Splitting Spanner Atoms: A Tool for Acyclic Core Spanners

Dominik D. Freydenberger
Loughborough University, UK

Sam M. Thompson
Loughborough University, UK

Abstract

This paper investigates regex CQs with string equalities (SERCQs), a subclass of core spanners. As shown by Freydenberger, Kimelfeld, and Peterfreund (PODS 2018), these queries are intractable, even if restricted to acyclic queries. This previous result defines acyclicity by treating regex formulas as atoms. In contrast to this, we propose an alternative definition by converting SERCQs into FC-CQs – conjunctive queries in FC, a logic that is based on word equations. We introduce a way to decompose word equations of unbounded arity into a conjunction of binary word equations. If the result of the decomposition is acyclic, then evaluation and enumeration of results become tractable. The main result of this work is an algorithm that decides in polynomial time whether an FC-CQ can be decomposed into an acyclic FC-CQ. We also give an efficient conversion from synchronized SERCQs to FC-CQs with regular constraints. As a consequence, tractability results for acyclic relational CQs directly translate to a large class of SERCQs.

2012 ACM Subject Classification Theory of computation → Complexity theory and logic

Keywords and phrases Document spanners, information extraction, conjunctive queries

Digital Object Identifier 10.4230/LIPIcs.ICDT.2022.10

Related Version *Full Version*: https://arxiv.org/abs/2104.04758

Funding *Dominik D. Freydenberger*: Supported by EPSRC grant EP/T033762/1.

Acknowledgements The authors would like to thank Justin Brackemann, and the anonymous reviewers for all their helpful comments and suggestions.

1 Introduction

Document spanners were introduced by Fagin, Kimelfeld, Reiss, and Vansummeren [7] as a formalization of AQL, an information extraction query language used in IBM's SystemT. Informally, they can be described in two steps. First, so-called *extractors* convert an input document, a word over a finite alphabet, into relations of so-called *spans*. We assume the extractors to be *regex formulas* (as described in [7]), which are regular expressions with capture variables. Consider the following example of a regex formula

$$\gamma(x) := \Sigma^* \cdot x\{(\texttt{EBDT}) \vee (\texttt{ICDT})\} \cdot \Sigma^*.$$

Given some input word, $\gamma(x)$ can be used to extract a unary relation of spans such that each span represents a factor of the input word that is either "EBDT" or "ICDT".

The second step is that the extracted relations are combined using a relational algebra. Classes of spanners can be defined by the choice of relational operators. *Regular spanners* allow for union $\cup$, projection π, and natural join $\bowtie$. Depending on how they are represented, regular spanners have been shown to be efficient. For example, if a regular spanner is given as a so-called vset-automaton, results can be enumerated with constant delay after

© Dominik D. Freydenberger and Sam M. Thompson;
licensed under Creative Commons License CC-BY 4.0
25th International Conference on Database Theory (ICDT 2022).
Editors: Dan Olteanu and Nils Vortmeier; Article No. 10; pp. 10:1–10:18

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

linear time preprocessing [9, 2]. However, if a regular spanner is given as a join of regex formulas, evaluation is intractable – as shown in [13], evaluation for spanners of the form $P := \pi_\emptyset(\gamma_1 \bowtie \gamma_2 \cdots \bowtie \gamma_n)$ is NP-complete, even if P is acyclic.

Core spanners extend regular spanners by allowing equality selection $\zeta^=$, which checks whether two (potentially different) spans represent the same factor of the input document. Even when core spanners are restricted to queries of the form $\pi_\emptyset \zeta^=_{x_1,y_1} \cdots \zeta^=_{x_m,y_m} \gamma$ for a single regex formula γ, the evaluation problem is NP-complete [11]. Therefore, both joins and equalities introduce computational hardness.

Regex CQs can be understood as the spanner version of relational CQs, which are a central topic in database theory. In each case, a conjunctive query is a projection over a join of atoms. Apart from the setting, the key difference is that while the tables for relational CQs are usually part of the input, the tables for regex CQs are defined implicitly through the regex formulas. Hence, while one could extract these tables and then perform a standard CQ over the extractions, the number of tuples in the materialized relations may be exponential. As a consequence, tractable restrictions on relational queries (such as *acyclic* CQ*s*) do not lead to tractable fragments of regex CQs [13].

So-called SERCQs extend regex CQs by also allowing string equality, thus allowing us to examine both previously discussed sources of intractability. Consider the following SERCQ

$$P := \pi_{x,y}\ \zeta^=_{x,x'}\left(\gamma_{\mathsf{sen}}(z) \bowtie \gamma_{\mathsf{prod}}(x) \bowtie \gamma_{\mathsf{pos}}(y) \bowtie \gamma_{\mathsf{factors}}(x, x', z) \bowtie \gamma_{\mathsf{factor}}(y, z)\right),$$

where we assume γ_{sen} extracts sentences, γ_{prod} extracts product names, γ_{pos} extracts positive sentiments (such as "enjoyed"), and $\gamma_{\mathsf{factors}}(x, x', z)$ and $\gamma_{\mathsf{factor}}(y, z)$ ensure that x and x' are successive (but not necessarily consecutive) factors of z, and y is a factor of z respectively. Therefore, P extracts spans representing products that are mentioned twice within a sentence, along with a positive sentiment that appears in the same sentence.

Syntactic restrictions on conjunctive queries have been incredibly fruitful for finding tractable fragments. A well known result of Yannakakis [22] is that for *acyclic* conjunctive queries, evaluation can be solved in polynomial time. Further research on the complexity of acyclic conjunctive queries [15] and the enumeration of results for acyclic conjunctive queries [3] has shown the efficacy of this restriction. On the other hand, for document spanners, such syntactic restrictions are yet to unlock tractable fragments.

