Document Spanners: From Expressive Power to Decision Problems

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

1 **University of Bayreuth, Bayreuth, Germany**
2 **Goethe University, Frankfurt, Germany**

Abstract

We examine *document spanners*, a formal framework for information extraction that was introduced by Fagin et al. (PODS 2013). 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.

1998 ACM Subject Classification H.2.1 Data models, H.2.4 Textual databases, Relational databases, Rule-based databases, F.4.3 Classes defined by grammars or automata, Decision problems, F.1.1 Relations between models

Keywords and phrases Information extraction, document spanners, regular expressions, regex, patterns, word equations, decision problems, descriptional complexity

Digital Object Identifier 10.4230/LIPIcs.ICDT.2016.1

1 Introduction

Information Extraction (IE) is the task of automatically extracting structured information from texts. This paper examines *document 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 [6].

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*. 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 α matches a string w, each variable is mapped to the span in w that matches that subexpression. Hence, each match of α 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.

* Supported by Deutsche Forschungsgemeinschaft (DFG) under grant FR 3551/1-1.

19th International Conference on Database Theory (ICDT 2016).
Editors: Wim Martens and Thomas Zeume; pp. 1–33

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

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.

The first of these models are *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 $\alpha = xx\mathtt{ab}y$ (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. Jiang et al. [19]), and their inclusion problem is undecidable (cf. Bremer and Freydenberger [3]). As we show that core spanners can recognize pattern languages, this allows us to conclude that evaluation of core spanners is NP-hard, and that spanner containment is undecidable.

The second model we consider are *word equations*, which are equations of the form $\alpha = \beta$, where α and β are patterns, which can be used to define 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 [5]), 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.

The third model are *regexes* (also called *extended regular expressions* 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 [14]) and that allows the definition of non-regular languages. For example, the regex $x\{\Sigma^*\}\&x\&x$ generates all words www with $w \in \Sigma^*$, 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 regexes than for their "classical" counterparts. In particular, various problems of static analysis are undecidable (Freydenberger [11]).

But as shown by Fagin et al. [7], document spanners cannot define all languages that are definable by regexes. Intuitively, the reason for this is that regexes can use their repetition operators inside a Kleene star, which allows them to repeat an arbitrary word an unbounded number of times, while core spanners have to express repetitions with variables and string equality selections. Inspired by this observation, we introduce *variable-star free (or vstar-free) regexes* as those regexes that neither define nor use variables inside a Kleene star. We show that every vstar-free regex can be converted into an equivalent core spanner. Since all undecidability results by Freydenberger [11] also apply to vstar-free regexes, these undecidability results carry over to core spanners. This also has various consequences to the minimization and the relative succinctness of classes of spanner representations, and to the simplification of core spanners with difference operators. 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 regexes, 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. [1] 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 [13], ECRPQs have undecidability results that are comparable to those in the present paper, and to those for regexes (cf. Freydenberger [11]). Furthermore, Barceló and Muñoz [2] have used word equations with regular constraints for variants of CRPQs.

Structure of the paper. In Section 2, we give definitions of regexes 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. Due to space reasons, all proofs were moved to an appendix that is contained in the full version of the paper.

2 Preliminaries

Let $\mathbb{N}$ and $\mathbb{N}_{>0}$ be the sets of non-negative and positive integers, respectively. Let Σ be a fixed finite alphabet of *(terminal) symbols*. Except when stated otherwise, we assume $|\Sigma| \geq 2$. We use ε 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 Σ)* is a subset of $(\Sigma^*)^n$.

2.1 Regexes (Extended Regular Expressions)

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

▶ **Definition 1.** We fix an infinite set X of *variables* and define the set M of *meta symbols* as $M := \{\varepsilon, \emptyset, (,), \{, \}, \cdot, \vee, ^*, \&\}$. Let Σ, X, and M be pairwise disjoint. The set of *regexes (extended regular expressions)* is defined as follows:

1. The symbols $\emptyset$, ε, and every $a \in \Sigma$ are regexes.
2. If α_1 and α_2 are regex, then $(\alpha_1 \cdot \alpha_2)$ *(concatenation)*, $(\alpha_1 \vee \alpha_2)$ *(disjunction)*, and (α_1^*) *(Kleene star)* are regexes.
3. For every $x \in X$ and every regex α that contains neither $x\{\cdots\}$ nor $\&x$ as a subword, $x\{\alpha\}$ is a regex *(variable binding)*.
4. For every $x \in X$, $\&x$ is a regex *(variable reference)*.

If a subword β of a regex α is a regex itself, we call β a *subexpression (of α)*. The set of all subexpressions of α is denoted by $\mathsf{Sub}(\alpha)$, and the set of variables occurring in variable bindings in a regex α is denoted by $\mathsf{Vars}(\alpha)$. If a regex α contains neither variable references, nor variable bindings, we call α 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 α^+ 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 Σ as a shorthand for the regular expression $\bigvee_{a \in \Sigma} a$.

Before introducing the semantics of regexes formally, we give an intuitive explanation. An expression of the form $\alpha = x\{\beta\}$ matches the same strings as β, but α 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, α defines the (non-regular) *copy-language* $\{ww \mid w \in \Sigma^*\}$.

The following definition of the semantics of regexes is based on the semantics by Freydenberger [11], which is an adaption of the semantics from Câmpeanu et al. [4] (the former uses variables, the latter backreferences). In comparison to [11], the case for Kleene star has been changed, in order to make the definition compatible with the parse trees from Fagin et al. [7].

▶ **Definition 2.** Let γ be a regex over Σ and X. A *γ-parse tree* is a finite, directed, and ordered tree T_γ. Its nodes are labeled with tuples of the form $(w, \gamma') \in (\Sigma^* \times \mathsf{Sub}(\gamma))$. The root of every γ-parse tree T_γ is labeled with (w, γ), $w \in \Sigma^*$; and the following rules must hold for each node v of T_γ:

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, β_1) and (w_2, β_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, β_1) or (w, β_2).
4) If v is labeled (w, β^*), 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, β), …, (w_k, β).
5) If v is labeled $(w, x\{\beta\})$, then v has a single child, labeled (w, β).
6) If v is labeled $(w, \&x)$, let $\prec$ denote the post-order of the nodes of T_γ (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 γ-parse tree T_γ is labeled (w, γ), we call T_γ a *γ-parse tree for w*. If the context is clear, we omit γ and call T_γ 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 ε. For an example of a parse tree, see Figure 1.

We use parse trees to define the semantics of regexes:

▶ **Definition 3.** A regex γ recognizes the language $\mathcal{L}(\gamma)$ of all $w \in \Sigma^*$ for which there exists a γ-parse tree T_γ with (w, γ) as root label.

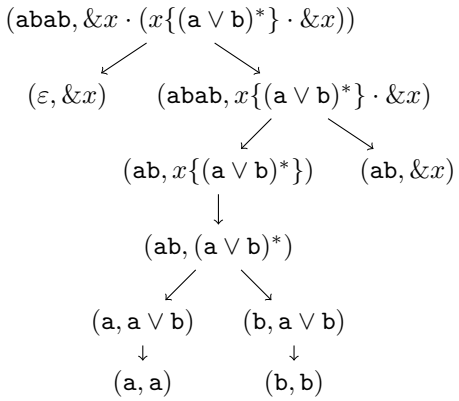

Figure 1 The α-parse tree for w, where $\alpha := \&x \cdot (x\{(\mathtt{a} \vee \mathtt{b})^*\} \cdot \&x)$ and $w := \mathtt{abab}$.

▶ **Example 4.** Let $\alpha := x\{\Sigma^+\} \cdot (\&x)^+$. Then $\mathcal{L}(\alpha) = \{w^n \mid w \in \Sigma^+, n \geq 2\}$. Furthermore, let $\beta := x\{\Sigma^+\} \cdot \&x \cdot x\{\Sigma^+\} \cdot \&x$. Then $\mathcal{L}(\beta) = \{x_1x_1x_2x_2 \mid x_1, x_2 \in \Sigma^+\}$. Finally, for some $a \in \Sigma$, let $\gamma := x\{aa^+\} \cdot (\&x)^+$. Then $\mathcal{L}(\gamma) = \{a^n \mid n \geq 2, n \text{ is not prime}\}$.

2.2 Document Spanners

Let $w := a_1a_2 \cdots a_n$ be a word over Σ, 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 5.** Let $w := \mathtt{aabbcabaa}$. As $|w| = 9$, both $[3, 3\rangle$ and $[5, 5\rangle$ are spans of w, but $[10, 11\rangle$ is not. As $3 \neq 5$, the first two spans are not equal, even though $w_{[3,3\rangle} = w_{[5,5\rangle} = \varepsilon$. The whole word w is described by the span $[1, 10\rangle$.

▶ **Definition 6.** Let SVars be a fixed, infinite set of *span variables*, where Σ and SVars are disjoint. Let $V \subset \mathsf{SVars}$ be a finite subset of 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 spanners, which map words w to (V, w)-relations:

▶ **Definition 7.** Let V and Σ 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 $P(w) = \mathtt{True}$ and $P(w) = \mathtt{False}$ instead of $P(w) = \{()\}$ and $P(w) = \emptyset$, respectively. Let P be a spanner and $w \in \Sigma^*$. 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 Υ_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 8.** A *regex formula* is a regex α over Σ and $X := \mathsf{SVars}$ such that α does not contain any variable references, and for every $\beta \in \mathsf{Sub}(\alpha)$ with $\beta = \gamma^*$, no subexpression of γ 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 γ.

