we use Theorem 3.21 together with the undecidability results from [12] to obtain multiple lower bounds for static analysis problems. Theorem 3.21 also raises the question whether every language that is recognized by a core spanner is also recognized by a vstar-free regular expression. As we have already seen in Example 3.8, it is possible to express the language

$$L_{\text{imp}} := \{w^n \mid w \in \Sigma^+, n \geq 2\}$$

with core spanners. Hence, under certain conditions, core spanners can simulate constructions like $(\&x)^*$.

While $L_{\text{imp}}$ might seem to be an obvious witness that separates the classes of languages that are recognized by core spanners and by vstar-free xregexes, proving this appears to be quite involved. Instead, we consider a related language, which allows us to use the following tool:

**Definition 3.22** Let $k \in \mathbb{N}_{>0}$. We call a set $A \subseteq \mathbb{N}^k$ *linear* if there exist an $r \geq 0$ and $m_0, \ldots, m_r \in \mathbb{N}^k$ with $A = \{m_0 + m_1 i_1 + m_2 i_2 + \cdots + m_r i_r \mid i_1, i_2, \ldots, i_r \in \mathbb{N}\}$. A set $A \subseteq \mathbb{N}^k$ is *semi-linear* if it is a finite union of linear sets. Assume $\Sigma = \{a_1, a_2, \ldots, a_k\}$ with $|\Sigma| = k$. The *Parikh map* $\Psi\colon \Sigma^* \to \mathbb{N}^k$ is defined by $\Psi(w) := (|w|_{a_1}, |w|_{a_2}, \ldots, |w|_{a_k})$, and is extended to languages by $\Psi(L) := \{\Psi(w) \mid w \in L\}$. We call $L$ *semi-linear* if $\Psi(L)$ is semi-linear.

According to Parikh's Theorem [32], every context-free language is semi-linear. Moreover, as shown by Ginsburg and Spanier [19], a set is semi-linear if and only if it is definable in Presburger arithmetic. Building on this, we state the following.

**Theorem 3.23** *For every $\alpha \in \mathsf{vsfXR}$, the language $\mathcal{L}(\alpha)$ is semi-linear.*

*Proof* In order to increase the readability, we prove the claim for the case $|\Sigma| = 2$ (the adaption to larger alphabets is obvious). We assume $\Sigma = \{\mathtt{a}, \mathtt{b}\}$ and define $\Psi(\mathtt{a}) := (1, 0)$ and $\Psi(\mathtt{b}) := (0, 1)$. Assume that $\mathsf{Vars}(\alpha) = \{x_1, \ldots, x_k\}$ for some $k \in \mathbb{N}_{>0}$.

It suffices to prove the claim for $\alpha \in \mathsf{XRP}$, as semi-linear sets are closed under union, and (according to Lemma 3.17) every vstar-free xregex is equivalent to a finite union of xregex paths.

As explained in the proof of Lemma 3.19 (in the construction of $\gamma$), we can also assume without loss of generality that every variable binding $x\{\cdots\}$ occurs exactly once in $\alpha$, and that no variable reference $\&x_i$ uses the default binding $\varepsilon$. In particular, this means that in every $\alpha$-parse tree, each variable $x_i$ stores exactly one word $w_i$.

Let $\alpha$ be an xregex path that satisfies these conditions. Our goal is to construct a Presburger formula $\varphi$ such that $\varphi(n^{\mathtt{a}}, n^{\mathtt{b}})$ is true if and only if $(n^{\mathtt{a}}, n^{\mathtt{b}}) \in \Psi(\mathcal{L}(\alpha))$. This formula will use variables $x_i^{\mathtt{a}}$ and $x_i^{\mathtt{b}}$ to represent $|w_i|_{\mathtt{a}}$ and $|w_i|_{\mathtt{b}}$, respectively. Recall that, due to our initial assumptions, each reference $\&x_i$ refers to the same word $w_i$; hence, we can safely define the corresponding variables $x_i^{\mathtt{a}}$ and $x_i^{\mathtt{b}}$ "globally" in $\varphi$.

Let $I \subseteq \{1, \ldots, k\}$. We use $\mathbf{x}$ and $\mathbf{x}_I$ as abbreviations for the sequences $x_1^{\mathtt{a}}, x_1^{\mathtt{b}}, \ldots, x_k^{\mathtt{a}}, x_k^{\mathtt{b}}$ and $\big(x_i^{\mathtt{a}}, x_i^{\mathtt{b}} : i \in I\big)$, and define

$$\varphi(n^{\mathtt{a}}, n^{\mathtt{b}}) := \exists \mathbf{x}\colon \varphi_\alpha(n^{\mathtt{a}}, n^{\mathtt{b}}, \mathbf{x}),$$

where $\varphi_\alpha$ with $\mathsf{Vars}(\alpha) = \{x_1, \ldots, x_k\}$ is constructed according to the following general procedure.

Given an xregex path $\gamma$, we define a Presburger formula $\varphi_\gamma$ as follows: First, as $\gamma$ is an xregex path, there is a decomposition $\gamma = \gamma_1 \cdot \gamma_2 \cdots \gamma_l$ ($l \in \mathbb{N}_{>0}$), where each $\gamma_i$

is either a proper regular expression, a variable reference, or a variable binding of the form $x\{\hat{\gamma}_i\}$ such that $\hat{\gamma}_i$ is also an xregex path. For each $\gamma_i$, we use variables $n_i^{\mathtt{a}}$ and $n_i^{\mathtt{b}}$ to denote the number of $\mathtt{a}$ or $\mathtt{b}$ that occur in the subword that is generated by $\gamma_i$.

We denote the set of all variables that are bound or referenced in $\gamma_i$ by

$$\mathsf{VarsBR}(\gamma_i) := \mathsf{Vars}(\gamma_i) \cup \{x \mid \&x \text{ occurs in } \gamma_i\}.$$

In a slight abuse of notation, we identify $\mathbf{x}_{\mathsf{VarsBR}(\gamma_i)}$ with the sequence $(x^{\mathtt{a}}, x^{\mathtt{b}} : x \in \mathsf{VarsBR}(\gamma_i))$.

Keeping this in mind, we define

$$\varphi_\gamma\left(n^{\mathtt{a}}, n^{\mathtt{b}}, \mathbf{x}_{\mathsf{VarsBR}(\gamma)}\right) := \exists n_1^{\mathtt{a}}, n_1^{\mathtt{b}}, \ldots n_l^{\mathtt{a}}, n_l^{\mathtt{b}}:$$
$$\left(\left(n^{\mathtt{a}} = n_1^{\mathtt{a}} + \cdots + n_l^{\mathtt{a}}\right) \wedge \left(n^{\mathtt{b}} = n_1^{\mathtt{b}} + \cdots + n_l^{\mathtt{b}}\right) \wedge \bigwedge_{i=1}^{l} \varphi_{\gamma_i}\left(n_i^{\mathtt{a}}, n_i^{\mathtt{b}}, \mathbf{x}_{\mathsf{VarsBR}(\gamma_i)}\right)\right),$$

where the Presburger formulas are defined as follows:

- If $\gamma_i$ is a proper regular expression, then as $\mathcal{L}(\gamma_i)$ is semi-linear (as a consequence of Parikh's theorem [32], every regular language is semi-linear). Hence, due to Ginsburg and Spanier [19], there is a Presburger formula $\hat{\varphi}_{\gamma_i}$ such that $\hat{\varphi}_{\gamma_i}(n^{\mathtt{a}}, n^{\mathtt{b}})$ is true if and only if $(n^{\mathtt{a}}, n^{\mathtt{b}}) \in \Psi(\mathcal{L}(\gamma_i))$. We define
$$\varphi_{\gamma_i}\left(n_i^{\mathtt{a}}, n_i^{\mathtt{b}}, \mathbf{x}_{\mathsf{VarsBR}(\gamma_i)}\right) := \hat{\varphi}_{\gamma_i}\left(n_i^{\mathtt{a}}, n_i^{\mathtt{b}}\right).$$
In order to avoid potential confusion, note that in this case $\mathbf{x}_{\mathsf{VarsBR}(\gamma_i)}$ is the empty sequence. This is due to the fact that $\gamma_i$ is a proper regular expression, which implies $\mathsf{VarsBR}(\gamma_i) = \emptyset$.
- If $\gamma_i = \&x_j$ for some $1 \leq j \leq l$, we define
$$\varphi_{\gamma_i}\left(n_i^{\mathtt{a}}, n_i^{\mathtt{b}}, \mathbf{x}_{\mathsf{VarsBR}(\gamma_i)}\right) := \left(n_i^{\mathtt{a}} = x_j^{\mathtt{a}}\right) \wedge \left(n_i^{\mathtt{b}} = x_j^{\mathtt{b}}\right).$$
- If $\gamma_i = x_j\{\delta\}$ for some $1 \leq j \leq l$ and some xregex path $\delta$, we define
$$\varphi_{\gamma_i}\left(n_i^{\mathtt{a}}, n_i^{\mathtt{b}}, \mathbf{x}_{\mathsf{VarsBR}(\gamma_i)}\right) := \left(n_i^{\mathtt{a}} = x_j^{\mathtt{a}}\right) \wedge \left(n_i^{\mathtt{b}} = x_j^{\mathtt{b}}\right) \wedge \varphi_\delta\left(n_i^{\mathtt{a}}, n_i^{\mathtt{b}}, \mathbf{x}_{\mathsf{VarsBR}(\delta)}\right).$$

While the definition recurses in the case of xregex paths that contain variable bindings (the third case in the definition of $\varphi_{\gamma_i}$ above), the formula $\varphi$ is still ensured to be finite and well-defined (as $\delta$ is always a subexpression of $\gamma$ and, hence, shorter).