To address this gap, we consider a different approach and represent SERCQs as a conjunctive query fragment of the logic FC[REG], introduced by Freydenberger and Peterfreund [14]. This logic is based on word equations, regular constraints, and first-order logic connectives. Consider the following FC[REG] conjunctive query

$$\varphi := \mathsf{Ans}(x, y) \leftarrow (z \doteq z_2 \cdot x \cdot z_3 \cdot x \cdot z_4) \wedge (z \doteq z_5 \cdot y \cdot z_6) \wedge (z \mathrel{\dot\in} \gamma_{\mathsf{sen}}) \wedge (x \mathrel{\dot\in} \gamma_{\mathsf{prod}}) \wedge (y \mathrel{\dot\in} \gamma_{\mathsf{pos}}).$$

If γ_{sen} is a regular expression that accepts sentences, γ_{prod} accepts a product name, and γ_{pos} accepts a positive sentiment, then φ is "equivalent" to the previously given SERCQ. They are not equivalent in a strict sense – a key difference being that SERCQs reason over spans, whereas FC[REG]-CQs reason over factors of the input words. Reasoning over words does bring some advantages: For example, φ simply uses relations of words (for example, γ_{prod}) encoded as a regular expression, and if we wanted to do something analogous for regex-formulas, we would first have to extract the corresponding relation of spans.

When dealing with word equations, we run into an issue that we already encountered for regex formulas: Their relations may contain an exponential number of tuples. This is due to the unbounded arity of word equations. However, an FC atom can be considered shorthand for a concatenation term. For example, the word equation $y \doteq x_1 x_2 x_3 x_4$ can be represented

as $y \doteq f(f(x_1, x_2), f(x_3, x_4))$ where f denotes binary concatenation. This then lends itself to the "decomposition" of the word equation into a CQ consisting of smaller word equations. We can express the above word equation as $(y \doteq z_1 \cdot z_2) \wedge (z_1 \doteq x_1 \cdot x_2) \wedge (z_2 \doteq x_3 \cdot x_4)$. For such a *decomposition*, the relations defined by each word equation can be stored in linear space and we can enumerate them with constant delay. Thus, if the resulting query is acyclic, then the tractability properties of acyclic conjunctive queries directly translate to the FC-CQ.

Contributions of this paper. The goal of this work is to bridge the gap between acyclic relational CQs and information extraction. To this end, we define FC[REG]-CQs, a conjunctive query fragment of FC[REG], and show show that any so-called synchronized SERCQ can be converted into an equivalent FC[REG]-CQ in polynomial time (Lemma 3.6).

We define the decomposition of an FC-CQ into a 2FC-CQ, where 2FC-CQ denotes the set of FC-CQs where the right-hand side of each word equation is of at most length two. Our first main result is a polynomial-time algorithm that decides whether a *pattern*[1] can be decomposed into an acyclic 2FC-CQ (Theorem 4.12).

Building on this, we give a polynomial-time algorithm that decomposes an FC-CQ into an acyclic 2FC-CQ, or determines that this is not possible (Theorem 5.14). As soon as we have an acyclic 2FC-CQ, the upper bound results for model checking and enumeration of results follow from previous work on relational acyclic CQs [15, 3].

We mainly focus on FC-CQs (i. e., no regular constraints) due to the fact that we can add regular constraints for "free". This is because regular constraints are unary predicates, and therefore can be easily incorporated into a join tree. Thus, our work defines a class of FC[REG]-CQs for which model checking can be solved in polynomial time, and results can be enumerated with polynomial-delay (both in terms of combined complexity).

Our approach offers a new research direction for tractable document spanners. Most of the current literature approaches regular spanners by "compiling" the spanner representation (regex formulas that are combined with projection, union, and joins) into a single automaton, where the use of joins can lead to a number of states that is exponential in the size of the original representation. Instead, we look at decomposing FC conjunctive queries into small and tractable components. This allows us to use the wealth of research on relational algebra, while also allowing for the use of the string equality selection operator.

Related Work. Regarding data complexity, Florenzano, Riveros, Vgarte, Vansummeren, and Vrgoc [9] gave a constant-delay algorithm for enumerating the results of deterministic vset-automata, after linear time preprocessing. Amarilli, Bourhis, Mengal, and Niewerth [2] extended this result to non-deterministic vset-automata. Regarding combined complexity, Freydenberger, Kimelfeld, and Peterfreund [13] introduced regex CQs and proved that their evaluation is NP-complete (even for acyclic queries), and that fixing the number of atoms and the number of string equalities in SERCQs allows for polynomial-delay enumeration of results. Freydenberger, Peterfreund, Kimelfeld, and Kröll [12] showed that non-emptiness for a join of two sequential regex formulas is NP-hard, under schemaless semantics, even for a single character document. Connections between the theory of concatenation and spanners have been considered in [11, 10, 14], which give many of the lower bound complexity results for core spanners. Schmid and Schweikardt [21] examined a subclass of core spanners called refl-spanners, which incorporate string equality directly into a regular spanner. Peterfreund [19] considered extraction grammars, and gave an algorithm for unambiguous extraction grammars that enumerates results with constant-delay after quintic preprocessing.

[1] For the purposes of this introduction, a pattern can be considered a single FC atom.