To define the semantics of regex formulas, we use the definition of parse trees for regexes, see Definition 2. Intuitively, the goal of this definition is that, each occurrence of a variable x in a γ-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 9.** Let γ be a regex formula. We call γ *functional* if for every $w \in \Sigma^*$ and every γ-parse tree T_γ for w, each variable in $\mathsf{SVars}(\gamma)$ occurs in exactly one node label of T_γ. The class of all functional regex formulas is denoted by 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 contain exactly the same variables. Consider the following example:

▶ **Example 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 γ_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 γ_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 11.** Let γ be a functional regex formula and let T be a γ-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 γ is functional, for every $x \in \mathsf{SVars}(\gamma)$, exactly one node v_x of T has a label that contains x. We define $\mu^T \colon \mathsf{SVars}(\gamma) \to \mathsf{Spans}(w)$ 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 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 the tuples $([2,4\rangle, [3,3\rangle, [3,4\rangle)$, $([2,5\rangle, [3,4\rangle, [4,5\rangle)$, $([2,6\rangle, [3,5\rangle, [5,6\rangle)$, $([3,5\rangle, [4,4\rangle, [4,5\rangle)$, $([3,6\rangle, [4,5\rangle, [5,6\rangle)$.

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 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 $\mathsf{SVars}(P_1 \cup P_2) := \mathsf{SVars}(P_1) \cup \mathsf{SVars}(P_2)$ 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 $\pi_Y P(w) := P|_Y(w)$ for all $w \in \Sigma^*$, where $P|_Y(w)$ 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 $w \in \Sigma^*$, $(P_1 \bowtie P_2)(w)$ is the set of all $(V_1 \cup V_2, w)$-tuples μ for which there exist (V_i, w)-tuples μ_i $(i \in \{1, 2\})$ with $\mu|_{V_1}(w) = \mu_1(w)$ and $\mu|_{V_2}(w) = \mu_2(w)$.

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

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.

Regarding the join of two spanners P_1 and P_2, $P_1 \bowtie P_2$ 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 14.** Let $P_1 := \zeta^=_{x,y}[\![x\{\Sigma^*\} \cdot y\{\Sigma^*\}]\!]$ and $P_2 := \zeta^=_{x,y,z}[\![x\{\Sigma^*\} \cdot y\{\Sigma^*\} \cdot z\{\Sigma^*\}]\!]$. Then $\mathcal{L}(P_1) = \{ww \mid w \in \Sigma^*\}$, and the variables x and y always refer to the span of the first and second occurrence of w, respectively. Analogously, $\mathcal{L}(P_2) = \{www \mid w \in \Sigma^*\}$ (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 to the set of common variables; i. e., $\pi_{\{x,y\}} P_2$.

▶ **Definition 15.** A *spanner algebra* is a finite set of spanner operators. If 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 O with regex formulas from RGX. For each spanner representation of the form $o\rho$ (or $\rho_1\, o\, \rho_2$), where $o \in \mathsf{O}$, we define $[\![o\rho]\!] = o[\![\rho]\!]$ (and $[\![\rho_1\, o\, \rho_2]\!] = [\![\rho_1]\!]\, o\, [\![\rho_2]\!]$). Furthermore, $[\![\mathsf{RGX}^{\mathsf{O}}]\!]$ is the closure of $[\![\mathsf{RGX}]\!]$ under the spanner operators in O.

We define $\mathcal{L}(\rho) := \mathcal{L}(\llbracket\rho\rrbracket)$ for every spanner representation ρ. Fagin et al. [7] refer to $\llbracket\mathsf{RGX}\rrbracket$ as the class of *hierarchical regular spanners* and to $\llbracket\mathsf{RGX}^{\{\pi,\cup,\bowtie\}}\rrbracket$ 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 $\llbracket\mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}\rrbracket$. 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 16.** Let X be an infinite variable alphabet that is disjoint from Σ. 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 α by $\mathsf{Vars}(\alpha)$.

Intuitively, a pattern α generates exactly those words that can be obtained by replacing the variables in α 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. [19]).

▶ **Example 17.** 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 α, we can straightforwardly construct a regex for $\mathcal{L}(\alpha)$. A similar observation holds for core spanners:

▶ **Theorem 18.** *Given a pattern α, we can compute in polynomial time a $\rho_\alpha \in \mathsf{RGX}^{\{\zeta^=\}}$ such that $\mathcal{L}(\rho_\alpha) = \mathcal{L}(\alpha)$.*

▶ **Example 19.** 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 18 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 18 is so easy 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 18, 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 16) and can be used to define languages and relations, cf. Karhumäki et al. [21]:

▶ **Definition 20.** A *word equation* is a pair $\eta = (\eta_L, \eta_R)$ of patterns η_L and η_R. A pattern substitution σ is a *solution* of η 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 (η_L, η_R) as $\eta_L = \eta_R$. The following relations are well known examples of relations that are definable by word equations:

▶ **Definition 21.** Over Σ^*, we define relations $R_{\text{com}} := \{(x, y) \mid x, y \in \{u\}^* \text{ for some } u \in \Sigma^*\}$ and $R_{\text{cyc}} := \{(x, y) \mid x \text{ is a cyclic permutation of } y\}$.

As shown in Lothaire [25], the relation R_{com} is defined by the equation $xy = yx$, and R_{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 $\vec{x} = (x_1, \dots, x_k)$ with $x_1, \dots, x_k \in \mathsf{SVars}(P)$, the spanner $\zeta^R_{\vec{x}} P$ is also in C.

▶ **Proposition 22.** *The relations R_{com} and R_{cyc} are selectable by core spanners.*

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

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

Furthermore, the spanner that is constructed for R_{com} in the proof of Proposition 22 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 constrictions. We shall see in Theorem 28 further down that, if this restriction 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. [21]). In order to overcome this problem, we consider the following extension:

▶ **Definition 24.** Let $\eta = (\eta_L, \eta_R)$ be a word equation. A *regular constraints function*[1] is a function Cstr that maps each $x \in \mathsf{Vars}(\eta)$ to a regular language $\mathsf{Cstr}(x)$, where each of these languages is defined by a nondeterministic finite automaton. A solution σ of η is a *solution of η under constraints* Cstr if $\sigma(x) \in \mathsf{Cstr}(x)$ holds for every $x \in \mathsf{Vars}(\eta)$.

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

A syntactic extension of word equations are *existential concatenation formulas*, which are obtained by extending word equations with $\vee$, $\wedge$, and existential quantification over variables. For example, R_{cyc} is expressed by the formula $\varphi_{\text{cyc}}(x, y) := \exists z\colon (xz = zy)$. Using appropriate coding techniques, one can transform every existential concatenation formula into an equivalent word equation (see Diekert [5]). In particular, this transformation is possible in polynomial time.

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 σ if $\sigma(x) \in \mathcal{L}(A)$. We call the resulting formulas *existential concatenation formulas with regular constraints*, or $\mathsf{EC}^{\mathsf{reg}}$*-formulas*.

[1] While most existing literature uses the term *rational constraints*, we follow the terminology of [2].

▶ **Example 25.** 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 φ expresses the relation of all (x,y) that have a common subword z from $\mathcal{L}(A)$.

Using the same techniques as for formulas without constraints, $\mathsf{EC}^{\mathsf{reg}}$-formulas can be transformed into equivalent word equations with regular constraints, and this construction is possible in polynomial time as well (cf. Diekert [5]). In order to express core spanners with $\mathsf{EC}^{\mathsf{reg}}$-formulas, we introduce the following definition:

▶ **Definition 26.** 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 φ *realizes* P if, for all $w, w_1^P, w_1^C, \ldots, w_n^P, w_n^C \in \Sigma^*$, $\varphi(w, w_1^P, w_1^C, \ldots, w_n^P, w_n^C) = \mathtt{True}$ 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 27.** *Given $\rho \in \mathsf{RGX}^{\{\pi, \zeta^=, \cup, \bowtie\}}$ with $\mathsf{SVars}(\rho) = \{x_1, \ldots, x_n\}$, $n \geq 0$, we can compute in polynomial time 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 $[\![\rho]\!]$.*

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 28.** *Every word equation $\eta = (\eta_L, \eta_R)$ with regular constraints Cstr can be converted computably into a $\rho \in \mathsf{RGX}^{\{\zeta^=, \bowtie\}}$ with $\mathsf{SVars}(\rho) \subseteq \mathsf{Vars}(\eta)$ such that for all $w \in \Sigma^*$, there is a solution σ of η under constraints Cstr 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 $x \in \mathsf{Vars}(\eta)$.*

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; and computed (V,w)-relation encodes solutions of the equation.

▶ **Example 29.** Let $\mathtt{a}, \mathtt{b} \in \Sigma$ and define $\eta := (xy, yx)$ with $\mathsf{Cstr}(x) = \mathcal{L}(\mathtt{aab}^+)$, $\mathsf{Cstr}(y) = \Sigma^+$. The construction from the proof of Theorem 28 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 ρ; the only difference that the solution is encoded in $w = \sigma(\eta_L \cdot \eta_R)$, instead of $\sigma(\eta_L)$.