Recall that by our initial assumption, for every variable $x_i$, each variable reference $\&x_i$ refers to the same word $w_i$. Taking this into account, we can prove that

$$\Psi(\mathcal{L}(\alpha)) = \{(n^{\mathtt{a}}, n^{\mathtt{b}}) \mid \varphi(n^{\mathtt{a}}, n^{\mathtt{b}}) \text{ is true}\}$$

via a straightforward structural induction. □

We use Theorem 3.23 to separate the classes of languages that are recognized by core spanners and by vstar-free xregexes:

**Lemma 3.24** *Let* $L_{nsl} := \{(\mathtt{ab}^m)^n \mid m, n \geq 2\}$ *and* $\rho := \zeta_{x,y}^{R_{com}}(x\{\mathtt{abb}^+\}y\{\Sigma^+\})$ *for* $\Sigma := \{\mathtt{a}, \mathtt{b}\}$. *Then* $L_{nsl} = \mathcal{L}(\rho)$, *but there is no* $\alpha \in \mathsf{vsfXR}$ *with* $\mathcal{L}(\alpha) = L_{nsl}$.

*Proof* Assume that there is an $\alpha \in \mathsf{vsfXR}$ with $\mathcal{L}(\alpha) = L_{\text{nsl}}$. By Theorem 3.23, $L_{\text{nsl}}$ must be semi-linear. Note that $\Psi(L_{\text{nsl}}) = \{(n, mn) \mid m, n \geq 2\}$. As semi-linear sets are closed under projection (cf. Ginsburg and Spanier [19]), this implies that the set $C := \{mn \mid m, n \geq 2\}$ is semi-linear, and due to closure under complementation (also cf. [19]), the set $P = \{p \mid p \text{ is prime, } p = 0, \text{ or } p = 1\}$ is semi-linear as well. However, semi-linear sets are finite unions of linear sets, and so $P$ contains a subset $P_{c,a} := \{c + an \mid n \in \mathbb{N}_{>0}\}$ of prime numbers for $c \geq 2$ and $a \geq 2$. Obviously, $c + ac = c(1 + a) \in P_{c,a}$, but $c(1 + a)$ is a composite number. Hence, there is no $\alpha \in \mathsf{vsfXR}$ with $\mathcal{L}(\alpha) = L_{\text{nsl}}$. □

We do not need the join operator to define non-semi-linear languages: Consider the core spanner representation $\rho$ from Example 3.14 with $\mathcal{L}(\rho) = L_{\text{nsl}}$. If we construct $\hat{\rho}$ as explained below that example, we obtain $\mathcal{L}(\hat{\rho}) = \{ww \mid w \in L_{\text{nsl}}\}$, which is also not semi-linear.

It is worth pointing out Lemma 3.24 does not resolve the open question from [7] whether there is a language that is recognized by a core spanner, but not by an xregex, as Theorem 3.23 only applies to vstar-free xregexes. We have already seen languages that are not semi-linear, but are recognized by xregexes: The language $L_{\text{nsl}}$ is recognized by $\alpha_{\text{nsl}} := x\{\mathtt{abb}^+\}\&x^+$; and a similar approach is used for the following language (which we already met in Example 2.4):

*Example 3.25* Let $\Sigma := \{\mathtt{a}\}$, and define the language $L_{\text{npr}} := \{\mathtt{a}^{mn} \mid m, n \geq 2\}$. In other words, $L_{\text{npr}}$ is the language of all words $\mathtt{a}^i$ with $i \geq 4$ such that $i$ is not a prime number. Let $\alpha_{\text{npr}} := x\{\mathtt{aa}^+\} \cdot (\&x)^+$. Then $\mathcal{L}(\alpha_{\text{npr}}) = L_{\text{npr}}$.

While $L_{\text{nsl}}$ and $L_{\text{npr}}$ are defined by very similar xregexes, the latter cannot be recognized by core spanners. In order to show this with a semi-linearity argument, we observe:

**Theorem 3.26** *Let $|\Sigma| = 1$ and let $P$ be a core spanner over $\Sigma$. Then $\mathcal{L}(P)$ is semi-linear.*

*Proof* We prove this by showing that on unary terminal alphabets, every $\mathsf{EC}^{\mathsf{reg}}$-language is semi-linear. Due to Theorem 3.12, this proves the claim.

Let $\Sigma = \{\mathtt{a}\}$, and consider any $\mathsf{EC}^{\mathsf{reg}}$-formula $\varphi(w)$ over $\Sigma$. We show that $\mathcal{L}(\varphi)$ is semi-linear by converting $\varphi$ into a Presburger formula $\hat{\varphi}$ for the set $\Psi(\mathcal{L}(\varphi)) = \{|w| \mid w \in \mathcal{L}(\varphi)\}$. We obtain $\hat{\varphi}$ by rewriting $\varphi$ in the following way:

1. Each quantifier $\exists x$ is replaced with $\exists \hat{x}$.
2. Each regular constraint $L_A(x)$ is replaced with a formula $\hat{\varphi}_A(\hat{x})$ for the set $\{|x| \mid x \in \mathcal{L}(A)\}$. As each $\mathcal{L}(A)$ is a regular language, this is possible according to Ginsburg and Spanier [19].
3. Each word equation $\eta_L = \eta_R$ is replaced with the equation $\text{sum}(\eta_L) = \text{sum}(\eta_R)$, where the function sum is defined by $\text{sum}(\mathtt{a}) := 1$, $\text{sum}(x) := \hat{x}$ for $x \in X$, and $\text{sum}(\alpha \cdot \beta) := \text{sum}(\alpha) + \text{sum}(\beta)$.

For example, the word equation $xaxyx = ayzzya$ is converted into the Presburger equation $\hat{x} + 1 + \hat{x} + \hat{y} + \hat{x} = 1 + \hat{y} + \hat{z} + \hat{z} + \hat{y} + 1$ (for $\Sigma = \{a\}$). Intuitively, each variable $\hat{x}$ in $\hat{\varphi}$ contains the length of $x$ in $\varphi$ (which, as $|\Sigma| = 1$, corresponds to the Parikh image of that word). Hence, the Presburger formula $\hat{\varphi}$ defines the set $\Psi(\mathcal{L}(\varphi))$. According to [19], this implies that $\Psi(\mathcal{L}(\varphi))$ is semi-linear, which means that $\mathcal{L}(\varphi)$ is semi-linear. This concludes the proof. □

Note that this construction only applies to unary alphabets, as this is the only case where there is a one-to-one correspondence between words and their Parikh images.

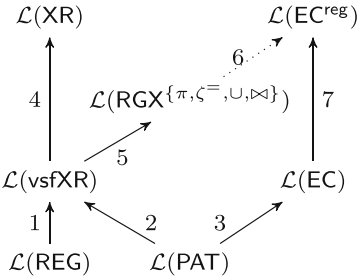
Apart from the observation that $L_{\text{npr}}$ from Example 3.25 is not recognized by core spanners, Theorem 3.26 also allows us to conclude the following.

**Corollary 3.27** *If $|\Sigma| = 1$, then $\mathcal{L}(P)$ is regular for every core spanner $P$.*

In other words, for unary terminal alphabets, core spanners recognize exactly the same class as regular spanners, namely the class of regular languages (which, in the unary case, is identical to the class of context-free languages). Furthermore, Lemma 3.24 and Theorem 3.26 together show the following.

**Corollary 3.28** *The class of languages that is recognized by core spanners is not closed under homomorphisms.*

We conclude this section with a summary of our insights into the relative expressive power of the various models. To increase readability, we use the following definitions: Let REG, XR, and PAT denote the class of regular expressions, xregex, or patterns, respectively. For a class of language recognizing mechanisms $\mathcal{D}$, let $\mathcal{L}(\mathcal{D})$ denote the class of languages that are recognized by elements of $\mathcal{D}$. For example, $\mathcal{L}(\mathsf{PAT})$ is the class of pattern languages, and $\mathcal{L}(\mathsf{RGX}^{\{\pi,\zeta^{=},\cup,\bowtie\}})$ is the class of languages that are recognized by core spanners. The hierarchy in Fig. 2 is obtained by combining the results in the present section with the fact that every pattern language contains either exactly one or infinitely many words (first observed by Angluin [1]), and that there are regular languages that are not EC-recognizable (see Karhumäki




1. by def.; $x\{a^*\}\&x$
2. obvious; Angluin [1]
3. by def.; Angluin [1]
4. by def.; Example 3.25 + Theorem 3.23
5. Theorem 3.21; Lemma 3.24
6. Theorem 3.12
7. by def.; Karhumäki et al. [26]

**Fig. 2** *To the left*: The relationship of the various language classes. An arrow denotes proper inclusion (of the source class in the target class), the dotted arrow denotes inclusion. *To the right*: The references for these results. See also the explanation at the end of Section 3

et al. [26]). Two sets of question remain open: Firstly, although Theorem 3.26 together with Example 3.25 shows that there is a language that is recognized by xregex, but not by $\mathsf{EC}^{\mathsf{reg}}$ (and, hence, also not by $\mathsf{EC}$ or $\mathsf{RGX}^{\{\pi,\zeta^{=},\cup,\bowtie\}}$), it remains open whether the reverse direction holds as well. Secondly, although we know that $\mathcal{L}(\mathsf{RGX}^{\{\pi,\zeta^{=},\cup,\bowtie\}}) \subseteq \mathcal{L}(\mathsf{EC}^{\mathsf{reg}})$, we do not know whether this inclusion is strict. In fact, it even remains open whether there is a language that is recognized by $\mathsf{EC}$, but not by $\mathsf{RGX}^{\{\pi,\zeta^{=},\cup,\bowtie\}}$. This second set of question is discussed in more detail in Freydenberger [13].