2 Preliminaries

Let $\emptyset$ denote the *empty set*, and for $n \geq 1$ let $[n] := \{1, 2, \ldots, n\}$. Given a set S, we use $|S|$ for the *cardinality* of S. If S is a subset of T then we write $S \subseteq T$ and if $S \neq T$ also holds, then $S \subset T$. We write $\mathcal{P}(S)$ for the powerset of S. The difference of two sets S and T is denoted as $S \setminus T$. If $\vec{x}$ is a tuple, we write $x \in \vec{x}$ to indicate that x is a component of $\vec{x}$. Let A be an alphabet. We use $|w|$ to denote the length of some word $w \in A^*$ and ε to denote the *empty word*. The number of occurrences of $a \in A$ within w is $|w|_a$. We write $u \cdot v$ or just uv for the concatenation of words $u, v \in A^*$. If $u = p \cdot v \cdot s$ for $p, s \in A^*$ then v is a *factor* of u, denoted $v \sqsubseteq u$. If $u \neq v$ also holds, then $v \sqsubset u$. Let Σ be an alphabet of *terminal symbols* and let Ξ be an infinite alphabet of *variables*. We assume that $\Sigma \cap \Xi = \emptyset$ and $|\Sigma| \geq 2$.

If $T := (V, E)$ is a tree, then a path between $x_1 \in V$ and $x_n \in V$ is the shortest sequence of edges from x_1 to x_n. If $(\{x_1, x_2\}, \{x_2, x_3\}, \ldots, \{x_{n-1}, x_n\})$ is a path, then we say a node y *lies* on this path if $y = x_j$ for some $j \in [n]$. We call the number of edges on a path from x_1 to x_n the *distance* between x_1 and x_n.

Document Spanners. Given $w := w_1 \cdot w_2 \cdots w_n$ where $w_i \in \Sigma$ for all $i \in [n]$, a so-called *span* of w is an interval $[i, j\rangle$ where $1 \leq i \leq j \leq n + 1$. A span $[i, j\rangle$ defines a factor $w_{[i,j\rangle} := w_i \cdot w_{i+1} \cdots w_{j-1}$ of w. Let $V \subset \Xi$, where V is finite, and let $w \in \Sigma^*$. A (V, w)*-tuple* is a function μ that maps each $x \in V$ to a span $\mu(x)$ of w. A *spanner* P, with variables V, is a function that maps every $w \in \Sigma^*$ to a set $P(w)$ of (V, w)-tuples. By $\mathsf{Vars}\,(P)$, we denote the set of variables of P.

Like [7], we use *regex formulas* as the primary extractors. Regex formulas are an extension of regular expressions with so-called *capture variables*. More formally: $\emptyset$, ε, and $\mathtt{a}$ where $\mathtt{a} \in \Sigma$ are all regex formulas, and if γ_1 and γ_2 are regex formulas then so are $(\gamma_1 \cdot \gamma_2)$, $(\gamma_1 \vee \gamma_2)$, $(\gamma_1)^*$, and $x\{\gamma_1\}$ where $x \in \Xi$. We use Σ as a shorthand for $\bigvee_{a \in \Sigma} a$. We can omit the parentheses when the meaning is clear. A variable binding $x\{\gamma\}$ matches the same words as γ and assigns the corresponding span of the input word to x. A regex formula is *functional* if on every match, each variable is assigned exactly one span. We denote the set of functional regex formulas by RGX. For $\gamma \in \mathsf{RGX}$, we use $[\![\gamma]\!]$ to define the corresponding spanner as follows. Every match of γ on w defines μ, a $(\mathsf{Vars}\,(\gamma), w)$-tuple, where for each $x \in \mathsf{Vars}\,(\gamma)$, we have that $\mu(x)$ is the span assigned to x. We use $[\![\gamma]\!](w)$ to denote the set of all such $(\mathsf{Vars}\,(\gamma), w)$-tuples. See [7] for more details.

We now define *synchronized* RGX*-formulas* (this follows the definition by Freydenberger, Kimelfeld, Kröll, and Peterfreund in [12]). An expression $\gamma \in \mathsf{RGX}$ is *synchronized* if for all sub-expressions of the form $(\gamma_1 \vee \gamma_2)$, no variable bindings occur in γ_1 or γ_2. We denote the class of synchronized RGX-formulas by $\mathsf{RGX}_{\mathsf{sync}}$.

The motivation for synchronized RGX-formulas is that non-synchronized formulas allow for "hidden" disjunctions within the atoms. This goes (arguably) against the spirit of CQs and (as shown in [12]) leads to "un-CQ-like" behavior.

▶ **Example 2.1.** Consider the regex formula $\gamma := \Sigma^* \cdot x\{\mathtt{a} \vee (\mathtt{b})^*\} \cdot y\{\Sigma^*\} \cdot \Sigma^*$. We have that $[\![\gamma]\!](w)$ contains those μ such that $\mu(x)$ is a factor of w which is either an $\mathtt{a}$ or a sequence of $\mathtt{b}$ symbols, and the span $\mu(y)$ occurs directly after $\mu(x)$. Since γ is functional, and for every sub-expression of the form $(\gamma_1 \vee \gamma_2)$, we have that $\mathsf{Vars}\,(\gamma_1) = \mathsf{Vars}\,(\gamma_2) = \emptyset$, it follows that γ is a *synchronized regex formula*.

Essentially, a synchronized regex formula is functional if no variable is redeclared, and no variable is used inside of a Kleene star.

This is extended into a *relational algebra* comprised of $\cup$ (union), π (projection), $\bowtie$ (natural join), and $\zeta^=$ (string equality). Let $w \in \Sigma^*$, and let P_1 and P_2 be spanners. We say P_1 and P_2 are *compatible* if $\mathsf{Vars}(P_1) = \mathsf{Vars}(P_2)$. If two spanners P_1 an P_2 are compatible, then $(P_1 \cup P_2)(w) := P_1(w) \cup P_2(w)$. For $Y \subseteq \mathsf{Vars}(P_1)$, the *projection* $\pi_Y P_1(w)$ is defined as the restriction of all $\mu \in P_1(w)$ to the set of variables Y, and hence $\mathsf{Vars}(\pi_Y P_1) := Y$.