3.3 Regexes

As shown by Fagin et al. [7], there are languages that are recognized by regexes, but not by core spanners. In order to prove this, [7] introduced the so-called "uniform-0-chunk"-language

L_{uzc}: Assuming $0, 1 \in \Sigma$, L_{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 regex $\alpha_{\text{uzc}} := 1^+ \cdot x\{0^*\} \cdot (1^+ \cdot \&x)^* \cdot 1^+$, but no core spanner recognizes L_{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 α_{uzc} uses variable references inside of a Kleene plus. This raises the question whether regexes 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 regexes:

▶ **Definition 30.** A regex α is *variable star-free* (short: *vstar-free*) if, for every $\beta \in \mathsf{Sub}(\alpha)$ with $\beta = \gamma^*$, no subexpression of γ is a variable binding or a variable reference. We denote the class of all vstar-free regexes by vsfRX.

As we shall see in Theorem 36 below, every language that is recognized by a vstar-free regex 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 18 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 regexes:

▶ **Definition 31.** An $\alpha \in \mathsf{vsfRX}$ is a *regex path* if, for every $\beta \in \mathsf{Sub}(\alpha)$ with $\beta = (\gamma_1 \vee \gamma_2)$, no subexpression of γ_1 or γ_2 is a variable binding or a variable reference. We denote the class of all regex paths by RXP.

Intuitively, a regex path $\alpha \in \mathsf{RXP}$ can be understood as a concatenation $\alpha = \alpha_1 \cdots \alpha_n$, where each α_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 a regex path. By "multiplying out" disjunctions that contain variables, we can convert every vstar-free regex into a disjunction of regex paths.

▶ **Lemma 32.** *Given $\alpha \in \mathsf{vsfRX}$, we can compute $\alpha_1, \ldots, \alpha_n \in \mathsf{RXP}$ with $\mathcal{L}(\alpha) = \bigcup_{i=1}^{n} \mathcal{L}(\alpha_i)$.*

▶ **Example 33.** Let $\alpha := x\{\Sigma^*\} \cdot (\&x \vee y\{\Sigma^*\}) \cdot (\&x \vee \&y)$. Multiplying out the disjunctions, we obtain regex paths $\alpha_1 = x\{\Sigma^*\} \cdot \&x \cdot \&x$, $\alpha_2 = x\{\Sigma^*\} \cdot y\{\Sigma^*\} \cdot \&x$, $\alpha_3 = x\{\Sigma^*\} \cdot \&x \cdot \&y$, and $\alpha_4 = x\{\Sigma^*\} \cdot y\{\Sigma^*\} \cdot \&y$. Then $\mathcal{L}(\alpha) = \bigcup_{i=1}^{4} \mathcal{L}(\alpha_i)$.

This transformation process might result in an exponential number of regex paths; but as efficiency is not of concern right now, this is not a problem. Each of these regex paths is then transformed into a functional regex formula:

▶ **Lemma 34.** *Given $\alpha \in \mathsf{RXP}$, we can be compute a $\rho \in \mathsf{RGX}^{\{\pi, \zeta^=\}}$ with $\mathcal{L}(\rho) = \mathcal{L}(\alpha)$.*

▶ **Example 35.** Consider the regex 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 34 leads to the equivalent regex 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

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

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 32 and Lemma 34 to observe the following.

▶ **Theorem 36.** *Given $\alpha \in \mathsf{vsfRX}$, we can compute a $\rho \in \mathsf{RGX}^{\{\pi,\zeta^{=},\cup\}}$ with $\mathcal{L}(\rho) = \mathcal{L}(\alpha)$.*

In Section 4.2, we use Theorem 36 together with the undecidability results from [11] to obtain multiple lower bounds for static analysis problems. Theorem 36 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 23, 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_{imp} might seem to be an obvious witness that separates the classes of languages that are recognized by core spanners and by vstar-free regexes, proving this appears to be quite involved. Instead, we consider a related language, which allows us to use the following tool:

▶ **Definition 37.** Let $k \in \mathbb{N}_{>0}$. We call a set $A \subseteq \mathbb{N}^k$ *linear* if there exist an $r \geq 0$ and $m_0, \dots, m_r \in \mathbb{N}^k$ with $A = \{m_0 + m_1 i_1 + m_2 i_2 + \cdots + m_r i_r \mid i_1, i_2, \dots, i_r \in \mathbb{N}\}$. A set $A \subseteq \mathbb{N}^k$ is *semi-linear* if it is a finite union of linear sets. Assume Σ is ordered; i. e., $\Sigma = \{a_1, a_2 \dots, a_k\}$. The *Parikh map* $\Psi\colon \Sigma^* \to \mathbb{N}^k$ is defined by $\Psi(w) := (|w|_{a_1}, |w|_{a_2}, \dots, |w|_{a_k})$, 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 [27], every context-free language is semi-linear. Moreover, as shown by Ginsburg and Spanier [15], a set is semi-linear if and only if it is definable in Presburger arithmetic. Building on this, we state the following:

▶ **Theorem 38.** *For every $\alpha \in \mathsf{vsfRX}$, the language $\mathcal{L}(\alpha)$ is semi-linear.*

We use Theorem 38 to separate the classes of languages that are recognized by core spanners and by vstar-free regexes:

▶ **Lemma 39.** *Let $L_{\text{nsl}} := \{(\mathtt{ab}^m)^n \mid m, n \geq 2\}$ and $\rho := \zeta^{R_{\text{com}}}_{x,y}(x\{\mathtt{abb}^+\}y\{\Sigma^+\})$ for $\Sigma := \{\mathtt{a}, \mathtt{b}\}$. Then $L_{\text{nsl}} = \mathcal{L}(\rho)$, but there is no $\alpha \in \mathsf{vsfRX}$ 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 ρ from Example 29 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 39 does not resolve the open question from [7] whether there is a language that is recognized by a core spanner, but not by a regex, as Theorem 38 only applies to vstar-free regexes. We have already seen languages that are not semi-linear, but are recognized by regexes: The language L_{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 4):

▶ **Example 40.** Let $\Sigma := \{\mathtt{a}\}$, and define the language $L_{\text{npr}} := \{\mathtt{a}^{mn} \mid m, n \geq 2\}$. In other words, L_{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_{nsl} and L_{npr} are defined by very similar regexes, the latter cannot be recognized by core spanners. In order to show this with a semi-linearity argument, we observe:

▶ **Theorem 41.** *Let $|\Sigma| = 1$ and let P be a core spanner over Σ. Then $\mathcal{L}(P)$ is semi-linear.*

Apart from the observation that L_{npr} from Example 40 is not recognized by core spanners, Theorem 41 also allows us to conclude that on unary alphabets, core spanners recognize exactly the class of regular languages (which, on unary alphabets, is identical to the class of context-free languages).

4 Decision Problems

4.1 Spanner Evaluation

We first examine the *combined complexity* of the evaluation problem for core spanners, the problem CSp-Eval: Given a $\rho \in \mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$, a $w \in \Sigma^*$, and a $(\mathsf{SVars}(\rho), w)$-tuple μ, 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 α and a word w, decide whether $w \in \mathcal{L}(\alpha)$. As shown by Jiang et al. [19], this problem is NP-complete. Due to Theorem 18, we observe the following (the proof of NP-membership is straightforward).

▶ **Theorem 42.** *CSp-Eval is* NP*-complete, even if restricted to* $\mathsf{RGX}^{\{\pi,\zeta^=\}}$.

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 can not be allowed to recognize the full class of pattern languages.

For pattern languages, it was shown by Ibarra et al. [18] 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, this is usually not considered satisfactory. In fact, it was first observed by Stephan et al. [29] 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. 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 [9] and Fernau et al. [10] discuss 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 [28] and Fernau et al. [8] 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.

We also consider the *data complexity* of the evaluation problem for core spanners. For every core spanner representation ρ over Σ, we define the decision problem CSp-Eval(ρ): Given a $w \in \Sigma^*$ and a w-tuple μ, is $\mu \in [\![\rho]\!](w)$? Using a slight variation of the proof of Theorem 42, we obtain the following.

▶ **Theorem 43.** *For every* $\rho \in \mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$, *CSp-Eval($\rho$) is in* NL.

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 27 to convert core spanner representations to $\mathsf{EC}^{\mathsf{reg}}$-formulas, for which satisfiability is in PSPACE (cf. Diekert [5]). Hence, we observe:

▶ **Theorem 44.** *CSp-Sat is* PSPACE*-complete, even if restricted to* $\mathsf{RGX}^{\{\zeta^=\}}$.

For the lower bound, the proof of Theorem 44 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 Σ^*, it follows from Theorem 28 that this problem would still be at least as hard as the satisfiability problem for word equations without constraints (cf. Diekert [5]).

Furthermore, it is 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 45.** *CSp-Hierarchicality is* PSPACE*-complete, even if restricted to* $\mathsf{RGX}^{\{\zeta^=,\bowtie\}}$.