## 4 Decision Problems

### 4.1 Spanner Evaluation

We first examine the *combined complexity* of the evaluation problem for core spanners. To this end, we define the problem CSp−Eval: Given a core spanner representation $\rho \in \mathsf{RGX}^{\{\pi,\zeta^{=},\cup,\bowtie\}}$, a word $w \in \Sigma^*$, and a $(\mathsf{SVars}(\rho), w)$-tuple $\mu$, is $\mu \in [\![\rho]\!](w)$? In order to prove lower bounds for this problem, we consider the membership problem for pattern languages: Given a pattern $\alpha$ and a word $w$, decide whether $w \in \mathcal{L}(\alpha)$. As shown by Jiang et al. [24], this problem is NP-complete (for pattern languages that do not allow replacing variables with $\varepsilon$, this was already shown by Angluin [1]). Due to Theorem 3.3, we observe the following (the proof of NP-membership is straightforward).

**Theorem 4.1** CSp−Eval *is* NP*-complete, even if restricted to* $\mathsf{RGX}^{\{\pi,\zeta^{=}\}}$.

*Proof* In order to prove NP-hardness, it suffices to give a polynomial time reduction from the membership problem for pattern languages to CSp−Eval. Given a pattern $\alpha$ and a word $w$, we use Theorem 3.3 to construct a spanner representation $\rho_\alpha \in \mathsf{RGX}^{\{\zeta^{=}\}}$ in polynomial time such that $\mathcal{L}(\alpha) = \mathcal{L}(\rho_\alpha)$. Next, we define $\rho := \pi_\emptyset \rho_\alpha$. As $\rho$ represents a Boolean spanner, we define $\mu$ to be the empty tuple $()$. Now, $\mu \in [\![\rho]\!](w)$ holds if and only if $w \in \mathcal{L}(\alpha)$.

We prove membership in NP using the following NP-algorithm: Assume that we are given a core spanner representation $\rho$, a word $w \in \Sigma^*$, and a $w$-tuple $\mu$. For every regex formula $\gamma$ in $\rho$, we nondeterministically guess a $w$-tuple $\mu_\gamma$. By definition, each of these tuples has a size that is polynomial in $|w|$. In addition to this, for every union $(\rho_1 \cup \rho_2)$, we guess a representation $\rho_i$ that is ignored. We then verify these guesses deterministically: First, we discard all parts of $\rho$ that are ignored, and obtain a spanner representation $\hat{\rho} \in \mathsf{RGX}psj$. For all remaining regex formulas $\gamma$ in $\hat{\rho}$, we check whether $\mu_\gamma$ is consistent with $\gamma$ and $w$. Obviously, this can be done in polynomial time. If all of these checks pass, we evaluate all operators in $\hat{\rho}$. As $\hat{\rho}$ contains no unions, the result of these evaluations is always either $\emptyset$, or a set that contains exactly one $w$-tuple. Hence, this process only takes polynomial time. Furthermore, when it terminates, it results either in $\emptyset$, or in a $w$-tuple $\hat{\mu}$. In the latter case, we return `True` if $\hat{\mu} = \mu$. □

The question arises whether there are natural restrictions to CSp−Eval that make this problem tractable. It appears that any subclass of the core spanners that extends regular spanners in a meaningful way while having a tractable evaluation problem cannot be allowed to recognize the full class of pattern languages.

For pattern languages, it was shown by Ibarra et al. [23] that bounding the number of variables in the pattern leads to an algorithm for the membership problem with a running time that is polynomial, although in $\mathcal{O}(n^k)$ (where $n$ is the length of the word $w$, and $k$ the number of variables). From a parameterized complexity point of view (see e. g. Grohe and Flum [20]), this is usually not considered satisfactory. Without going too much into details, in parameterized complexity, one generally considers parameterized problems tractable that belong to the class FPT (from *fixed-parameter tractable*). This class is defined as follows: The input of a parameterized problem is a pair $(x, k)$, where $x$ is the input of the non-parameterized problem (e. g., a pattern $\alpha$ and a word $w$), and $k$ is a parameter of the input (e. g., the number of variables in $\alpha$). The parameterized problem is in FPT if there exist a computable function $f$, a constant $c \geq 0$, and an algorithm that decides the problem in time $O(f(k)n^c)$. We do not define the class W[1], but we note that the standard complexity theoretic assumption is that if a problem is W[1]-hard, it is not in FPT.

It was first observed by Stephan et al. [34] that the membership problem for pattern languages is W[1]-complete if the number of variable occurrences (not of variables) is used as a parameter (see Fernau et al. [11] for the full proof). As the number of variable occurrences in a pattern corresponds to the number of variables in an equivalent spanner, this implies that using the number of variables in a spanner as parameter leads to W[1]-hardness for this parameter of CSp−Eval.

Fernau and Schmid [10] and Fernau et al. [11] discuss these and various other potential restrictions to pattern languages that still do not lead to tractability (among these a bound on the length of the replacement of each variable, which corresponds to a bound on the length of spans). On the other hand, Reidenbach and Schmid [33] and Fernau et al. [9] examine parameters for patterns that make the membership problem tractable. While this does not directly translate to spanners, the authors consider these directions promising for further research.

But apart from these potential restrictions on the use of string equality, other restrictions are needed, as the use of join also makes evaluation intractable:

**Proposition 4.2** CSp−Eval *is* NP*-complete, even if restricted to* $\mathsf{RGX}^{\{\pi,\bowtie\}}$.

*Proof* We prove this with a reduction from the Clique problem: Given an undirected graph $G = (V, E)$ and a number $k \leq |V|$, decide whether $G$ contains a clique of size $k$. This problem is NP-complete (cf. Garey and Johnson [18]). Consider an undirected graph $G = (V, E)$ with $V = \{1, \ldots, n\}$ for some $n \geq 1$, and a number $k \leq n$. Let $\mathtt{a} \in \Sigma$ and define $w := \mathtt{a}^n$ and $\rho := \bowtie_{1 \leq i < j \leq k} \alpha_{i,j}$, where each $\alpha_{i,j}$ is defined by

$$\alpha_{i,j} := \bigvee_{\substack{\{u,v\} \in E, \\ u<v}} \mathtt{a}^{u-1}\, x_i\{\mathtt{a}\}\, \mathtt{a}^{v-u-1}\, x_j\{\mathtt{a}\}\, \mathtt{a}^{n-v}.$$

In other words, each part of the disjunction corresponds to a choice of $u$ and $v$, which allows $[\![\alpha_{i,j}]\!](w)$ to map $x_i$ to the $u$-th and $x_j$ to the $v$-th letter of $w$. Then $\mu \in [\![\rho]\!](w)$ holds if and only if there exist distinct nodes $v_1, \ldots, v_k \in V$ such that $\{v_i, v_j\} \in E$ for all $1 \leq i < j \leq k$; and $\mu(x_i) = [v_i, v_i + 1\rangle$ for $1 \leq i \leq k$. Thus, the empty tuple is an element of $[\![\pi_\emptyset \rho]\!](w)$ if and only if $G$ contains a clique of size $k$. □

We also consider the *data complexity* of the evaluation problem for core spanners. For every core spanner representation $\rho$ over $\Sigma$, we define the decision problem CSp−Eval($\rho$): Given a word $w \in \Sigma^*$ and a $w$-tuple $\mu$, is $\mu \in [\![\rho]\!](w)$? Using a slight variation of the proof of Theorem 4.1, we obtain the following.

**Theorem 4.3** CSp−Eval*($\rho$) is in* NLOGSPACE *for every* $\rho \in \mathsf{RGX}^{\{\pi, \zeta^=, \cup, \bowtie\}}$.

*Proof* This result follows from a slight change to the NP-decision procedure from the proof of Theorem 4.1: We can represent the guessed $w$-tuples $\mu_\gamma$ for each regex formula $\gamma$ by using two pointers for each $\mu_\gamma(x) = [i, j\rangle$ (one pointer for $i$, one for $j$). As $\rho$ is fixed, a finite number of such pointers suffices to represent all $w$-tuples. Furthermore, the verification of these guesses can also be realized nondeterministically with only a constant amount of additional pointers. □

### 4.2 Static Analysis

We consider the following common decision problems for core spanner representations, where the input is $\rho \in \mathsf{RGX}^{\{\pi, \zeta^=, \cup, \bowtie\}}$ or $\rho_1, \rho_2 \in \mathsf{RGX}^{\{\pi, \zeta^=, \cup, \bowtie\}}$:

1. CSp−Sat: Is $[\![\rho]\!](w) \neq \emptyset$ for some $w \in \Sigma^*$?
2. CSp−Hierarchicality: Is $[\![\rho]\!]$ hierarchical?
3. CSp−Universality: Is $[\![\rho]\!] = \Upsilon_{\mathsf{SVars}(\rho)}$?
4. CSp−Equivalence: Is $[\![\rho_1]\!] = [\![\rho_2]\!]$?
5. CSp−Containment: Is $[\![\rho_1]\!] \subseteq [\![\rho_2]\!]$?
6. CSp−Regularity: Is $[\![\rho]\!] \in [\![\mathsf{RGX}^{\{\pi, \cup, \bowtie\}}]\!]$?

We approach the first two of these problems by using Theorem 3.12 to convert core spanner representations to $\mathsf{EC}^{\mathsf{reg}}$-formulas, for which satisfiability is in PSPACE (cf. Diekert [6]). Hence, we observe:

**Theorem 4.4** *The problem* CSp−Sat *is* PSPACE*-complete, even if it is restricted to spanner representations from* $\mathsf{RGX}^{\{\zeta^=\}}$.