The *natural join*, $P_1 \bowtie P_2$, is obtained by defining $\mathsf{Vars}(P_1 \bowtie P_2) := \mathsf{Vars}(P_1) \cup \mathsf{Vars}(P_2)$, and $(P_1 \bowtie P_2)(w)$ as the set of all $(\mathsf{Vars}(P_1) \cup \mathsf{Vars}(P_2), w)$-tuples for which there exists $\mu_1 \in P_1(w)$ and $\mu_2 \in P_2(w)$ such that $\mu_1(x) = \mu_2(x)$ for all $x \in \mathsf{Vars}(P_1) \cap \mathsf{Vars}(P_2)$. The *string equality operator* $\zeta^=_{x_1,x_2} P_1$ is defined by $\zeta^=_{x_1,x_2} P_1(w) := \{\mu \in P_1(w) \mid w_{\mu(x_1)} = w_{\mu(x_2)}\}$, where $\mathsf{Vars}\left(\zeta^=_{x_1,x_2} P_1\right) := \mathsf{Vars}(P_1)$.

Given a class of regex-formulas C and a spanner algebra O, we use C^{O} to denote the set of spanner representations which can be constructed by repeated combinations of operators from O with a regex-formula from C. We write $[\![C^{\mathsf{O}}]\!]$ to denote the closure of $[\![C]\!]$ under O.

The class of *core spanners* (introduced by Fagin, Kimelfeld, Reiss, and Vansummeren [7]) is defined as $[\![\mathsf{RGX}^{\mathsf{core}}]\!]$ where $\mathsf{core} := \{\pi, \zeta^=, \cup, \bowtie\}$. The class of *regex* CQ*s with string equality* (SERCQs) is defined as expressions of the form:

$$P := \pi_Y \left(\zeta^=_{x_1,y_1} \cdots \zeta^=_{x_l,y_l} (\gamma_1 \bowtie \cdots \bowtie \gamma_k)\right),$$

where $\gamma_i \in \mathsf{RGX}$ for all $i \in [k]$. We call an SERCQ a *synchronized* SERCQ if every regex formula is a synchronized RGX-formula.

▶ **Example 2.2.** Consider $P := \zeta^=_{x_1,x_2}(\gamma_1 \bowtie \gamma_2)$ where $\gamma_1 := \Sigma^* \cdot x_1\{\Sigma^+\} \cdot \mathtt{a} \cdot \Sigma^*$ and $\gamma_2 := \Sigma^* \cdot x_2\{\Sigma^+\} \cdot \mathtt{b} \cdot \Sigma^*$. Given $w \in \Sigma^*$, we have that $[\![P]\!](w)$ contains those μ such that the factor $w_{\mu(x_1)}$ is non-empty, and is immediately followed by the symbol $\mathtt{a}$, the factor $w_{\mu(x_2)}$ is immediately followed by the symbol $\mathtt{b}$, and $w_{\mu(x_1)} = w_{\mu(x_2)}$. Since both γ_1 and γ_2 are synchronized, P is a synchronized SERCQ.

Computational Model and Complexity Measures. We use the *random access machine* model with uniform cost measures, where the size of each machine word is logarithmic in the size of the input. We represent factors of a word $w \in \Sigma^*$ as spans of w. This allows us to check whether $u = v$ for $u, v \sqsubseteq w$ in constant time after preprocessing that takes linear time and space [16, 5] (see Proposition 4.1 for more details). The complexity results we state are in terms of *combined complexity*. That is, both the query and the word are considered part of the input. When considering the enumeration of results for a query executed on a word, we say that we can enumerate results with *polynomial-delay* if there exists an algorithm which returns the first result in polynomial time, the time between two consecutive results is polynomial, and the time between the last result and terminating is polynomial.

3 Conjunctive Queries for FC

This section introduces FC[REG]-CQs, a conjunctive query fragment of FC with regular constraints. We give some complexity results regarding SERCQs and show an efficient conversion from synchronized SERCQs to FC[REG]-CQs.

A pattern is a word $\alpha \in (\Sigma \cup \Xi)^*$, and a *word equation* is a pair $\eta := (\alpha_L, \alpha_R)$ where $\alpha_L, \alpha_R \in (\Sigma \cup \Xi)^*$ are patterns known as the *left* and *right* side respectively. We usually write such η as $(\alpha_L \doteq \alpha_R)$. The length of a word equation, denoted $|(\alpha_L \doteq \alpha_R)|$, is $|\alpha_L| + |\alpha_R|$. A *pattern substitution* is a morphism $\sigma\colon (\Sigma \cup \Xi)^* \to \Sigma^*$ such that $\sigma(\mathtt{a}) = \mathtt{a}$ holds for all $\mathtt{a} \in \Sigma$. Since σ is a morphism, we have $\sigma(\alpha_1 \cdot \alpha_2) = \sigma(\alpha_1) \cdot \sigma(\alpha_2)$ for all $\alpha_1, \alpha_2 \in (\Sigma \cup \Xi)^*$.

A pattern substitution σ is a *solution* to a word equation $(\alpha_L \doteq \alpha_R)$ if and only if $\sigma(\alpha_L) = \sigma(\alpha_R)$. When applying a pattern substitution σ to a pattern α, we assume that its domain $\mathsf{dom}(\sigma)$ satisfies $\mathsf{var}(\alpha) \subseteq \mathsf{dom}(\sigma)$. Freydenberger and Peterfreund [14] introduced FC as a first-order logic that is based on word equations. In the present paper, we do not consider the full logic FC. Instead, we introduce its conjunctive queries.