For the remaining problems, we use Theorem 36, and the fact that the undecidability results from Freydenberger [11] also hold for vstar-free regexes:

▶ **Theorem 46.** *CSp-Universality and CSp-Equivalence are not semi-decidable, and CSp-Regularity is neither semi-decidable, nor co-semi-decidable. This holds even if the input is restricted to* $\mathsf{RGX}^{\{\pi,\zeta^=,\cup\}}$.

As the proof of Theorem 46 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 $\Upsilon^{\mathbf{H}}_{\mathsf{SVars}(\rho)}$.

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 regexes $\alpha_{\mathcal{X}}$ that use only a single variable x, the resulting core spanners use an unbounded amount variables, as every occurrence of a variable reference $\&x$ in a regex 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 [11]). Theorem 46 also implies that CSp-Containment is not semi-decidable. This holds even for a more restricted class of spanners:

▶ **Theorem 47.** *CSp-Containment is not semi-decidable, even if restricted to* $\mathsf{RGX}^{\{\pi,\zeta^=\}}$.

As shown by Bremer and Freydenberger [3], 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 [26], Freydenberger and Reidenbach [12]). Undecidability of this problem would imply undecidability of CSp-Equivalence, even if restricted to representations from $\mathsf{RGX}^{\{\pi,\zeta^=\}}$.

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 [17], 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 further down. Hence, instead of defining a single fixed notion of size, we use the following general definition of complexity measures from Kutrib [24]:

▶ **Definition 48.** Let SR be a class of spanner representations. A *complexity measure* for SR is a recursive function $c : \mathsf{SR} \to \mathbb{N}$ such that for each Σ, the set of all $\rho \in \mathsf{SR}$ that represent spanners over Σ can be computably 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 48 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 SR. Regardless of the chosen complexity measure, computable minimization of core spanners is impossible:

▶ **Theorem 49.** *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.*

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 $\mathsf{VA}_{\mathsf{stk}}$) and extended with the same spanner operators; and they compare the expressive power of these spanner representations to $\mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$ and its subclasses. Without going into details, we note that their equivalence proofs use computable conversions between the models. Hence, Theorem 49 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 [16], we can use the fact that $\mathsf{CSp\text{-}Regularity}$ is not co-semi-decidable to compare the relative succinctness of regular and core spanner representations:

▶ **Theorem 50.** *Let c_1, c_2 be complexity measures for $\mathsf{RGX}^{\{\pi,\cup,\bowtie\}}$ 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 $[\![\hat{\rho}]\!] = [\![\rho]\!]$.*

Hence, the blowup from $\mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$ to $\mathsf{RGX}^{\{\pi,\cup,\bowtie\}}$ is not bounded by a 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* $\mathsf{C}(P) := \Upsilon_{\mathsf{SVars}(P)} \setminus P$, and its *hierarchical complement* $\mathsf{C}^{\mathbf{H}}(P) := \Upsilon^{\mathbf{H}}_{\mathsf{SVars}(P)} \setminus P$.

▶ **Theorem 51.** *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 $\mathsf{C}([\![\rho]\!]) \in [\![\mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}]\!]$, but $c(\rho) > f(c(\hat{\rho}))$ holds for every $\hat{\rho} \in \mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$ with $[\![\hat{\rho}]\!] = \mathsf{C}([\![\rho]\!])$. This also holds if we consider $\mathsf{C}^{\mathbf{H}}$ instead of C.*

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 51 that such a simple representation, if it exists, cannot be obtained computably, 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 regexes. 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., [1, 11, 13]).

On a more positive note, there is reason to hope that these connections can be beneficial for spanners: There is recent work on restricted classes of pattern languages with an efficient membership problem (e. g., [9, 28]), which could lead to subclasses of spanners that can be evaluated more efficiently. Furthermore, as Theorems 27 and 28 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. [5]) 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 27 have the special property that there is a variable x_w (for w), and for every solution σ and every variable x, $\sigma(x)$ is a subword of $\sigma(x_w)$. It remains to be seen whether this restriction leads to favorable lower bounds.

Also note that conversion from vstar-free regular expressions to core spanner representations that is used for Theorem 36 can lead to an exponential increase in size. If this size blowup cannot be avoided, allowing vstar-free regexes as primitive spanner representations might be useful as syntactic sugar.

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 for their feedback.

References

1 P. Barceló, L. Libkin, A. W. Lin, and P. T. Wood. Expressive languages for path queries over graph-structured data. *ACM T. Database Syst.*, 37(4):31, 2012.

2 P. Barceló and P. Muñoz. Graph logics with rational relations: the role of word combinatorics. In *Proc. CSL-LICS 2014*, 2014.

3 J. Bremer and D. D. Freydenberger. Inclusion problems for patterns with a bounded number of variables. *Inform. Comput.*, 220–221:15–43, 2012.

4 C. Câmpeanu, K. Salomaa, and S. Yu. A formal study of practical regular expressions. *Int. J. Found. Comput. Sci.*, 14:1007–1018, 2003.

5 V. Diekert. Makanin's Algorithm. In M. Lothaire, editor, *Algebraic Combinatorics on Words*, chapter 12, pages 387–442. Cambridge University Press, 2002.

6 R. Fagin, B. Kimelfeld, F. Reiss, and S. Vansummeren. Cleaning inconsistencies in information extraction via prioritized repairs. In *Proc. PODS 2014*, 2014.

7 R. Fagin, B. Kimelfeld, F. Reiss, and S. Vansummeren. Document spanners: A formal approach to information extraction. *J. ACM*, 62(2):12, 2015.

8 H. Fernau, F. Manea, R. Mercas, and M. L. Schmid. Pattern matching with variables: Fast algorithms and new hardness results. In *Proc. STACS 2015*, 2015.

9 H. Fernau and M. L. Schmid. Pattern matching with variables: A multivariate complexity analysis. *Inf. Comput.*, 242:287–305, 2015.

10 H. Fernau, M. L. Schmid, and Y. Villanger. On the parameterised complexity of string morphism problems. *Theory Comput. Sys.*, 2015.

11 D. D. Freydenberger. Extended regular expressions: Succinctness and decidability. *Theory Comput. Sys.*, 53(2):159–193, 2013.

12 D. D. Freydenberger and D. Reidenbach. Bad news on decision problems for patterns. *Inform. Comput.*, 208(1):83–96, 2010.

13 D. D. Freydenberger and N. Schweikardt. Expressiveness and static analysis of extended conjunctive regular path queries. *J. Comput. Syst. Sci.*, 79(6):892–909, 2013.

14 J. E. F. Friedl. *Mastering Regular Expressions*. O'Reilly Media, 3rd edition, 2006.

15 S. Ginsburg and E. Spanier. Semigroups, presburger formulas, and languages. *Pac. J. Math.*, 16(2):285–296, 1966.

16 J. Hartmanis. On Gödel speed-up and succinctness of language representations. *Theor. Comput. Sci.*, 26(3):335–342, 1983.

17 M. Holzer and M. Kutrib. Descriptional complexity–an introductory survey. *Scientific Applications of Language Methods*, 2:1–58, 2010.

18 O. H. Ibarra, T.-C. Pong, and S. M. Sohn. A note on parsing pattern languages. *Pattern Recogn. Lett.*, 16(2):179–182, 1995.

19 T. Jiang, E. Kinber, A. Salomaa, K. Salomaa, and S. Yu. Pattern languages with and without erasing. *Int. J. Comput. Math.*, 50:147–163, 1994.

20 T. Jiang, A. Salomaa, K. Salomaa, and S. Yu. Decision problems for patterns. *J. Comput. Syst. Sci.*, 50:53–63, 1995.

21 J. Karhumäki, F. Mignosi, and W. Plandowski. The expressibility of languages and relations by word equations. *J. ACM*, 47(3):483–505, 2000.

22 D. Kozen. Lower bounds for natural proof systems. In *Proc. FOCS 1977*, 1977.

23 D. Kozen. *Theory of Computation*. Springer-Verlag, 2006.

24 M. Kutrib. The phenomenon of non-recursive trade-offs. *Int. J. Found. Comput. Sci.*, 16(5):957–973, 2005.

25 M. Lothaire. *Combinatorics on Words*. Cambridge University Press, 1997.

26 E. Ohlebusch and E. Ukkonen. On the equivalence problem for E-pattern languages. *Theor. Comput. Sci.*, 186:231–248, 1997.

27 R. J. Parikh. On context-free languages. *J. ACM*, 13(4):570–581, 1966.

28 D. Reidenbach and M. L. Schmid. Patterns with bounded treewidth. *Inform. Comput.*, 239:87–99, 2014.

29 F. Stephan, R. Yoshinaka, and T. Zeugmann. On the parameterised complexity of learning patterns. In *Proc. ISCIS 2011*, 2011.

A Results from Section 3

A.1 Proof of Theorem 18

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 construct a regex formula $\hat{\alpha} := \hat{\alpha}_1 \cdots \hat{\alpha}_n$, where for each $i \in \{1, \ldots, n\}$, $\hat{\alpha}_i$ is defined as follows:

1. If α_i is a terminal (i. e., there is an $a \in \Sigma$ with $\alpha_i = a$), let $\hat{\alpha}_i := a$.
2. If α_i is a variable (i. e., there is an $x \in X$ with $\alpha_i = x$), let $j := |\alpha_1 \cdots \alpha_i|_x$, and define $\hat{\alpha}_i := x_j\{\Sigma^*\}$ (with $x_j \in \mathsf{SVars}$).