*Proof* We begin with the upper bound. According to Theorem 3.12, for every core spanner representation $\rho$, there exists an $\mathsf{EC}^{\mathsf{reg}}$-formula $\varphi$ that realizes $[\![\rho]\!]$. Furthermore, $\varphi$ can be computed in polynomial time. In particular, $\varphi$ is satisfiable if and only if $\rho$ is satisfiable. As satisfiability for $\mathsf{EC}^{\mathsf{reg}}$-formulas is in PSPACE (cf. Diekert [6]), this question can be answered in PSPACE.

For the lower bound, we construct a reduction to CSp−Sat from the intersection emptiness problem for regular expressions, which is defined as follows: Given

(proper) regular expressions $\alpha_1, \ldots, \alpha_n$, decide whether $\bigcap_{i=1}^{n} \mathcal{L}(\alpha_i) = \emptyset$. As a direct consequence of the proof of Lemma 3.2.3 in Kozen [27], this problem is PSPACE-complete (although Kozen's proof uses automata, these are defined via regular expressions). Recall that every proper regular expression is also a functional regex formula. Hence, we can construct a Boolean spanner representation

$$\rho := \zeta^{=}_{x_1,\ldots,x_n} x_1\{\alpha_1\} \cdots x_n\{\alpha_n\}.$$

Obviously, for every $w \in \Sigma^*$, we have $P(w) \neq \emptyset$ if and only if there exists a word $v \in \Sigma^*$ with $w = v^n$ and $v \in \mathcal{L}(\alpha_i)$ for $1 \leq i \leq n$. Hence, $P$ is satisfiable if and only if $\bigcap_{i=1}^{n} \mathcal{L}(\alpha_i) \neq \emptyset$. As PSPACE is closed under complementation, this proves PSPACE-hardness of CSp−Sat, even when restricted to representations from the class $\mathsf{RGX}^{\{\zeta^{=}\}}$. □

The proof of the lower bound in Theorem 4.4 uses the PSPACE-hardness of the intersection emptiness problem for regular expressions. But even if the variables in the regex formulas were only bound to $\Sigma^*$, it follows from Theorem 3.13 that this problem would still be at least as hard as the satisfiability problem for word equations without constraints. Considering that even proving the decidability was hard (see Diekert [6] for an overview), approaching CSp−Sat without knowledge on word equations would have required enormous additional effort.

It is also possible to use $\mathsf{EC}^{\mathsf{reg}}$-formulas to express a violation of the criteria for hierarchicality. This allows us to state the following result:

**Theorem 4.5** *The problem* CSp−Hierarchicality *is* PSPACE*-complete, even if it is restricted to* $\mathsf{RGX}^{\{\zeta^{=},\times\}}$.

*Proof* We begin with of the upper bound. The main idea is that non-hierarchicality can be expressed in $\mathsf{EC}^{\mathsf{reg}}$-formulas. Hence, our goal is to construct a polynomial time procedure that, given a core spanner representation $\rho \in \mathsf{RGX}^{\{\pi,\zeta^{=},\cup,\bowtie\}}$, constructs an $\mathsf{EC}^{\mathsf{reg}}$-formula $\varphi_{\mathrm{NH}}$ that is satisfiable if and only if $[\![\rho]\!]$ is not hierarchical.

Recall that, by definition, for every spanner $P$ and every word $w \in \Sigma^*$, a $w$-tuple $\mu \in P(w)$ is not hierarchical if there exist variables $x, y \in \mathsf{SVars}(P)$ such that all of the following hold:

1. The span $\mu(x)$ does not contain $\mu(y)$,
2. the span $\mu(y)$ does not contain $\mu(x)$, and
3. the spans $\mu(x)$ and $\mu(y)$ overlap (i. e., they are not disjoint).

If this is the case, we say that $\mu(x)$ and $\mu(y)$ *strictly overlap*. It is easy to see that two spans $[i_1, j_1\rangle$ and $[i_2, j_2\rangle$ strictly overlap if one of the following *strict overlap conditions* is met:

1. $i_1 < i_2 < j_1 < j_2$,
2. $i_2 < i_1 < j_2 < j_1$.

For an illustration of these two conditions, see Fig. 3. Our next goal is to define an $\mathsf{EC}^{\mathsf{reg}}$-formula $\varphi_{\mathrm{ovl}}(x^P, x^C, y^P, y^C)$ that expresses the first condition when

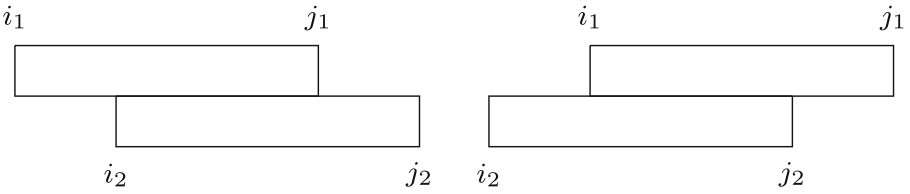



**Fig. 3** The two possibilities how two spans can strictly overlap (see proof of Theorem 4.5). To the left: $i_1 < i_2 < j_1 < j_2$. To the right: $i_2 < i_1 < j_2 < j_1$

combined with an $\mathsf{EC}^{\mathsf{reg}}$-formula that realizes a spanner (we do not need to define a formula for the second condition, as both conditions are symmetrical). To this purpose, we first define the $\mathsf{EC}^{\mathsf{reg}}$-formula

$$\varphi_{\text{ppref}}(x, y) := \exists z : (L_A(z) \wedge (xz = y)),$$

where $A$ is an NFA with $\mathcal{L}(A) = \Sigma^+$. Clearly, $(x, y) \in \Sigma^* \times \Sigma^*$ satisfies $\varphi_{\text{ppref}}$ if and only if $x$ is a proper prefix of $y$. Next, we define

$$\begin{aligned}\varphi_{\text{ovl}}(x^P, x^C, y^P, y^C) := \\ \exists z_1, z_2 : \ &((z_1 = x^P x^C) \wedge (z_2 = y^P y^C) \\ &\wedge \varphi_{\text{ppref}}(x^P, y^P) \wedge \varphi_{\text{ppref}}(y^P, z_1) \wedge \varphi_{\text{ppref}}(z_1, z_2)).\end{aligned}$$

The idea behind the construction is as follows: Recall that this formula is going to be used together with an $\mathsf{EC}^{\mathsf{reg}}$-formula that realizes a spanner. Hence, $x^P$ and $x^C$ represent a span $[1 + |x^P|, 1 + |x^P x^C|\rangle = [i_1, j_1\rangle$, while $y^P$ and $y^C$ represent a span $[1 + |y^P|, 1 + |y^P y^C|\rangle = [i_2, j_2\rangle$. In particular, $x^P x^C$ and $y^P y^C$ are both prefix of some common word $w$. Hence, $i_1 < i_2$ holds if and only if $x^P$ is a proper prefix of $y^P$. Likewise, $i_2 < j_1$ and $j_1 < j_2$ hold if and only if $y^P$ is a proper prefix of $x^P x^C$, or $x^P x^C$ is a proper prefix of $y^P y^C$, respectively.

In other words, $\varphi_{\text{ovl}}$ checks whether the first of the two strict overlap conditions is satisfied.

We are now ready to construct $\varphi_{\text{NH}}$. Let $\rho \in \mathsf{RGX}^{\{\pi, \zeta^=, \cup, \bowtie\}}$, and assume that $\mathsf{SVars}(\rho) = \{x_1, \ldots, x_n\}$ for some $n \geq 2$ (spanners with less than two variables are trivially hierarchical). Using Theorem 3.12), we then construct an $\mathsf{EC}^{\mathsf{reg}}$-formula $\varphi_\rho(x_w, x_1^P, x_1^C, \ldots, x_n^P, x_n^C)$ that realizes $[\![\rho]\!]$. We now define

$$\varphi_{\text{NH}} := \exists x_w, x_1^P, x_1^C, \ldots, x_n^P, x_n^C : \left( \varphi_\rho\left(x_w, x_1^P, x_1^C, \ldots, x_n^P, x_n^C\right) \wedge \bigvee_{\substack{1 \leq i, j \leq n; \\ i \neq j}} \varphi_{\text{ovl}}\left(x_i^P, x_i^C, x_j^P, x_j^C\right) \right).$$

Assume that $[\![\rho]\!]$ is not hierarchical. Then there exist a word $w \in \Sigma^*$, a $w$-tuple $\mu \in [\![\rho]\!]$, and $x_l, x_m \in \mathsf{SVars}(\rho)$ such that $\mu(x_l)$ and $\mu(x_m)$ strictly overlap. As $\varphi_\rho$ realizes $[\![\rho]\!]$, we have that $\mu$ defines an assignment $(w, w_{[1,i_1\rangle}, w_{[i_1,j_1\rangle}, \ldots, w_{[1,i_n\rangle}, w_{[i_n,j_n\rangle})$ that satisfies this subformula (where $[i_k, j_k\rangle = \mu(x_k)$). Furthermore, as $\mu(x_m)$ and $\mu(x_l)$ strictly overlap, either

$\varphi_{\text{ovl}}(x_l^P, x_l^C, x_m^P, x_m^C)$ or $\varphi_{\text{ovl}}(x_m^P, x_m^C, x_l^P, x_l^C)$ is satisfied (if $i_l < i_m$ or $i_m < i_l$, respectively). Hence, $\varphi_{\text{NH}}$ is satisfiable.