▶ **Definition 3.1.** *An* FC-CQ *is an* FC*-formula of the form* $\varphi(\vec{x}) := \exists\vec{y}\colon \bigwedge_{i=1}^{n} \eta_i$*, where* $\eta_i := (x_i \doteq \alpha_i)$*,* $x_i \in \Xi$*, and* $\alpha_i \in (\Sigma \cup \Xi)^*$ *for all* $i \in [n]$*. We use the shorthand* $\varphi := \mathsf{Ans}(\vec{x}) \leftarrow \bigwedge_{i=1}^{n} \eta_i$ *where* $\vec{x}$ *is the tuple of free variables. We call* $\mathsf{Ans}(\vec{x})$ *the* head *of* φ*, and* $\bigwedge_{i=1}^{n} \eta_i$ *the* body *of* φ*.*

We write $\varphi(\vec{x})$ to denote that $\vec{x}$ is the set of free variables of φ. The set of all variables used in φ is denoted by $\mathsf{var}(\varphi)$. We distinguish a variable $\mathfrak{u} \in \Xi$, called the *universe variable*, that shall represent the input document w. The universe variable is not considered a free variable, and we adopt the convention that $\mathfrak{u} \notin \mathsf{var}(\varphi)$ for all φ (even if $\mathfrak{u}$ occurs in φ). Next, we define the semantics for FC-CQs.

▶ **Definition 3.2.** *For* $\varphi \in$ FC-CQ *and a pattern substitution* σ *with* $\mathsf{var}(\varphi) \cup \{\mathfrak{u}\} \subseteq \mathsf{dom}(\sigma)$*, we define* $\sigma \models \varphi$ *as follows:* $\sigma \models (\alpha_l \doteq \alpha_R)$ *if* $\sigma(\eta_L) = \sigma(\eta_R)$ *and* $\sigma(x) \sqsubseteq \sigma(\mathfrak{u})$ *for all* $x \in \mathsf{var}(\alpha_L \doteq \alpha_R)$*. For* $\sigma \models \exists x\colon \varphi$ *we have that* $\sigma_{x\mapsto u} \models \varphi$ *holds for some* $u \sqsubseteq \sigma(\mathfrak{u})$*, where* $\sigma_{x\mapsto u}$ *is defined as* $\sigma_{x\mapsto u}(x) := u$ *and* $\sigma_{x\mapsto u}(y) := \sigma(y)$ *for all* $y \in (\Sigma \cup \Xi)$ *where* $y \neq x$*. We use the canonical definition for conjunction.*

Hence, for all $\sigma \models \varphi(\vec{x})$, the universe for variables in $\mathsf{var}(\varphi)$ is the set of factors of $\sigma(\mathfrak{u})$. If $\varphi(\vec{x}) \in$ FC-CQ and $w \in \Sigma^*$, then $[\![\varphi]\!](w)$ denotes the set of all $\sigma(\vec{x})$ such that $\sigma \models \varphi$ and $\sigma(\mathfrak{u}) = w$. When determining $[\![\varphi]\!](w)$ for a given w, we know that $\mathfrak{u}$ represents w, and hence $\mathfrak{u}$ can be treated as a constant (see [14] for more information on the role of the universe variable). If $\varphi \in$ FC is *Boolean* (that is, it has no free variables), $[\![\varphi]\!](w)$ is either the empty set, or the set containing the empty tuple, which we interpret as False and True, respectively.

In [14], FC was extended to FC[REG] by adding *regular constraints*. This allows for atoms of the form $(x \,\dot{\in}\, \gamma)$, where γ is a *regular expression*; and $\sigma \models (x \,\dot{\in}\, \gamma)$ if and only if $\sigma(x) \in \mathcal{L}(\gamma)$ and $\sigma(x) \sqsubseteq \sigma(\mathfrak{u})$. We extend FC-CQ to FC[REG]-CQ in the same way.

Complexity. We now define various decision problems for FC-CQ and FC[REG]-CQ: The *non-emptiness problem* is, given $w \in \Sigma^*$ and φ, decide whether $[\![\varphi]\!](w) \neq \emptyset$. The *evaluation problem* is, given σ and φ, decide whether $\sigma \models \varphi$. The *model checking problem* is the special case of non-emptiness and evaluation that only considers Boolean queries, note that for Boolean queries $\mathsf{Dom}(\sigma) = \{\mathfrak{u}\}$. Given $w \in \Sigma^*$ and φ, the *enumeration problem* is outputting all $[\![\varphi]\!](w)$. The *containment problem* is, given φ and ψ, decide whether $[\![\varphi]\!](w) \subseteq [\![\psi]\!](w)$ for all $w \in \Sigma^*$. Previous results on patterns and FC (see [4, 6, 14]) directly imply the following.

▶ **Proposition 3.3.** *For each of* FC-CQ *and* FC[REG]-CQ*, the evaluation problem is* NP*-complete, and the containment problem is undecidable.*

As discussed in [14], FC and FC[REG] can be evaluated analogously to relational first-order logic (FO), by materializing the tables that are defined by the atoms and then proceeding "as usual". Hence, bounding the width of a formula (the maximum number of free variables in a subformula) bounds the size of the intermediate tables, and thereby the complexity of evaluation. As the complexity of evaluating FC and FO are the same (PSPACE-complete in general, NP-complete for the existential-positive fragment), it is no surprise that this correspondence also translates to conjunctive queries. From Section 5 on, we further develop this connection by finding tractable subclasses of FC[REG]-CQ.

As containment for CQs is decidable (although NP-complete), it can be used for query minimization (see Chapter 6 of [1]). But by Proposition 3.3, this does not apply to FC-CQ.

Document Spanners and FC-CQs. Our next goal is to establish a connection between SERCQs and FC[REG]-CQs. However, first we must overcome the fact that FC reasons over strings, whereas spanners reason over intervals of positions. We deal with this by defining the notion of an FC-formula *realizing* a spanner, as described in [11, 10, 14].