In other words, the j-th occurrence of a variable x is replaced with $x_j\{\Sigma^*\}$. 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 α, 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 σ with $\sigma(\alpha)$ to construct a corresponding w-tuple μ for ρ_α. Likewise, for every $w \in \mathcal{L}(\rho_\alpha)$, there exists a corresponding w-tuple μ from which we can reconstruct a pattern substitution σ with $\sigma(\alpha) = w$: By the construction of ρ_α, 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 $\sigma(x) := w_{\mu(x_i)}$. ◀

A.2 Proof of Proposition 22

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

R_{cyc}: Note that for all $x, y \in \Sigma^*$, x is a cyclic permutation of y (and vice versa) if and only if there exist $u, v \in \Sigma^*$ with $x = uv$ and $y = vu$. 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 α_x and α_y are defined as

$$\alpha_x := \Sigma^* x \{u_1\{\Sigma^*\} \cdot v_1\{\Sigma^*\}\} \Sigma^*,$$
$$\alpha_y := \Sigma^* y \{v_2\{\Sigma^*\} \cdot u_2\{\Sigma^*\}\} \Sigma^*.$$

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 $\mu \in P(w)$, 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_{com}: This proof relies on another fact from combinatorics on words. For all $x, y \in \Sigma^*$, $xy = yx$ holds if and only if $(x, y) \in R_{\text{com}}$. 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

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

In order to prove that $[\![P_{\mathrm{com}}]\!] = \Upsilon^{R_{\mathrm{com}}}_{x,y}$, first assume that, for some $w \in \Sigma^*$, $\mu \in P_{\mathrm{com}}(w)$. 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 α_1 to α_4:

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

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

A.3 Proof of Theorem 27

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 $\alpha_1 \cdot \alpha_2$, $\mathsf{SVars}(\alpha_1) \cap \mathsf{SVars}(\alpha_2) = \emptyset$ must hold.

While this is true in general, $\emptyset$ can be used to construct regex formulas that disprove this conjecture, e. g. $x\{\mathtt{a}\}(x\{\emptyset\} \vee \mathtt{b})$. As $\emptyset$ is never part of a parse tree, this regex formula 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 α is $\emptyset$-*reduced* if $\alpha = \emptyset$, or if α does not contain any occurrence of $\emptyset$. Using simple rewrite rules, we can observe the following.

▶ Claim 1. Given a regex formula α, we can compute in polynomial time an $\emptyset$-reduced regex formula α_R with $[\![\alpha_R]\!] = [\![\alpha]\!]$.

Proof of Claim 1. In order to compute α_R, it suffices to rewrite α 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 α and β, where β is obtained by applying any number of these rewrite rules, $[\![\beta]\!] = [\![\alpha]\!]$ holds. Furthermore, one can use these rules to convert α into an equivalent and $\emptyset$-reduced α_R in polynomial time: If α is stored in a tree structure, it suffices to apply all applicable rules in bottom-up manner. ◀(for 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 $\mathsf{SVars}(\rho) = \{x_1, \dots, x_n\}$, constructs an $\mathsf{EC}^{\mathsf{reg}}$-formula $\varphi_\rho(x_w, x_1^P, x_1^C, \dots x_n^P, x_n^C)$ such that for all $w, w_1^P, w_1^C, \dots, w_n^P, w_n^C \in \Sigma^*$, $\varphi_\rho(w, w_1^P, w_1^C, \dots, w_n^P, w_n^C) = \mathtt{True}$ 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 $[i_k, j_k\rangle = \mu(x_k)$.

The most complicated part of this proof is the construction of $\mathsf{EC}^{\mathsf{reg}}$-formulas from regex formulas.

▶ Claim 2. From every functional regex formula $\rho \in \mathsf{RGX}$ with $\mathsf{SVars}(\rho) = \{x_1, \dots, x_n\}$, $n \geq 0$, we can construct in polynomial time an $\mathsf{EC}^{\mathsf{reg}}$-formula $\varphi_\rho(x_w, x_1^P, x_1^C, \dots, x_n^P, x_n^C)$ that realizes $[\![\rho]\!]$.

Proof of Claim 2. Due to Claim 1, we can assume without loss of generality that ρ is $\emptyset$-reduced. We define φ_ρ recursively as follows:

1. If ρ does not contain any variables (i. e., $n = 0$), ρ is a proper regular expression. Using canonical transformation techniques, we can construct in polynomial time a nondeterministic finite automaton A with $\mathcal{L}(A) = \mathcal{L}(\rho)$, and we define

$$\varphi_\rho(x_w) := L_A(x_w).$$

Then $\varphi_\rho(w)$ is true if and only if $L_A(w)$ is true, which holds if and only if $w \in \mathcal{L}(A) = \mathcal{L}(\rho)$.

2. If ρ contains variables, we assume that $\mathsf{SVars}(\rho) = \{x_1, \dots, x_n\}$ with $n \geq 1$. By definition of regex formulas, no variable of ρ may occur inside of a Kleene star. Hence, we can distinguish three cases:

a. $\rho = \rho_1 \vee \rho_2$, where ρ_1, ρ_2 are functional regex formulas with $\mathsf{SVars}(\rho_1) = \mathsf{SVars}(\rho_2) = \mathsf{SVars}(\rho)$. We define

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

This formula is self explanatory: $\mu \in [\![\rho]\!](w)$ holds if and only if $\mu \in [\![\rho_1]\!](w)$ or $\mu \in [\![\rho_2]\!](w)$.

b. $\rho = \rho_1 \cdot \rho_2$, where ρ_1, ρ_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 $\mathsf{SVars}(\rho_1) = \{x_1, \dots, x_m\}$ and $\mathsf{SVars}(\rho_2) = \{x_{m+1}, \dots, x_n\}$ with $0 \leq m \leq n$. We define

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

The idea behind this formula is as follows: As $\rho = \rho_1 \vee \rho_2$, whenever $[\![\rho]\!](w) \neq \emptyset$ holds, w can be decomposed into $w = w_1 \cdot w_2$, where w_1 is parsed in ρ_1, and w_2 in ρ_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 ρ_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.

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

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

This formula uses the fact that in this case, for each $\mu \in [\![\rho]\!](w)$, $\mu(x_1) = [1, |w| + 1\rangle$ must hold. This is encoded by $x_1^P = \varepsilon$ and $x_1^C = w$.

Now note that the construction of φ_ρ follows the syntax of ρ (no decisions beyond this are necessary), and the size of φ_ρ is polynomial in the size of ρ. Hence, φ_ρ can be computed in polynomial time. Finally, a straightforward induction on the structure of ρ shows that φ_ρ realizes $[\![\rho]\!]$. ◀(for Claim 2)

Now consider the case of an arbitrary core spanner representation $\rho \in \mathsf{RGX}^{\{\pi, \zeta^=, \cup, \bowtie\}}$ with $\mathsf{SVars}(\rho) = \{x_1, \ldots, x_n\}$, $n \geq 0$. We distinguish the following cases:

1. $\rho = \pi_Y \hat{\rho}$, with $Y = \mathsf{SVars}(\rho)$ and $\mathsf{SVars}(\hat{\rho}) \supseteq \mathsf{SVars}(\rho)$. Assume w. l. o. g. that $\mathsf{SVars}(\hat{\rho}) = \{x_1, \ldots, x_{n+m}\}$ with $m \geq 0$. We define

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

2. $\rho = \zeta^=_{\vec{x}} \hat{\rho}$, with $\vec{x} \in (\mathsf{SVars}(\rho))^m$, $2 \leq m \leq n$, and $\mathsf{SVars}(\hat{\rho}) = \mathsf{SVars}(\rho)$. Assume w. l. o. g. that $\vec{x} = x_1, \ldots, x_m$. We define

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

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.

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

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

In this case, we use $\mu \in [\![\rho]\!](w)$ if and only if $\mu \in [\![\rho_1]\!](w)$ or $\mu \in [\![\rho_2]\!](w)$.

4. $\rho = (\rho_1 \bowtie \rho_2)$ 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 $0 \leq l \leq n$ and $1 \leq m \leq n + 1$. We define

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

First, note that we have ensured that $\mathsf{SVars}(\rho_1) \cap \mathsf{SVars}(\rho_2) = \{x_l, \ldots x_m\}$. 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 $i \in \{l, \ldots, m\}$. For each of these variables x_i, φ_{ρ_1} and φ_{ρ_2} model the span with the same variables x_i^P and x_i^C.

5. ρ is a regex formula. This case is covered in Claim 2.