Likewise, $\varphi_{\text{NH}}$ is only satisfied if $\varphi_\rho$ and (at least) one $\varphi_{\text{ovl}}(x_l^P, x_l^C, x_m^P, x_m^C)$ are satisfied. This corresponds to a $w$-tuple $\mu$ where $\mu(x_l)$ and $\mu(x_m)$ strictly overlap. Hence, $\mu$ is not hierarchical, which means that $[\![\rho]\!]$ is not hierarchical.

Therefore, $\varphi_{\text{NH}}$ is satisfiable if and only if $[\![\rho]\!]$ is not hierarchical. Furthermore, $\varphi_{\text{NH}}$ can be constructed in polynomial time, as we only need to construct $\varphi_\rho$ (which is possible in polynomial time, according to the proof of Theorem 4.4), and an amount of $\varphi_{\text{ovl}}$-formulas that is quadratic in $|\mathsf{SVars}(\rho)|$, each of which has a constant length. Both constructions rely solely on the syntax of $\rho$, and require no further computation.

As satisfiability of $\mathsf{EC}^{\mathsf{reg}}$-formulas can be decided in PSPACE, the complement of CSp−Hierarchicality is in PSPACE; and as PSPACE is closed under complementation, this means that CSp−Hierarchicality is in PSPACE.

For the lower bound, we slightly modify the proof of the lower bound for CSp−Sat. Again, we use the intersection emptiness problem for regular expressions. Given proper regular expressions $\alpha_1, \ldots, \alpha_n$, we define

$$\rho := \zeta^{=}_{x_1,\ldots,x_n}(x_1\{\mathtt{aaa}\cdot\alpha_1\}\cdots x_n\{\mathtt{aaa}\cdot\alpha_n\}) \times (y\{\Sigma\cdot\Sigma^+\}\cdot\Sigma) \times (\Sigma\cdot z\{\Sigma^+\cdot\Sigma\}),$$

for some $\mathtt{a} \in \Sigma$. By replacing each $\alpha_i$ in that proof with $\mathtt{aaa}\cdot\alpha_i$, we ensure that every word $w \in \Sigma^*$ with $[\![\zeta^{=}_{x_1,\ldots,x_n}(x_1\{\mathtt{aaa}\cdot\alpha_1\}\cdots x_n\{\mathtt{aaa}\cdot\alpha_n\}]\!](w) \neq \emptyset$ has at least length 3 (which is the minimal word length for which non-hierarchical spanners are possible). Furthermore, for each such $w$, the variable $y$ is assigned the span that contains all positions of $w$ except the last one, and $z$ is assigned the span that contains all positions except the first one. Hence, these spans strictly overlap, which means that $\rho$ is not hierarchical. On the other hand, if $[\![\zeta^{=}_{x_1,\ldots,x_n}(x_1\{\mathtt{aaa}\cdot\alpha_1\}\cdots x_n\{\mathtt{aaa}\cdot\alpha_n\})]\!](w) = \emptyset$, then $[\![\rho]\!] = \emptyset$. Therefore, $\rho$ is hierarchical if and only if there is no $w \in \bigcap_{1\leq i\leq n}\mathcal{L}(\alpha_i)$. As this problem is PSPACE-complete, CSp−Hierarchicality is PSPACE-hard. □

For the remaining problems, we use Theorem 3.21, and the fact that the undecidability results from Freydenberger [12] also hold for vstar-free xregexes:

**Theorem 4.6** *The problems* CSp−Universality *and* CSp−Equivalence *are not semi-decidable, but co-semi-decidable. The problem* CSp−Regularity *is neither semi-decidable, nor co-semi-decidable. These results hold even if the input is restricted to* $\mathsf{RGX}^{\{\pi,\zeta^{=},\cup\}}$.

*Proof* The co-semi-decidability of the first two problems is obvious. We discuss this for universality: For any core spanner representation $\rho$, we can always decide whether $[\![\rho]\!](w) = \Upsilon_{\mathsf{SVars}(\rho)}(w)$ holds. Hence, we can semi-decide non-universality by enumerating all $w \in \Sigma^*$ until we find a word $w$ with $[\![\rho]\!](w) \neq \Upsilon_{\mathsf{SVars}(\rho)}(w)$. Thus, CSp−Universality is co-semi-decidable. The proof for CSp−Equivalence works analogously.

We now proceed to the proofs of the lower bounds. As shown by Freydenberger [12], if $|\Sigma| \geq 2$, for xregexes $\alpha$, the following holds:

– It is not semi-decidable whether $\mathcal{L}(\alpha) = \Sigma^*$,
– It is neither semi-decidable, nor co-semi-decidable whether $\mathcal{L}(\alpha)$ is a regular language.

The proof in [12] takes a Turing machine $\mathcal{X}$ (with some additional technical restrictions) and computes an xregex $\alpha_{\mathcal{X}}$ with a single variable $x$ such that $\mathcal{L}(\alpha) = \Sigma^*$ if and only if $\mathcal{X}$ accepts no input, and $\mathcal{L}(\alpha_{\mathcal{X}})$ is regular if and only if $\mathcal{X}$ accepts only finitely many inputs.

These xregexes $\alpha_{\mathcal{X}}$ are defined over the alphabet $\Sigma = \{0, \#\}$ and, when adapted to the notation of this paper, are always of the following shape:

$$\alpha_{\mathcal{X}} = \alpha_{struc} \vee \alpha_{state} \vee \alpha_{head} \vee \alpha_{mod} \vee \alpha_{var}.$$

It is important to note that all subexpressions except $\alpha_{var}$ are proper regular expressions, while

$$\alpha_{var} = (0 \vee \#)^*\#0 \cdot x\{0^*\} \cdot (\alpha_1 \vee \alpha_2 \vee \cdots \vee \alpha_n)$$

for some $n \in \mathbb{N}p$ that depends on $\mathcal{X}$, where all $\alpha_i$ are xregex paths that do not contain variable bindings, and no other variable references than $\&x$.

We note that the single variable biding $x\{0^*\}$ and all variable references $\&x$ do not occur under a Kleene star, and conclude that $\alpha_{\mathcal{X}}$ is a vstar-free xregex.

By Theorem 3.21, we can effectively convert every $\alpha_{\mathcal{X}}$ into a Boolean spanner representation $\rho_{\mathcal{X}} \in \mathsf{RGX}^{\{\pi,\zeta^=,\cup\}}$ with $\mathcal{L}(\rho_{\mathcal{X}}) = \mathcal{L}(\alpha_{\mathcal{X}})$.

Then $[\![\rho_{\mathcal{X}}]\!] = \Upsilon_\emptyset$ holds if and only if $\mathcal{L}(\alpha_{\mathcal{X}}) = \Sigma^*$. As this question is not semi-decidable, CSp−Universality is also not semi-decidable. As CSp−Universality is a special case of CSp−Equivalence, the latter problem is also not semi-decidable.

Furthermore, $[\![\rho_{\mathcal{X}}]\!]$ is a regular spanner if and only if $\mathcal{L}(\alpha_{\mathcal{X}})$ is a regular language (as shown by Fagin et al. [7], when viewed as language definition mechanisms, regular spanners define exactly the class of regular languages). This question is neither semi-decidable, nor co-semi-decidable; hence, this applies to CSp−Regularity as well. □

As the proof of Theorem 4.6 relies only on Boolean spanners, the decidability status of CSp−Regularity does not change if the problem asks for hierarchical regularity (i. e., membership in $[\![\mathsf{RGX}]\!]$) instead of regularity, as the two classes coincide for Boolean spanners. Likewise, CSp−Universality remains not semi-decidable if one replaces $\Upsilon_{\mathsf{SVars}(\rho)}$ with $\Upsilon^H_{\mathsf{SVars}(\rho)}$.

In the construction from this proof, variables are only bound to a language $a^+$. Hence, the same undecidability results hold for spanners that use selections by equal length relation, instead of the string equality relation. While the proof builds on xregexes $\alpha_{\mathcal{X}}$ that use only a single variable $x$, the resulting core spanners use an unbounded amount of variables, as every occurrence of a variable reference $\&x$ in an xregex path is converted to a spanner variable $x_i$. But undecidability remains even if we bound the number of variables in the spanners, as the $\alpha_{\mathcal{X}}$ can be modified to use only a bounded number of variable references (see Section 4.1 in [12]). Theorem 4.6 also implies that CSp−Containment is not semi-decidable. This holds even for a more restricted class of spanners:

**Theorem 4.7** *The problem* CSp−Containment *is not semi-decidable, even if it is restricted to* $\mathsf{RGX}^{\{\pi,\zeta^=\}}$.

*Proof* This proof uses the undecidability of the inclusion problem for pattern languages, which is defined as follows: Given two patterns $\alpha$ and $\beta$, decide whether $\mathcal{L}(\alpha) \subseteq \mathcal{L}(\beta)$.

For unbounded sizes of $\Sigma$, this undecidability was proven by Jiang et al. [25], and Freydenberger and Reidenbach [15] adapted this proof to all (non-unary) finite terminal alphabets.

Given two patterns $\alpha, \beta$, we can use Theorem 3.3 to construct spanner representations $\rho_\alpha, \rho_\beta \in \mathsf{RGX}^{\{\zeta^=\}}$ with $\mathcal{L}(\rho_X) = \mathcal{L}(X)$ for $X \in \{\alpha, \beta\}$, and turn these into representations of Boolean spanners $\hat{\rho}_X := \pi_\emptyset \rho_X$. Then $[\![\hat{\rho}_\alpha]\!](w) \subseteq [\![\hat{\rho}_\beta]\!](w)$ holds for all $w \in \Sigma^*$ if and only if $\mathcal{L}(\alpha) \subseteq \mathcal{L}(\beta)$.