▶ **Definition 3.4.** *A pattern substitution σ* expresses *a (V, w)-tuple μ, if for all $x \in V$, we have that* $\mathsf{Dom}(\sigma) = \{x^P, x^C \mid x \in V\}$*, and $\sigma(x^P) = w_{[1,i\rangle}$ and $\sigma(x^C) = w_{[i,j\rangle}$ for the span $\mu(x) = [i,j\rangle$. An* FC[REG]-CQ *φ* realizes *a spanner P if* $\mathsf{free}(\varphi) = \{x^P, x^C \mid x \in \mathsf{Vars}(P)\}$ *and $\sigma \models \varphi$ for all $w \in \Sigma^*$ where $\sigma(\mathfrak{u}) = w$, if and only if σ expresses some $\mu \in P(w)$.*

Less formally, for each $\mu \in P(w)$, we have that $\mu(x) = [i,j\rangle$ is uniquely represented by the prefix, $\sigma(x^P) = w_{[1,i\rangle}$, and the content, $\sigma(x^C) = w_{[i,j\rangle}$.

▶ **Example 3.5.** Consider the following FC[REG]-CQ.

$$\varphi := \mathsf{Ans}(x_1^P, x_1^C, x_2^P, x_2^C) \leftarrow (\mathfrak{u} \doteq x_1^P \cdot x_1^C \cdot \mathtt{a} \cdot s_1) \land (\mathfrak{u} \doteq x_2^P \cdot x_2^C \cdot \mathtt{b} \cdot s_2) \\ \land (x_1^C \doteq x_2^C) \land (x_1^C \mathrel{\dot\in} \Sigma^+) \land (x_2^C \mathrel{\dot\in} \Sigma^+).$$

We can see that φ realizes the SERCQ given in Example 2.2.

Recall that synchronized SERCQs consist of RGX-formulas that do not have variables within sub-expressions of the form $(\gamma_1 \lor \gamma_2)$. As we observe in the following result, a synchronized SERCQ can be efficiently translated into an equivalent FC[REG]-CQ.

▶ **Lemma 3.6.** *Given a synchronized* SERCQ *P, we can construct in polynomial time an* FC[REG]-CQ *that realizes P.*

The proof of Lemma 3.6 follows from [14, 11, 10]. The converse of Lemma 3.6 follows directly from [14]. However, one would need to define how FC[REG]-CQ-formulas can be realized by regex formulas closed under spanner algebra (details on this can be found in [10, 14]). We omit such a result as it is not the focus on this work.

In this section, we have introduced FC[REG]-CQs, and shown an efficient conversion from synchronized SERCQs to FC[REG]-CQs. Therefore, while the present paper mainly considers a tractable fragment of FC[REG]-CQ, this tractability carries over to a subclass of SERCQs.

4 Acyclic Pattern Decomposition

This section examines decomposing terminal-free patterns (i. e., patterns $\alpha \in \Xi^+$) into acyclic 2FC-CQs, where 2FC-CQ denotes the set of FC-CQs where each word equation has a right-hand side of at most length two. Patterns are the basis for FC-CQ atoms, and hence, this section gives us a foundation on which to investigate the decomposition of FC-CQs. We do not consider regular constraints, or patterns with terminals. This is because regular constraints are unary predicates, and therefore can be easily added to a join tree; and terminals can be expressed through regular constraints. We use 2FC-CQs for two reasons. Firstly, binary concatenation is the most elementary form of concatenation, as it cannot be decomposed into further (non-trivial) concatenations. Secondly, this ensures that each word equation has very low width, and therefore we can store the tables in linear space and enumerate them with constant delay – as shown in the following.

▶ **Proposition 4.1.** *Given $w \in \Sigma^*$, we can construct a data structure in linear time that, for $x, y, z \in \Xi$, allow us to enumerate $[\![x \doteq y \cdot z]\!](w)$ with constant-delay, and to decide in constant time if $\sigma \in [\![x \doteq y \cdot z]\!](w)$ holds.*

Although the cardinality of $[\![x \doteq y \cdot z]\!](w)$ is cubic in $|w|$, Proposition 4.1 allows us to represent this relation in linear space. As we can query such relations in constant time, they behave "nicer" than relations in relational algebra. Furthermore, after materializing the relations defined by each atom of an 2FC-CQ, Proposition 4.1 allows us to treat the 2FC-CQ as a relational conjunctive query. We now introduce a way to *decompose* a pattern into a conjunction of word equations where the right hand side of each atom is at most length two. We start by looking at a canonical way to decompose terminal-free patterns.

Let $\alpha \in \Xi^+$ be a terminal-free pattern. To decompose α, first we factorize α so that it can be written using only binary concatenation We define BPat, the set of all *well-bracketed patterns*, recursively as follows:

▶ **Definition 4.2.** $x \in \mathsf{BPat}$ *for all* $x \in \Xi$*, and if* $\tilde{\alpha}, \tilde{\beta} \in \mathsf{BPat}$*, then* $(\tilde{\alpha} \cdot \tilde{\beta}) \in \mathsf{BPat}$*.*[2]

We extend the notion of a factor to a *sub-bracketing.* We write $\tilde{\alpha} \sqsubseteq \tilde{\beta}$ if $\tilde{\alpha}$ is a factor of $\tilde{\beta}$ and $\tilde{\alpha}, \tilde{\beta} \in \mathsf{BPat}$. Let $\alpha \in \Xi^+$, by $\mathsf{BPat}(\alpha)$ we denote the set of all bracketings which correspond to the pattern α (i.e., if we remove the brackets, then the resulting pattern is α). Every $\tilde{\alpha} \in \mathsf{BPat}(\alpha)$ can be converted into an equivalent formula $\Psi_{\tilde{\alpha}} \in$ 2FC-CQ using the following.