The formula φ_ρ can be derived from ρ without requiring further computation, and its size is polynomial in the size of ρ. Hence, φ_ρ can be constructed in polynomial time. A straightforward induction on the structure of ρ shows that φ_ρ realizes $[\![\rho]\!]$. ◀

A.4 Proof of Theorem 28

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 18. Only two changes are made. Firstly, instead of binding a variable x to some Σ^*, we respect the constraints by using a regular expression for $\mathsf{Cstr}(x)$. Secondly, in order to ensure $\mathsf{SVars}(\rho) \subseteq \mathsf{Vars}(\eta)$, 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 $\hat{\eta}_1 := \hat{\alpha}_1 \cdots \hat{\alpha}_m$ and $\hat{\eta}_2 := \hat{\alpha}_{m+1} \cdots \hat{\alpha}_n$, where for each position in $1 \leq i \leq n$, we define $\hat{\alpha}_i$ as follows:

1. If α_i is a terminal (i. e., there is an $a \in \Sigma$ with $\alpha_i = a$), let $\hat{\alpha}_i := a$.

2. If α_i is a variable (i. e., there is an $x \in X$ with $\alpha_i = x$), let γ be a regular expression with $\mathcal{L}(\gamma) = \mathsf{Cstr}(x)$. 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 $k := |\eta_L \eta_R|_x \geq 2$, 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 σ of η under constraints Cstr 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 σ is a solution of η under constraints Cstr. Let $w := \sigma(\eta_L)$ (which implies $w = \sigma(\eta_R)$, as σ is a solution of η). We use this to define a w-tuple μ as follows: Due to our construction, each variable $\hat{x} \in \mathsf{SVars}(\rho)$ corresponds to a uniquely defined α_i with $\alpha_i = x$. If $1 \leq i \leq m$, $\hat{x}$ occurs in $\hat{\eta_L}$, and if $m+1 \leq i \leq n$, $\hat{x}$ 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

$$l := |\sigma(\alpha_1 \cdots \alpha_{i-1})| + 1, \qquad r := |\sigma(\alpha_1 \cdots \alpha_i)| + 1$$

- If $\hat{x}$ occurs in $\hat{\eta_R}$, let

$$l := |\sigma(\alpha_{m+1} \cdots \alpha_{i-1})| + 1, \qquad r := |\sigma(\alpha_{m+1} \cdots \alpha_i)| + 1.$$

Either way, we know that $w_{\mu(\hat{x})} = \sigma(x)$ holds, which implies $w_{\mu(\hat{x})} \in \mathsf{Cstr}(x)$. Analogously, we can use σ 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 σ 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 $\mathcal{L}(\gamma) = \mathsf{Cstr}(x)$ must hold, which means that $w_{\mu(x)} \in \mathsf{Cstr}(x)$, an hence $\sigma(x) \in \mathsf{Cstr}(x)$. 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 $\alpha_i = a \in \Sigma$, $\hat{w}_i := a$. 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

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

We observe $\sigma(\eta_1) = \sigma(\eta_2) = w$, which concludes this direction of the proof. ◀

A.5 Proof of Lemma 32

Proof. If a vstar-free regex α is not a regex 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. β is a disjunction; i. e., $\beta = (\gamma_1 \vee \gamma_2)$ for some $\gamma_1, \gamma_2 \in \mathsf{vsfRX}$,
2. β contains a variable binding $x\{\cdots\}$ or a variable reference $\&x$.

We now rewrite α into two vstar-free regexes α_1 and α_2, by replacing β with γ_1 or γ_2, respectively. We observe that this rewriting step does not change the language:

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

Proof of Claim. If $w \in \mathcal{L}(\alpha)$, there exists an α-parse tree T for w; in other words, the root of T is labeled with (w, α). Now we distinguish two possibilities: If T does not use the occurrence of β that was rewritten to create α_1 and α_2, we can immediately transform T into an α_i-parse tree T_i ($i \in \{1, 2\}$) by replacing the root label with (w, α_i), and changing all children accordingly. Hence, $w \in \mathcal{L}(\alpha_i)$ holds.

On the other hand, if T uses this occurrence of β, then there exists a 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 β that was rewritten in α_1 and α_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 α_i-parse tree T_i by removing v (i. e., $\hat{v}$ replaces v), relabeling the root of T to (w, α_i), and changing all labels between the root and $\hat{v}$ accordingly. As T_i is a α_i-parse tree for w, $w \in \mathcal{L}(\alpha_i)$ 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 α_i-parse tree for w into an α-parse tree by inserting a node $(\hat{w}, \beta)$ (if necessary), and changing the labels accordingly. ◀(for Claim)

Note that this equivalence relies on the fact that α is vstar-free, which implies that β does not occur inside a Kleene star. For regexes 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 regex that is not a regex path. This procedure terminates, as every rewriting removes a disjunction that contains at least one variable (binding or reference). Hence if α contains $k \in \mathbb{N}_{>0}$ disjunctions, this process results in regex paths $\alpha_1, \ldots, \alpha_n$ for some $n \leq 2^k$, and $\mathcal{L}(\alpha) = \bigcup_{i=1}^{n} \mathcal{L}(\alpha_i)$. ◀

A.6 Proof of Lemma 34

Proof. Note that, according to our definition of regexes, no variable binding for a variable x may contain another binding of x. More formally, for every regex $\alpha = x\{\hat{\alpha}\}$, no $\beta \in \mathsf{Sub}(\hat{\alpha})$ may be of the form $\beta = x\{\hat{\beta}\}$. This holds in particular for regex paths.

As explained above, the regex path α can be understood as a concatenation $\alpha = \alpha_1 \cdots \alpha_n$, where each α_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 a regex path.

If we now consider any α-parse tree T and pick any occurrence of any variable reference $\&x$ in α, we know that there is a uniquely defined node v in T that is labeled with $(w, \&x)$ (for some word w) and that corresponds to this occurrence of $\&x$. Moreover, all nodes in T that are above v are labeled with some concatenation. We can observe an analogous statement for all occurrences of variable bindings.

Hence, for every occurrence of a variable reference $\&x$ in α, exactly one of the following two cases applies:

1. In every α-parse tree, this occurrence of $\&x$ uses the default value ε instead of using a variable binding.
2. In every α-parse tree, this occurrence of $\&x$ uses a variable binding $x\{\cdots\}$.

Furthermore, in the second case, it is uniquely defined which occurrence of $x\{\cdots\}$ determines the value of this occurrence of $\&x$.

Therefore, we can rewrite α into a regex path β with $\mathcal{L}(\beta) = \mathcal{L}(\alpha)$ that has the additional property that every variable binding $x\{\cdots\}$ occurs at most once in β. If a binding $x\{\cdots\}$ occurs twice, we can rename one occurrence to $\hat{x}\{\cdots\}$ (where $\hat{x}$ is a new variable), and all occurrences of $\&x$ are renamed accordingly to $\&\hat{x}$.

Likewise, we can rewrite β into a regex path γ with $\mathcal{L}(\gamma) = \mathcal{L}(\alpha)$ with the additional property that no occurrence of a variable reference $\&x$ uses the default value ε, by replacing all those occurrences with ε.

We are now ready to transform γ into a regex formula δ by replacing all variable references in a manner that is similar to the proof of Theorem 18. More specifically, we construct δ by replacing, for each $x \in \mathsf{Vars}(\gamma)$, the i-th occurrence of $\&x$ in γ with $x_i\{\Sigma^*\}$. Note that δ is functional: Each variable in $\mathsf{SVars}(\delta)$ appears exactly once in δ; and as δ is also a regex path, this implies that every δ-parse tree contains every variable exactly once.

For every variable x for which there occur references $\&x$ in γ, 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, $\mathcal{L}(\rho) = \mathcal{L}(\alpha)$. ◀

A.7 Proof of Theorem 38

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 $\mathsf{Vars}(\alpha) = \{x_1, \dots, x_k\}$ for some $k \in \mathbb{N}_{>0}$.

It suffices to prove the claim for $\alpha \in \mathsf{RXP}$, as semi-linear sets are closed under union, and (according to Lemma 32) every vstar-free regex is equivalent to a finite union of regex paths.

As explained in the proof of Lemma 34 (in the construction of γ), we can also assume without loss of generality that every variable binding $x\{\cdots\}$ occurs exactly once in α, and that no variable reference $\&x_i$ uses the default binding ε. In particular, this means that in every α-parse tree, each variable x_i stores exactly one word w_i.

Let α be a regex path that satisfies these conditions. Our goal is to construct a Presburger formula φ 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 φ.

We use $\vec{x}$ as an abbreviation for $x_1^{\mathtt{a}}, x_1^{\mathtt{b}}, \dots, x_k^{\mathtt{a}}, x_k^{\mathtt{b}}$ and define $\varphi(n^{\mathtt{a}}, n^{\mathtt{b}}) := \exists \vec{x} \colon \varphi_\alpha(n^{\mathtt{a}}, n^{\mathtt{b}}, \vec{x})$, where φ_α is constructed according to the following general procedure.