This shows that CSp−Containment is not decidable. As it is obviously co-semi-decidable, this also shows that CSp−Containment is not semi-decidable. □

As shown by Bremer and Freydenberger [4], the inclusion problem for pattern languages remains undecidable if the number of variables in the patterns is bounded. In fact, that proof constructs patterns where even the number of variable occurrences is bounded. Therefore, CSp−Containment is not semi-decidable even if restricted to representations from $\mathsf{RGX}^{\{\pi,\zeta^=\}}$ with a bounded number of variables. It is a hard open question whether the equivalence problem for pattern languages is decidable (cf. Ohlebusch and Ukkonen [31], Freydenberger and Reidenbach [15]). Undecidability of this problem would imply undecidability of CSp−Equivalence, even if restricted to representations from $\mathsf{RGX}^{\{\pi,\zeta^=\}}$.

We conclude this part of the section with a table that summarizes our results on decision problems:

| Problem | Status | Reference |
|---|---|---|
| CSp−Eval | NP-complete | Theorem 4.1, Proposition 4.2 |
| CSp−Eval($\rho$) | in NLOGSPACE | Theorem 4.3 |
| CSp−Sat | PSPACE-complete | Theorem 4.4 |
| CSp−Hierarchicality | PSPACE-complete | Theorem 4.5 |
| CSp−Universality | co-semi-decidable, not semi-decidable | Theorem 4.6 |
| CSp−Equivalence | co-semi-decidable, not semi-decidable | Theorem 4.6 |
| CSp−Containment | co-semi-decidable, not semi-decidable | Theorem 4.7 |
| CSp−Regularity | neither semi-, nor co-semi-decidable | Theorem 4.6 |

Details under which restrictions the lower bounds persist can be found in the respective results.

#### 4.2.1 Minimization and Relative Succinctness

In order to address the minimization of spanner representations, we first formalize the notion of the size or complexity of a spanner representation. Even for proper regular expressions, there are various different definitions of size, see e. g. Holzer and Kutrib [22], and there might be convincing reasons to add additional weight to the number of variables or other parameters. As we shall see, these distinctions do not affect the negative results that we prove later. Hence, instead of defining a single fixed notion of size, we use the following general definition of complexity measures from Kutrib [29]:

**Definition 4.8** Let $\mathsf{SR}$ be a class of spanner representations. A *complexity measure* for $\mathsf{SR}$ is a recursive function $c\colon \mathsf{SR} \to \mathbb{N}$ such that for each $\Sigma$, the set of all $\rho \in \mathsf{SR}$ that represent spanners over $\Sigma$ can be effectively enumerated in order of increasing $c(\rho)$, and does not contain infinitely many $\rho \in \mathsf{SR}$ with the same value $c(\rho)$.

By *recursive*, we mean a function that is total and computable. Definition 4.8 is general enough to include all notions of complexity that take into account that descriptions are commonly encoded with a finite number of distinct symbols, and that it should be decidable if a word over these symbols is a valid encoding from $\mathsf{SR}$. Regardless of the chosen complexity measure, computable minimization of core spanners is impossible:

**Theorem 4.9** *Let $c$ be a complexity measure for $\mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$. There is no algorithm that, given a $\rho \in \mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$, computes an equivalent $\hat{\rho} \in \mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$ that is $c$-minimal.*

*Proof* Define $\mathsf{U}_{\mathsf{min}}$ to be the set of $c$-minimal core spanner representations of $\Upsilon_\emptyset$. By the definition of a complexity measure, $\mathsf{U}_{\mathsf{min}}$ is finite. Hence, given a core spanner representation $\rho$, we can decide whether $\rho \in \mathsf{U}_{\mathsf{min}}$.

Now assume there is an algorithm $\mathsf{MIN}_c$ that minimizes core spanner representations with respect to $c$. Given a core spanner representation $\rho$, we can decide whether $[\![\rho]\!] = [\![\Upsilon_\emptyset]\!]$, by checking whether $\mathsf{MIN}_c(\rho) \in \mathsf{U}_{\mathsf{min}}$. But as shown in Theorem 4.6, this problem is undecidable. Hence, $\mathsf{MIN}_c$ cannot exist. □

In addition to regex formulas, Fagin et al. [7] also define spanner representations that are based on so-called vset- and vstk-automata (denoted by $\mathsf{VA}_{\mathsf{set}}$ and $\mathsf{VA}_{\mathsf{stk}}$). They show $[\![\mathsf{VA}_{\mathsf{stk}}]\!] = [\![\mathsf{RGX}]\!]$ and $[\![\mathsf{VA}_{\mathsf{set}}]\!] = [\![\mathsf{RGX}^{\{\pi,\cup,\bowtie\}}]\!]$, and conclude that $[\![\mathsf{VA}_{\mathsf{set}}{}^{\{\pi,\zeta^=,\cup,\bowtie\}}]\!] = [\![\mathsf{VA}_{\mathsf{stk}}{}^{\{\pi,\zeta^=,\cup,\bowtie\}}]\!] = [\![\mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}]\!]$. Without going futher into details, we note that their equivalence proofs use computable conversions between the models. Hence, Theorem 4.9 also applies to those spanner representations from [7] that can express core spanners, like $\mathsf{VA}_{\mathsf{stk}}{}^{\{\pi,\zeta^=,\cup,\bowtie\}}$ and $\mathsf{VA}_{\mathsf{set}}{}^{\{\pi,\zeta^=,\cup,\bowtie\}}$, and it implies that an algorithm that converts from one of these classes of representations to another cannot guarantee that its result is minimal.

Using a technique by Hartmanis [21], we can use the fact that CSp−Regularity is not co-semi-decidable to compare the relative succinctness of regular and core spanner representations:

**Theorem 4.10** *Let $c_1$ and $c_2$ be complexity measures for the classes $\mathsf{RGX}^{\{\pi,\cup,\bowtie\}}$ and $\mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$, respectively. For every recursive function $f : \mathbb{N} \to \mathbb{N}$, there exists a $\rho \in \mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$ such that $[\![\rho]\!] \in [\![\mathsf{RGX}^{\{\pi,\cup,\bowtie\}}]\!]$, but $c_1(\hat{\rho}) > f(c_2(\rho))$ holds for every $\hat{\rho} \in \mathsf{RGX}^{\{\pi,\cup,\bowtie\}}$ with $[\![\hat{\rho}]\!] = [\![\rho]\!]$.*

*Proof* For the sake of a contradiction, assume that there exist complexity measures $c_1$ for $\mathsf{RGX}^{\{\pi,\cup,\bowtie\}}$ and $c_2$ for $\mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$, as well as a recursive function $f$ such that, for every core spanner representation $\rho \in \mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$ with $[\![\rho]\!] \in [\![\mathsf{RGX}^{\{\pi,\cup,\bowtie\}}]\!]$, there exists a regular spanner representation $\hat{\rho} \in \mathsf{RGX}^{\{\pi,\cup,\bowtie\}}$ with $[\![\hat{\rho}]\!] = [\![\rho]\!]$ and $c_1(\hat{\rho}) \leq f(c_2(\rho))$. Our goal is to show that this implies that the set

$$\mathsf{NR} := \{\rho \in \mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}} \mid \text{there is no } \rho_R \in \mathsf{RGX}^{\{\pi,\cup,\bowtie\}} \text{ with } [\![\rho]\!] = [\![\rho_R]\!]\}$$

is semi-decidable. As CSp−Regularity is not co-semi-decidable (Theorem 4.6), this will yield the desired contradiction.

We define a semi-decision procedure for NR as follows: Given a core spanner $\rho \in \mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$, compute a complexity bound $n := f(c_2(\rho))$. We define

$$F_n := \{\rho_R \in \mathsf{RGX}^{\{\pi,\cup,\bowtie\}} \mid c_1(\rho_R) \leq n\}.$$

By Definition 4.8, the set $F_n$ is finite, and we can effectively enumerate its elements $\rho_1, \ldots, \rho_k$ for $k := |F_n|$.

Also by definition, we know that if there exists a $\rho_R \in \mathsf{RGX}^{\{\pi,\cup,\bowtie\}}$ with $[\![\rho_R]\!] = [\![\rho]\!]$, there exists a $\hat{\rho}_R \in \mathsf{RGX}^{\{\pi,\cup,\bowtie\}}$ with $[\![\hat{\rho}_R]\!] = [\![\rho]\!]$ and $\hat{\rho}_R \in F_n$. In other words: If $[\![\rho]\!]$ is expressible with regular spanners, it is expressible with a regular spanner representation $\hat{\rho}$ that satisfies the complexity bound $n$.

For all $\rho_i \in F_n$, we now semi-decide $[\![\rho]\!] \neq [\![\rho_i]\!]$. In order to do this, we enumerate all $w \in \Sigma^*$. In each step, if $[\![\rho]\!](w) \neq [\![\rho_i]\!](w)$ holds, we mark $\rho_i$ as not equivalent to $\rho$.

If all spanners in $F_n$ are marked, we know that no regular spanner $[\![\rho_R]\!]$ with $[\![\rho_R]\!] = [\![\rho]\!]$ exists, and put out `True`. As $F_n$ is finite, this point is reached in a finite number of steps if there is no such spanner. On the other hand, if such a spanner exists, the procedure will never terminate. Hence, we have defined a semi-decision procedure for NR, which implies that CSp−Regularity is co-semi-decidable, a contradiction to Theorem 4.6. □

Hence, the blowup from $\mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$ to $\mathsf{RGX}^{\{\pi,\cup,\bowtie\}}$ is not bounded by any recursive function. As above, we can replace each of this classes with a class with the same expressive power; for example, we can replace $\mathsf{RGX}^{\{\pi,\cup,\bowtie\}}$ with $\mathsf{VA}_{\mathsf{stk}}^{\{\pi,\cup,\bowtie\}}$, $\mathsf{VA}_{\mathsf{set}}$, or $\mathsf{VA}_{\mathsf{set}}^{\{\pi,\cup,\bowtie\}}$ (or, as the proof uses Boolean spanners, RGX or $\mathsf{VA}_{\mathsf{stk}}$, or any class between those).