▶ **Definition 4.3.** *While there exists $\tilde{\beta} \sqsubseteq \tilde{\alpha}$ where $\tilde{\beta} = (x \cdot y)$ for some $x, y \in \Xi$, we replace every occurrence of $\tilde{\beta}$ in $\tilde{\alpha}$ with a new, unique variable $z \in \Xi \setminus \mathsf{var}(\alpha)$ and add the word equation $(z \doteq x \cdot y)$ to $\Psi_{\tilde{\alpha}}$. When $\tilde{\alpha} = \tilde{\beta}$, we have that $z = \mathfrak{u}$.*

Therefore, up to renaming of variables, every $\tilde{\alpha} \in \mathsf{BPat}$ has a corresponding formula $\Psi_{\tilde{\alpha}} \in$ 2FC-CQ. We call $\Psi_{\tilde{\alpha}}$ the *decomposition* of $\tilde{\alpha}$. The decomposition can be thought of as a logic formula expressing a *straight-line program* of the pattern (see [17] for a survey on algorithms for SLPs). We now give an example of *decomposing* a bracketing.

▶ **Example 4.4.** Let $\alpha := x_1 x_2 x_1 x_1 x_2$ and let $\tilde{\alpha} \in \mathsf{BPat}(\alpha)$ be defined as follows:

$$\tilde{\alpha} := (((x_1 \cdot x_2) \cdot x_1) \cdot (x_1 \cdot x_2)).$$

We now list $\tilde{\alpha}$ after every sub-bracketing is replaced with a variable. We also give the corresponding word equation that is added to $\Psi_{\tilde{\alpha}}$.

$$(((\underline{(x_1 \cdot x_2)} \cdot x_1) \cdot \underline{(x_1 \cdot x_2)}) \qquad z_1 \doteq x_1 \cdot x_2$$
$$(\underline{(z_1 \cdot x_1)} \cdot z_1) \qquad z_2 \doteq z_1 \cdot x_1$$
$$\underline{(z_2 \cdot z_1)} \qquad \mathfrak{u} \doteq z_2 \cdot z_1$$

Therefore, we get the decomposition $\Psi_{\tilde{\alpha}} \in$ 2FC-CQ, which is defined as

$$\Psi_{\tilde{\alpha}} := \mathsf{Ans}() \leftarrow (z_1 \doteq x_1 \cdot x_2) \wedge (z_2 \doteq z_1 \cdot x_1) \wedge (\mathfrak{u} \doteq z_2 \cdot z_1).$$

Notice that every sub-bracketing of $\tilde{\alpha}$ has a corresponding word equation in $\Psi_{\tilde{\alpha}}$.

The decomposition of $\tilde{\alpha}$ is somewhat similar to the *Tseytin transformations*, see [20], which transforms a propositional logic formula into a formula in *Tseytin normal form.*

Our next focus is to study which patterns can be decomposed into an *acyclic* 2FC-CQ.

---

[2] For convenience, we tend use $\tilde{\alpha}$ to denote a bracketing of the pattern $\alpha \in \Xi^+$.

▶ **Definition 4.5** (Join Tree). *A* join tree *for* $\Psi \in \mathsf{2FC\text{-}CQ}$ *with body* $\bigwedge_{i=1}^{n} \chi_i$ *is an undirected tree* $T := (V, E)$*, where* $V := \{\chi_i \mid i \in [n]\}$*, and for all* $\chi_i, \chi_j \in V$*, if* $x \in \mathsf{var}(\chi_i)$ *and* $x \in \mathsf{var}(\chi_j)$*, then* x *appears in all nodes that lie on the path between* χ_i *and* χ_j *in* T*.*

Note that we use χ (with indices) to denote atoms of a 2FC-CQ to distinguish them from word equations with arbitrarily large right-hand sides – which we denote by η (with indices). We call $\Psi \in \mathsf{2FC\text{-}CQ}$ *acyclic* if there exists a join tree for Ψ. Otherwise, we call Ψ *cyclic*.

▶ **Definition 4.6** (Acyclic Patterns). *If* $\Psi_{\tilde{\alpha}} \in \mathsf{2FC\text{-}CQ}$ *is a decomposition of* $\tilde{\alpha} \in \mathsf{BPat}$ *and* $\Psi_{\tilde{\alpha}}$ *is acyclic, then we call* $\tilde{\alpha}$ acyclic*. If* $\Psi_{\tilde{\alpha}}$ *is cyclic, then we call* $\tilde{\alpha}$ cyclic*. If there exists* $\tilde{\alpha} \in \mathsf{BPat}(\alpha)$ *which is acyclic, then we say that* α *is* acyclic*. Otherwise,* α *is* cyclic*.*

When determining whether a decomposition $\Psi_{\tilde{\alpha}} \in \mathsf{2FC\text{-}CQ}$ is acyclic, we treat each word equation (atom) of $\Psi_{\tilde{\alpha}}$ as a single relational symbol. We also consider $\mathfrak{u}$ to be a constant symbol, since $\sigma(\mathfrak{u}) = w$ always holds. This raises the question as to whether every pattern has an acyclic decomposition. The answers is no, as the following result shows.

▶ **Proposition 4.7.** $x_1 x_2 x_1 x_3 x_1$ *is a cyclic pattern, and* $x_1 x_2 x_3 x_1$ *is an acyclic pattern that has a cyclic bracketing.*

This leads to the following question: *Can we decide whether a pattern is acyclic in polynomial time?* Given a pattern $\alpha \in \Xi^+$, we have that $|\mathsf{BPat}(\alpha)| = C_{|\alpha|-1}$, where C_i is the ith *Catalan number*, see [18]. As the Catalan numbers grow exponentially, a straightforward enumeration of bracketings to finding an acyclic bracketing is not enough.