Given a regex path γ, we define a Presburger formula φ_γ as follows: First, as γ is a regex path, there is a decomposition $\gamma = \gamma_1 \cdot \gamma_2 \cdots \gamma_l$ ($l \in \mathbb{N}_{>0}$), where each γ_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 a regex path. For each γ_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 γ_i. Keeping this in mind, we define

$$\varphi_\gamma(n^{\mathtt{a}}, n^{\mathtt{b}}, \vec{x}) := \exists n_1^{\mathtt{a}}, n_1^{\mathtt{b}}, \dots n_l^{\mathtt{a}}, n_l^{\mathtt{b}} \colon \\ \Big((n^{\mathtt{a}} = n_1^{\mathtt{a}} + \cdots + n_l^{\mathtt{a}}) \wedge (n^{\mathtt{b}} = n_1^{\mathtt{b}} + \cdots + n_l^{\mathtt{b}}) \wedge \bigwedge_{i=1}^{l} \varphi_{\gamma_i}(n_i^{\mathtt{a}}, n_i^{\mathtt{b}}, \vec{x}) \Big),$$

where the Presburger formulas are defined as follows:

- If γ_i is a proper regular expression, then as $\mathcal{L}(\gamma_i)$ is semi-linear (as a consequence of Parikh's theorem [27], every regular language is semi-linear). Hence, due to Ginsburg and Spanier [15], 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 $\varphi_{\gamma_i}(n_i^{\mathtt{a}}, n_i^{\mathtt{b}}, \vec{x}) := \hat{\varphi}_{\gamma_i}(n_i^{\mathtt{a}}, n_i^{\mathtt{b}})$.
- If $\gamma_i = \&x_j$ for some $1 \leq j \leq l$, we define

$$\varphi_{\gamma_i}(n_i^{\mathtt{a}}, n_i^{\mathtt{b}}, \vec{x}) := (n_i^{\mathtt{a}} = x_j^{\mathtt{a}}) \wedge (n_i^{\mathtt{b}} = x_j^{\mathtt{b}}).$$

- If $\gamma_i = x_j\{\delta\}$ for some $1 \leq j \leq l$ and some regex path δ, we define

$$\varphi_{\gamma_i}(n_i^{\mathtt{a}}, n_i^{\mathtt{b}}, \vec{x}) := (n_i^{\mathtt{a}} = x_j^{\mathtt{a}}) \wedge (n_i^{\mathtt{b}} = x_j^{\mathtt{b}}) \wedge \varphi_\delta(n_i^{\mathtt{a}}, n_i^{\mathtt{b}}, \vec{x}).$$

While the definition recurses in the case of regex paths that contain variable bindings (the third case in the definition of φ_{γ_i} above), the formula φ is still ensured to be finite and well-defined (as δ is always a subexpression of γ 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. ◀

A.8 Proof of Lemma 39

Proof. Assume that there is an $\alpha \in \mathsf{vsfRX}$ with $\mathcal{L}(\alpha) = L_{\text{nsl}}$. By Theorem 38, L_{nsl} must be semi-linear. Note that $\Psi(L_{\text{nsl}}) = \{(n, mn) \mid m, n \neq 2\}$. As semi-linear sets are closed under projection, this implies that $C := \{mn \mid m, n \geq 2\}$ is semi-linear. But C is the set of all composite numbers, which is not semi-linear. ◀

A.9 Proof of Theorem 41

Proof. This proof relies on vset-automata, consistent vset-paths, and vset-path unions which are defined in [7]. As we do not use these notions outside of this proof, we omit the definitions, and refer to [7].

We start with some preliminary observations. First, note that according to the core-simplification-lemma in [7], for every core spanner P, there exist a vset-automaton A, a sequence of string equality selections S, and a set of variables $V \subseteq \mathsf{SVars}(A)$ with $P = \pi_V S[\![A]\!]$.

Furthermore, as stated in Lemma 4.1 in [7], for every vset-automaton A, there exists a vset-path union G with $[\![G]\!] = [\![A]\!]$. Hence, $P = \pi_V S[\![G]\!]$. Also, by definition, for every vset-path union U, there exists a finite number of consistent vset-paths $G_1, \ldots, G_n$ with $[\![G]\!] = \bigcup_{i=1}^{n} [\![G_i]\!]$. Hence,

$$P = \pi_V S[\![G]\!] = \pi_V S \bigcup_{i=1}^{n} [\![G_i]\!] = \pi_V (\bigcup_{i=1}^{n} S[\![G_i]\!]).$$

As we are only interested in $\mathcal{L}(P)$, we can omit the projection, and only consider the spanner $P' := \bigcup_{i=1}^{n} S[\![G_i]\!]$, for which $\mathcal{L}(P) = \mathcal{L}(P')$ holds. Finally, as semi-linear sets are closed under union, it suffices to show that each language that is recognized by a spanner $S[\![G_i]\!]$ is semi-linear:

▶ **Claim.** *Given a consistent vset-path U and sequence S of string equality selections on $\mathsf{SVars}(G)$, the spanner $P_{SG} := S[\![G]\!]$ recognizes a semi-linear language.*

Proof of Claim. As Σ is unary, assume that $\Sigma = \{\mathsf{a}\}$. We prove the claim by giving a construction method for a Presburger formula φ_{SG} such that

$$\mathcal{L}(P_{SG}) = \{\mathsf{a}^n \mid n \in \mathbb{N}, \varphi_{SG}(n) \text{ is true}\}.$$

We assume that $\mathsf{SVars}(G) = \{x_1, \ldots, x_k\}$ for some $k \in \mathbb{N}$. Furthermore, there exists an $l \in \mathbb{N}$ such that $\gamma_1, \ldots \gamma_l$ are the (proper) regular expressions that occur in G. As G is a consistent vset-path, for every i with $1 \leq i \leq k$, we can define o_i, c_i with $1 \leq o_i \leq c_i \leq l$ such that $\gamma_{o_i}, \gamma_{o_i+1}, \ldots, \gamma_{c_i}$ are the regular expressions that appear in G between $x_i\vdash$ and $\dashv x_i$. In other words, o_i and c_i denote where x_i is opened and closed (respectively), and define which regular expressions determine the content of x_i.

As a consequence of Parikh's theorem [27], every regular language is semi-linear. Therefore, due to Ginsburg and Spanier [15], for every γ_i $(1 \leq i \leq l)$, there exists a Presburger formula φ_i^{γ} such that

$$\mathcal{L}(\gamma_i) = \{\mathsf{a}^j \mid \varphi_i^{\gamma}(j) \text{ is true}\}.$$

In order to process the string equality selection in S, we define the set E_S of all pairs $(i, j) \in \mathbb{N}^2$ with $i \neq j$, such that S contains a string selection $\zeta_V^{=}$, and $x_i, x_j \in V$. Hence, E_S contains the pairs of all indices of variables that are joined in a selection in S.

We now define the Presburger formula φ_{SG} by

$$\varphi_{SG}(n) := \exists m_1, \ldots, m_k \colon \exists a_1, \ldots, a_l \colon$$
$$\Big((n = a_1 + a_2 + \cdots + a_l) \wedge \bigwedge_{i=1}^{l} \varphi_i^{\gamma}(a_i) \wedge \bigwedge_{i=1}^{k} (m_i = a_{o_i} + a_{o_i+1} + \cdots + a_{c_i}) \wedge \bigwedge_{(i,j)\in E_S} m_i = m_j \Big)$$

Here, the variables $a_1, \ldots, a_l$ describe which parts of the word $\mathtt{a}^n$ are matched to the regular expressions $\gamma_1, \ldots, \gamma_l$, and $m_1, \ldots, m_k$ represent the contents of the variables $x_1, \ldots, x_k$.

It is now easy to see that the first three of the four subformulas in $\varphi_{SG}(n)$ are true if and only if the following holds:

- $\mathtt{a}^n = \mathtt{a}^{a_1} \cdots \mathtt{a}^{a_l}$,
- $\mathtt{a}^i \in \mathcal{L}(\gamma_i)$ for $1 \leq i \leq l$,
- there is a run of G on $\mathtt{a}^n$ such that each variable x_i stores the span $[o, c\rangle$, where $o := \sum_{1 \leq j < o_i} a_j$ and $c := 1 + \sum_{1 \leq j \leq c_i} a_j$, and this span has length m_i.

As the fourth subformula expresses that variables that are in a string equality selection in S must be matched to equal strings (or, as Σ is unary, strings of the same length), we can conclude that $\varphi_{SG}(n)$ is true if and only if $\mathtt{a}^n \in \mathcal{L}(P_{SG})$. Hence, the language $\mathcal{L}(P_{SG})$ is semi-linear. ◀(for Claim)

As laid out in the preliminary observations of this proof, we can now conclude from the closure of semi-linear sets under union that for every core spanner P over a unary alphabet, $\mathcal{L}(P)$ is semi-linear. ◀

B Results from Section 4

B.1 Proof of Theorem 42

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 α and a word w, we use Theorem 18 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 ρ represents a Boolean spanner, we define μ to be the empty tuple $()$. Now, $\mu \in [\![\rho]\!](w)$ holds if and only if $w \in \mathcal{L}(\alpha)$.