We also consider the relative succinctness of representations of core spanners and representations of their complements. For every spanner $P$, we define its *complement* $compl(P) := \Upsilon_{\mathsf{Vars}(P)} \setminus P$, and its *hierarchical complement* $compl\,H(P) := \Upsilon^{H}_{\mathsf{Vars}(P)} \setminus P$.

**Theorem 4.11** *Let c be a complexity measure for* $\mathsf{RGX}^{\{\pi,\zeta^{=},\cup,\bowtie\}}$. *For every recursive function* $f : \mathbb{N} \to \mathbb{N}$, *there exists a* $\rho \in \mathsf{RGX}^{\{\pi,\zeta^{=},\cup,\bowtie\}}$ *such that*

1. $\mathsf{C}([\![\rho]\!]) \in [\![\mathsf{RGX}^{\{\pi,\zeta^{=},\cup,\bowtie\}}]\!]$, *but*
2. $c(\rho) > f(c(\hat{\rho}))$ *holds for every* $\hat{\rho} \in \mathsf{RGX}^{\{\pi,\zeta^{=},\cup,\bowtie\}}$ *with* $[\![\hat{\rho}]\!] = compl([\![\rho]\!])$.

*This also holds if we consider* $\mathsf{C}^{\mathbf{H}}$ *instead of* $\mathsf{C}$.

*Proof* It suffices to prove the claim for Boolean core spanner representations (hence, we can focus on the case of $\mathsf{C}$, and do not need to consider $\mathsf{C}^{\mathbf{H}}$ separately). For convenience, we define the set of all Boolean core spanner representations

$$\mathsf{BCSR} := \{\rho \in \mathsf{RGX}^{\{\pi,\zeta^{=},\cup,\bowtie\}} \mid \mathsf{SVars}(\rho) = \emptyset\}.$$

As preparation for the actual proof, we consider the following sets of Boolean core spanner representations:

$$\mathsf{FIN} := \{\rho \in \mathsf{BCSR} \mid \mathcal{L}(\rho) \text{ is finite}\},$$
$$\mathsf{COF} := \{\rho \in \mathsf{BCSR} \mid \mathcal{L}(\rho) \text{ is co-finite}\}.$$

This proof heavily relies on various sets from the first two levels of the arithmetic hierarchy (cf. Kozen [28]). Without going into further details, note that $\Sigma^0_1$ is the family of all sets that are semi-decidable (recursively enumerable), $\Pi^0_1$ is the family of all thats that are co-semi-decidable (co-recursively enumerable), and $\Delta^0_1 = \Sigma^0_1 \cap \Pi^0_1$ is the family of all sets that are decidable.

Regarding the next level, $\Sigma^0_2$ is the family of all sets that are semi-decidable when using oracles for sets in $\Sigma^0_1$ (or in $\Pi^0_1$), $\Pi^0_2$ is the family of all sets that are co-semi-decidable when using such oracles. Finally, $\Delta^0_2 = \Sigma^0_2 \cap \Pi^0_2$ is the family of all sets that are decidable when using oracles for sets in $\Sigma^0_1$ or in $\Pi^0_1$. □

A central part of our reasoning in this proof is the following observation:

*Claim 1* $\mathsf{COF} \notin \Delta^0_2$.

*Proof* As shown in Freydenberger [12], the xregexes that we used in the proof of Theorem 4.6 also prove that co-finiteness for vstar-free xregexes is $\Sigma^0_2$-complete.

Hence, the proof of Theorem 4.6 also implies that $\mathsf{COF}$ is $\Sigma^0_2$-hard. This immediately implies $\mathsf{COF} \notin \Delta^0_2$; as otherwise, $\Sigma^0_2 = \Delta^0_2$ would hold, which contradicts the fact that the arithmetical hierarchy is a proper hierarchy. □ (Claim 1)

Our goal is to use Claim 1 to obtain the contradiction on which this proof rests. More precisely, we shall prove that any recursive bound on the size of the core spanner for a complement can be used to prove $\mathsf{COF} \in \Delta_2^0$. One of the central parts of our reasoning shall be the following result.

*Claim 2* $\mathsf{FIN} \in \Sigma_1^0$.

*Proof* We give the following semi-decision procedure for FIN. Let $\rho \in \mathsf{BCSR}$. Enumerate all finite sets $S \subset \Sigma^*$. For each set, we check the following two conditions:

1. $S \subseteq \mathcal{L}(\rho)$
2. $\mathcal{L}(\rho) \cap (\Sigma^* \setminus S) = \emptyset$

Note that both conditions are decidable: As $S$ is finite, the first condition can be checked by deciding if $w \in \mathcal{L}(\rho)$ for each $w \in S$.

For the second condition, we first construct a regular expression $\alpha$ with $\mathcal{L}(\alpha) = (\Sigma^* \setminus S)$. Then, we define the Boolean core spanner representation $\rho_S := \alpha \cap \rho$. As $\mathcal{L}(\rho_S) = \mathcal{L}(\alpha) \cap \mathcal{L}(\rho) = (\Sigma^* \setminus S) \cap \mathcal{L}(\rho)$, we can decide the second condition by checking if $\mathcal{L}(\rho_S) = \emptyset$ (which is decidable, according to Theorem 4.4).

If $S$ satisfies both conditions, $S = \mathcal{L}(\rho)$ holds. Hence, $\mathcal{L}(\rho)$ is finite, and the semi-decision procedure returns `True`. Furthermore, for every $\rho \in \mathsf{FIN}$, the procedure will (after a finite number of enumerated finite sets) check the set $S = \mathcal{L}(\rho)$, and then return `True`. Thus, FIN is semi-decidable, which is equivalent to $\mathsf{FIN} \in \Sigma_1^0$. □ (Claim 2)

The next observation is not very deep; but in order to streamline the flow of our later reasoning, we state it as a separate claim.

*Claim 3* For every $\rho \in \mathsf{BCSR}$, we have that $\rho \in \mathsf{COF}$ holds if and only if there is a $\hat{\rho} \in \mathsf{FIN}$ with $[\![\hat{\rho}]\!] = \mathsf{C}([\![\rho]\!])$.

*Proof* Let $\rho \in \mathsf{BCSR}$. We begin with the *if*-direction. Assume there exists a $\hat{\rho} \in \mathsf{FIN}$ with $[\![\hat{\rho}]\!] = \mathsf{C}([\![\rho]\!])$. As $\hat{\rho} \in \mathsf{FIN}$, the language $\mathcal{L}(\hat{\rho})$ is finite, which implies that $\mathcal{L}(\rho) = \Sigma^* \setminus \mathcal{L}(\hat{\rho})$ is co-finite. Hence, $\rho \in \mathsf{COF}$.

For the *only-if* direction, let $\rho \in \mathsf{COF}$; i. e., $\mathcal{L}(\rho)$ is co-finite. Hence, $\Sigma^* \setminus \mathcal{L}(\rho)$ is finite, and regular. Thus, there exists a proper regular expression $\hat{\rho}$ with $\mathcal{L}(\hat{\rho}) = \Sigma^* \setminus \mathcal{L}(\rho)$. As every proper regular expression is also a functional regex formula with no variables (and, hence, Boolean), $\hat{\rho} \in \mathsf{BCSR}$ follows. This gives $\hat{\rho} \in \mathsf{FIN}$, while $[\![\hat{\rho}]\!] = \mathsf{C}([\![\rho]\!])$ holds by our choice of $\hat{\rho}$. □ (Claim 3)

We now proceed to the main part of the proof, which uses these claims. Let $c$ be a complexity measure for the class $\mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$. Assume that there exists a recursive function $f\colon \mathbb{N} \to \mathbb{N}$ such that for all $\rho \in \mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$ for which $\mathsf{C}([\![\rho]\!])$ is a core spanner, there exists a $\hat{\rho} \in \mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$ with $[\![\hat{\rho}]\!] = \mathsf{C}([\![\rho]\!])$ and $c(\rho) \leq f(c(\hat{\rho}))$.

Our goal is to show that this assumption implies that $\mathsf{COF}$ is in $\Delta_2^0$. We prove this by defining a decision procedure with oracles for $\Sigma_1^0$ and $\Pi_2^0$ on the input $\rho \in \mathsf{BCSR}$ as follows. First, compute $n := f(c(\rho))$, and let

$$R_n := \{\hat{\rho} \in \mathsf{BCSR} \mid c(\hat{\rho}) \leq n\}.$$

From Claim 3, we know that $\rho \in \mathsf{COF}$ if and only if there is a $\hat{\rho} \in \mathsf{FIN}$ with $[\![\hat{\rho}]\!] = \mathsf{C}([\![\rho]\!])$. Due to our assumption on $f$, this holds if and only if such a $\hat{\rho}$ exists in $R_n$.

We now check for each $\hat{\rho} \in R_n$ whether it satisfies these two criteria:

1. $\hat{\rho} \in \mathsf{FIN}$
2. $[\![\hat{\rho}]\!] = \mathsf{C}([\![\rho]\!])$

Due to Claim 2, we know that $\mathsf{FIN}$ is in $\Sigma_1^0$. Hence, the first criterion can be decided with a $\Sigma_1^0$-oracle.

Regarding the second criterion, note that $[\![\hat{\rho}]\!] \neq \mathsf{C}([\![\rho]\!])$ is semi-decidable (as it suffices to find a $w \in \Sigma^*$ that disproves the equality). Hence, this criterion is co-semi-decidable, which means that it can be decided with a $\Pi_1^0$-oracle.

If there exists a $\hat{\rho} \in R_n$ that satisfies both criteria, the procedure returns `True`. In this case, $\rho \in \mathsf{COF}$ holds by Claim 3; hence, this is correct.

If no such $\hat{\rho}$ can be found among the (finitely many) elements of $R_n$, the procedure returns `False`. As mentioned above, this is correct due to our assumptions on $f$.

As $\mathsf{COF}$ can be decided by using oracles for $\Sigma_1^0$ and $\Pi_1^0$, we know that $\mathsf{COF} \in \Delta_2^0$ must hold. This contradicts Claim 1. As our only assumption was the existence of the recursive bound $f$, no such bound can exist.

In other words, there are core spanners where the (hierarchical) complement is also a core spanner, but the blowup between their representations is not bounded by any recursive function. Again, this holds for the other classes of representations as well.

This result has consequences to an open question of Fagin et al. One of the central tools in [7] is the core-simplification-lemma, which states that every core spanner is definable by an expression of the form $\pi_V S A$, where $A$ is a vset-automaton, $V \subseteq \mathsf{SVars}(A)$, and $S$ is a sequence of selections $\zeta^{=}_{x,y}$ for $x, y \in \mathsf{SVars}(A)$.

In addition to core spanners, Fagin et al. also discuss adding a set difference operator $\setminus$, and ask "whether we can find a simple form, in the spirit of the core-simplification lemma, when adding difference to the representation of core spanners". It is a direct consequence of Theorem 4.11 that such a simple representation, if it exists, cannot be obtained effectively, as reducing the number of difference operators can lead to a non-recursive blowup. While this observation does not prove that such a simple form does not exist, it suggests that any proof of its existence should be expected to be non-constructive.

## 5 Conclusions and Further Work

In Section 3, we have seen that core spanners can express languages that are defined by patterns or by vstar-free xregexes. We used this in Section 4 to derive various lower

bounds on decision problems, even for subclasses of core spanner representations. Note that in most of the cases, these lower bounds do not require the join operator, and mostly rely on the string equality selection. This can be interpreted as a sign that string equality (or repetition) is an expensive operator, in particular as similar results have been observed for related models (e. g., [2, 12, 16]). On the other hand, Proposition 4.2 demonstrates that even without string equality, join is also an expensive operator. The authors take this as a sign that the search for good restrictions on core spanners will probably have to combine restrictions on string equality and on join.

There is also reason to hope that the connections to patterns and word equations can be beneficial for spanners: There is recent work on restricted classes of pattern languages with an efficient membership problem (e. g., [10, 33]), which could lead to subclasses of spanners that can be evaluated more efficiently. Furthermore, as Theorems 3.12 and 3.13 show, core spanners and word equations with regular constraints are closely related. Recent work on word equations has also considered tasks like enumerating all solutions of an equation. The employed compression techniques (cf. [6]) might also be used to improve the evaluation of core spanners. In particular, the $\mathsf{EC}^{\mathsf{reg}}$-formulas that are constructed in the proof of Theorem 3.12 have the special property that there is a variable $x_w$ (for $w$), and for every solution $\sigma$ and every variable $x$, we have that $\sigma(x)$ is a subword of $\sigma(x_w)$.

Freydenberger [13] builds on this observation and introduces a fragment of $\mathsf{EC}^{\mathsf{reg}}$ that has exactly the same expressive power as core spanners. The connection is even stronger: As shown in [13], there exist polynomial time conversions between this fragment and core spanner representations. It remains to be seen whether the connection between spanners and word equations can also be used to find interesting subclasses of core spanners that have friendlier upper bounds (in particular regarding evaluation).

Also note that conversion from vstar-free regular expressions to core spanner representations that is used for Theorem 3.21 can lead to an exponential increase in size. As shown in [13], this blowup can be avoided by using a more involved construction.

Finally, while we only mentioned this explicitly in Section 4.2.1, note that most of the other results in this paper can also be directly converted to the appropriate spanner representations that use vset- and vstk-automata from [7].

**Acknowledgements** We thank Florin Manea for his suggestion to use word equations with regular constraints, and Thomas Zeume for reporting a list of typos. We also thank the anonymous reviewers of both this paper and the conference version for their feedback.



## References

1. Angluin, D.: Finding patterns common to a set of strings. J. Comput. Syst. Sci. **21**, 46–62 (1980)
2. Barceló, P., Libkin, L., Lin, A.W., Wood, P.T.: Expressive languages for path queries over graph-structured data. ACM Trans. Database Syst. **37**(4), 31 (2012)

3. Barceló, P., Muñoz, P.: Graph Logics with Rational relations: The Role of Word Combinatorics. In: Proc. CSL-LICS 2014 (2014)
4. Bremer, J., Freydenberger, D.D.: Inclusion problems for patterns with a bounded number of variables. Inform. Comput. **220–221**, 15–43 (2012)
5. Câmpeanu, C., Salomaa, K., Yu, S.: A formal study of practical regular expressions. Int. J. Found Comput. Sci. **14**, 1007–1018 (2003)
6. Diekert, V.: Makanin's Algorithm. In: Lothaire, M. (ed.) Algebraic Combinatorics on Words, chapter 12, pages 387–442. Cambridge University Press (2002)
7. Fagin, R., Kimelfeld, B., Reiss, F., Vansummeren, S.: Document spanners: A formal approach to information extraction. J. ACM **62**(2), 12 (2015)
8. Fagin, R., Kimelfeld, B., Reiss, F., Vansummeren, S.: Declarative cleaning of inconsistencies in information extraction. ACM Trans. Database Syst. **41**(1), 6 (2016)
9. Fernau, H., Manea, F., Mercas, R., Schmid, M.L.: Pattern Matching with variables: Fast Algorithms and New Hardness Results. In: Proc. STACS 2015 (2015)
10. Fernau, H., Schmid, M.L.: Pattern matching with variables: A multivariate complexity analysis. Inf. Comput. **242**, 287–305 (2015)
11. Fernau, H., Schmid, M.L., Villanger, Y.: On the parameterised complexity of string morphism problems. Theory Comput. Sys. (2015)
12. Freydenberger, D.D.: Extended regular expressions: Succinctness and decidability. Theory Comput. Sys. **53**(2), 159–193 (2013)
13. Freydenberger, D.D.: A Logic for Document Spanners. In: Proc ICDT (2017). Accepted
14. Freydenberger, D.D., Holldack, M.: Document spanners: From Expressive Power to Decision Problems. In: Proc. ICDT 2016, p. 2016
15. Freydenberger, D.D., Reidenbach, D.: Bad news on decision problems for patterns. Inform. Comput. **208**(1), 83–96 (2010)
16. Freydenberger, D.D., Schweikardt, N.: Expressiveness and static analysis of extended conjunctive regular path queries. J. Comput. Syst. Sci. **79**(6), 892–909 (2013)
17. Friedl, J.E.F.: Mastering Regular Expressions. O'Reilly Media. 3rd edition (2006)
18. Garey, M.R., Johnson, D.S.: Computers and intractability. W. H. Freeman and Company (1979)
19. Ginsburg, S., Spanier, E.: Semigroups, presburger formulas, and languages. Pac. J. Math. **16**(2), 285–296 (1966)
20. Grohe, M., Flum, J.: Parameterized complexity theory. Texts in Theoretical Computer Science. Springer (2006)
21. Hartmanis, J.: On gödel speed-up and succinctness of language representations. Theor. Comput. Sci. **26**(3), 335–342 (1983)
22. Holzer, M., Kutrib, M.: Descriptional complexity–an introductory survey. Sci. Appl. Language Methods **2**, 1–58 (2010)
23. Ibarra, O.H., Pong, T.-C., Sohn, S.M.: A note on parsing pattern languages. Pattern Recogn. Lett. **16**(2), 179–182 (1995)
24. Jiang, T., Kinber, E., Salomaa, A., Salomaa, K., Yu, S.: Pattern languages with and without erasing. Int. J Comput. Math. **50**, 147–163 (1994)
25. Jiang, T., Salomaa, A., Salomaa, K., Yu, S.: Decision problems for patterns. J. Comput. Syst Sci. **50**, 53–63 (1995)
26. Karhumàki, J., Mignosi, F., Plandowski, W.: The expressibility of languages and relations by word equations. J. ACM **47**(3), 483–505 (2000)
27. Kozen, D.: Lower Bounds for Natural Proof Systems. In: Proc. FOCS 1977, p. 1977
28. Kozen, D.: Theory of computation. Springer-Verlag (2006)
29. Kutrib, M.: The phenomenon of non-recursive trade-offs. Int. J. Found. Comput. Sci. **16**(5), 957–973 (2005)
30. Lothaire, M.: Combinatorics on Words. Cambridge University Press (1997)
31. Ohlebusch, E., Ukkonen, E.: On the equivalence problem for E-pattern languages. Theor. Comput. Sci. **186**, 231–248 (1997)
32. Parikh, R.J.: On context-free languages. J. ACM **13**(4), 570–581 (1966)
33. Reidenbach, D., Schmid, M.L.: Patterns with bounded treewidth. Inform. Comput. **239**, 87–99 (2014)
34. Stephan, F., Yoshinaka, R., Zeugmann, T.: On the Parameterised Complexity of Learning Patterns. In: Proc. ISCIS 2011, p. 2011