In order to show membership in NP, it suffices to consider the following NP-algorithm. Assume that we are given a core spanner representation ρ, a word $w \in \Sigma^*$, and a w-tuple μ. For every regex formula γ in ρ, we nondeterministically guess a w-tuple μ_γ. 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 ρ_i that is ignored. We then verify these guesses deterministically: First, we discard all parts of ρ that are ignored, and obtain a spanner representation $\hat{\rho} \in \mathsf{RGX}^{\{\pi, \zeta^=, \bowtie\}}$. For all remaining regex formulas γ in $\hat{\rho}$, we check whether μ_γ is consistent with γ 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 $\hat{\mu} = \mu$. ◀

B.2 Proof of Theorem 43

Proof. This result follows from a slight change to the NP-decision procedure from the proof of Theorem 42. Note that we can represent the guessed w-tuples μ_γ for each regex formula γ by using two pointers for each $\mu_\gamma(x) = [i, j\rangle$ (one pointer for i, one for j). As ρ 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. ◀

B.3 Proof of Theorem 44

Proof. We begin with the upper bound. According to Theorem 27, for every core spanner representation ρ, there exists an $\mathsf{EC}^{\mathsf{reg}}$-formula φ that realizes $[\![\rho]\!]$. Furthermore, φ can be computed in polynomial time. In particular, φ is satisfiable if and only if ρ is satisfiable. As satisfiability for $\mathsf{EC}^{\mathsf{reg}}$-formulas is in PSPACE (cf. Diekert [5]), 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 Kozen [22], this problem is PSPACE-complete. Recall that every proper regular expression is also a functional regex formula. Hence, we can construct a Boolean spanner

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

Obviously, for every $w \in \Sigma^*$, $P(w) \neq \emptyset$ if and only if there exists a $v \in \Sigma^*$ 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 $\mathsf{RGX}^{\{\zeta^=\}}$. ◀

B.4 Proof of Theorem 45

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 polynomialtime procedure that, given a $\rho \in \mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$, constructs an $\mathsf{EC}^{\mathsf{reg}}$-formula φ_{NH} that is satisfiable if and only if $[\![\rho]\!]$ is not hierarchical.

Recall that, by definition, for every spanner P and every $w \in \Sigma^*$, 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. $i_1 < i_2$, $i_2 < j_1$, and $j_1 < j_2$,
2. $i_2 < i_1$, $i_1 < j_2$, and $j_2 < j_1$.

For an illustration of these two conditions, see Figure 2. 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 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

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

Figure 2 The two possibilities how two spans can strictly overlap (see proof of Theorem 45). To the left: $i_1 < i_2 < j_1 < j_2$. To the right: $i_2 < i_1 < j_2 < j_1$.

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

$$\varphi_{\text{ovl}}(x^P, x^C, y^P, y^C) := \exists z_1, z_2 : \\ \left(\left(z_1 = x^P x^C \right) \wedge \left(z_2 = y^P y^C \right) \wedge \varphi_{\text{ppref}}\left(x^P, y^P \right) \wedge \varphi_{\text{ppref}}\left(y^P, z_1 \right) \wedge \varphi_{\text{ppref}}\left(z_1, z_2 \right) \right).$$

The idea behind the construction is as follows: Recall that this formula is going to be used in together with an EC^{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, φ_{ovl} checks whether the first of the two strict overlap conditions is satisfied.

We are now ready to construct φ_{NH}. Let $\rho \in \mathsf{RGX}^{\{\pi, \zeta^=, \cup, \bowtie\}}$, and assume $\mathsf{SVars}(\rho) = \{x_1, \dots, x_n\}$ for some $n \geq 2$ (spanners with less than two variables are trivially hierarchical). First, we construct an EC^{reg}-formula $\varphi_\rho(x_w, x_1^P, x_1^C, \dots, x_n^P, x_n^C)$ that realizes $[\![\rho]\!]$ (as in Theorem 27). We now define

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

Assume that $[\![\rho]\!]$ is not hierarchical. Then there exists a $w \in \Sigma^*$, a $\mu \in [\![\rho]\!]$ and $x_l, x_m \in \mathsf{SVars}(\rho)$ such that $\mu(x_l)$ and $\mu(x_m)$ strictly overlap. As φ_ρ realizes $[\![\rho]\!]$, μ defines an assignment $(w, w_{[1,i_1\rangle}, w_{[i_1,j_1\rangle}, \dots, w_{[1,i_n\rangle}, w_{[i_n,j_n\rangle})$ that satisfies this subformula (where $[i_k, j_k\rangle = \mu(x_k)$). Furthermore, as $\mu(x_i)$ and $\mu(x_j)$ 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, φ_{NH} is satisfiable.

Likewise, φ_{NH} is only satisfied if φ_ρ 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 μ where $\mu(x_l)$ and $\mu(x_m)$ strictly overlap. Hence, μ is not hierarchical, which means that $[\![\rho]\!]$ is not hierarchical.

Therefore, φ_{NH} is satisfiable if and only if $[\![\rho]\!]$ is not hierarchical. Furthermore, φ_{NH} can be constructed in polynomial time, as we only need to construct φ_ρ (which is possible in polynomial time, according to the proof of Theorem 44), and an amount of φ_{ovl}-formulas that is quadratic in $|\mathsf{SVars}(\rho)|$, each of which has a constant length. Both constructions rely solely on the syntax of ρ, 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 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 α_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, 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 ρ 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, ρ 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. ◀

B.5 Proof of Theorem 46

Proof. As shown by Freydenberger [11], if $|\Sigma| \geq 2$, for regexes α, 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 [11] takes a Turing machine $\mathcal{X}$ (with some additional technical restrictions) and computes a regex $\alpha_{\mathcal{X}}$ 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 regexes $\alpha_{\mathcal{X}}$ are defined over the alphabet $\Sigma = \{0, \#\}$ and, when adapted to the notation of this paper, are always of the following shape:

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

It is important to note that all subexpressions except α_{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 $n \in \mathbb{N}_{>0}$ that depends on $\mathcal{X}$, where all α_i are regex 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 regex.

By Theorem 36, we can computably 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 regular. As this question is neither semi-decidable, nor co-semi-decidable, this applies to CSp-Regularity as well. ◀

B.6 Proof of Theorem 47

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

For unbounded sizes of Σ, this undecidability was proven by Jiang et al. [20], and Freydenberger and Reidenbach [12] adapted this proof to all (non-unary) finite terminal alphabets.

Given two patterns α, β, we can use Theorem 18 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 and, hence, not semi-decidable (as it is obviously co-semi-decidable). ◀

B.7 Proof of Theorem 49

Proof. This follows immediately from the undecidability of CSp-Universality (Theorem 46). By the definition of a complexity measure, there are only finitely many c-minimal representations of $\Upsilon_\emptyset$. Hence, if a computable minimization procedure MIN_c existed, we cold decide CSp-Universality for every core spanner representation ρ by checking if $\mathsf{MIN}_c(\rho)$ is a c-minimal representation of $\Upsilon_\emptyset$. ◀

B.8 Proof of Theorem 50

Proof. For the sake of a contradiction, assume that such a function f exists. 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 46), 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 48, F_n is finite, and we can computably 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 ρ_i as not equivalent to ρ.

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 46. ◀

B.9 Proof of Theorem 51

Proof. It suffices to prove the claim for Boolean core spanner representations (hence, we can focus on the case of 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 [23]). Without going into further details, note that Σ_1^0 is the family of all sets that are semi-decidable (recursively enumerable), Π_1^0 is the family of all thats that are co-semi-decidable (co-recursively enumerable), and $\Delta_1^0 = \Sigma_1^0 \cap \Pi_1^0$ is the family of all sets that are decidable.

Regarding the next level, Σ_2^0 is the family of all sets that are semi-decidable when using oracles for sets in Σ_1^0 (or in Π_1^0), Π_2^0 is the family of all sets that are co-semi-decidable when using such oracles. Finally, $\Delta_2^0 = \Sigma_2^0 \cap \Pi_2^0$ is the family of all sets that are decidable when using oracles for sets in Σ_1^0 or in Π_1^0.

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

▶ Claim 1. $\mathsf{COF} \notin \Delta_2^0$.

Proof of Claim 1. As shown in Freydenberger [11], the regexes that we used in the proof of Theorem 46 also prove that co-finiteness for vstar-free regexes is Σ_2^0-complete.

Hence, the proof of Theorem 46 also implies that COF is Σ_2^0-hard. This immediately implies $\mathsf{COF} \notin \Delta_2^0$; as otherwise, $\Sigma_2^0 = \Delta_2^0$ would hold, which contradicts the fact that the arithmetical hierarchy is a proper hierarchy. ◀(for 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 of Claim 2. 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 α 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 44).

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 $\mathsf{FIN} \in \Sigma^0_1$. ◀(for Claim 2)

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

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

Proof of Claim 3. 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 $\hat{\rho} \in \mathsf{FIN}$, $\mathcal{L}(\hat{\rho})$ 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 regular expression $\hat{\rho}$ with $\mathcal{L}(\hat{\rho}) = \Sigma^* \setminus \mathcal{L}(\rho)$. As $\hat{\rho} \in \mathsf{RGX}$, $\hat{\rho} \in \mathsf{BCSR}$ follows. This gives $\hat{\rho} \in \mathsf{FIN}$, while $[\![\hat{\rho}]\!] = \mathsf{C}([\![\rho]\!])$ holds by our choice of $\hat{\rho}$. ◀(for Claim 3)

We now begin with actual proof. Let c be a complexity measure for $\mathsf{RGX}^{\{\pi,\zeta^=,\cup,\bowtie\}}$. 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 COF is in Δ^0_2. We prove this by defining a decision procedure with oracles for Σ^0_1 and Π^0_2 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 FIN is in Σ^0_1. Hence, the first criterion can be decided with a Σ^0_1-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 Π^0_1-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 COF can be decided by using oracles for Σ^0_1 and Π^0_1, $\mathsf{COF} \in \Delta^0_2$ must hold. This contradicts Claim 1. As our only assumption was the existence of the recursive bound f, no such bound can exist. ◀