If $\Psi_{\tilde{\alpha}} \in \mathsf{2FC\text{-}CQ}$ is a decomposition of $\tilde{\alpha} \in \mathsf{BPat}(\alpha)$, then we call the variable $x \in \Xi$ which represents the whole pattern the *root variable*. If x is the root variable, then the atom $(x \doteq y \cdot z)$ for some $y, z \in \Xi$, is called the *root atom*. So far, the root variable has always been $\mathfrak{u}$. In Section 5, different root variables will be considered.

Let $\Psi_{\tilde{\alpha}} \in \mathsf{2FC\text{-}CQ}$ be the decomposition of $\tilde{\alpha} \in \mathsf{BPat}(\alpha)$, where $\alpha \in \Xi^+$. We define the *concatenation tree* of $\Psi_{\tilde{\alpha}}$ as a rooted, undirected, binary tree $\mathcal{T} := (\mathcal{V}, \mathcal{E}, <, \Gamma, \tau, v_r)$, where $\mathcal{V}$ is a set of nodes and $\mathcal{E}$ is a set of undirected edges. If v and v' have a shared parent node, then we use $v < v'$ to denote that v is the left child and v' is the right child of their shared parent. We also have $\Gamma := \mathsf{var}(\Psi_{\tilde{\alpha}})$ and the function $\tau \colon \mathcal{V} \to \Gamma$ that *labels* nodes from the concatenation tree with variables from $\mathsf{var}(\Psi_{\tilde{\alpha}})$. We use v_r to denote the root of $\mathcal{T}$. The *concatenation tree* of $\Psi_{\tilde{\alpha}}$ is defined as follows.

▶ **Definition 4.8.** *Let* $\Psi_{\tilde{\alpha}} := \mathsf{Ans}(\vec{x}) \leftarrow \bigwedge_{i=1}^{n} (z_i \doteq x_i \cdot x_i')$ *be a decomposition of* $\tilde{\alpha} \in \mathsf{BPat}(\alpha)$*. We carry out the construction of a concatenation tree in two steps. First, we build a tree recursively. If* $v \in \mathcal{V}$ *is labeled with* z_i *for* $i \in [n]$*, then there exists a left and right child of* v *that are labeled with* x_i *and* x_i' *respectively.*

In the second step, we prune the result of the above construction to remove redundancies. For each set of non-leaf nodes *that share a common label, we define an ordering* $\ll$*. If* $\tau(v_i) = \tau(v_j)$ *and the distance from the root of* $\mathcal{T}$ *to* v_j *is strictly less than the distance from the root to* v_i*, then* $v_j \ll v_i$*. If* $\tau(v_i) = \tau(v_j)$ *and the distance from* v_r *to* v_i *and* v_j *is equal, then* $v_j \ll v_i$ *if and only if* v_j *appears to the* right *of* v_i*. For each set of non-leaf nodes that share a common label, all nodes other than the* $\ll$*-maximum node are called* redundant*. All descendants of redundant nodes are removed.*

Concatenation trees for 2FC-CQs can be understood as a variation of *derivation trees* for straight-line programs [17]. While the pruning may seem somewhat unnatural, the concatenation tree of a decomposition is a useful tool that we shall use in Lemma 4.11 to characterize acyclic bracketings.

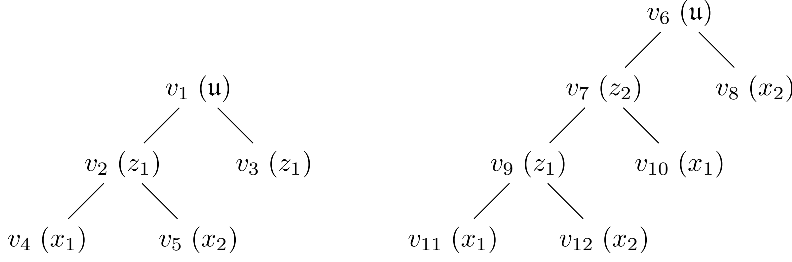

Figure 1 Concatenation trees for the decompositions of $((x_1 \cdot x_2) \cdot (x_1 \cdot x_2))$ and $(((x_1 \cdot x_2) \cdot x_1) \cdot x_2)$. This figure is used to illustrate Example 4.10.

Due to the pruning procedure, every non-leaf node represents a unique sub-bracketing. For every node v with left child v_l and right child v_r, we define $\mathsf{atom}(v) := (\tau(v) \doteq \tau(v_l) \cdot \tau(v_r))$. Note that for any two non-leaf nodes $v, v' \in \mathcal{V}$ where $v \neq v'$, we have that $\mathsf{atom}(v) \neq \mathsf{atom}(v')$. We call $v \in \mathcal{V}$ an x-parent if one of the child nodes of v is labeled x. If v is an x-parent, then $\mathsf{atom}(v)$ must contain the variable x.

▶ **Definition 4.9.** *Let $\Psi_{\tilde{\alpha}} \in \mathsf{2FC\text{-}CQ}$ be the decomposition of $\tilde{\alpha} \in \mathsf{BPat}$ and let $\mathcal{T}$ be the concatenation tree for $\Psi_{\tilde{\alpha}}$. For some $x \in \mathsf{var}(\Psi_{\tilde{\alpha}})$, we say that $\Psi_{\tilde{\alpha}}$ is* x-localized *if all nodes that exist on the path between any two x-parents in $\mathcal{T}$ are also x-parents.*

Since there is exactly one concatenation tree for a decomposition $\Psi_{\tilde{\alpha}} \in \mathsf{2FC\text{-}CQ}$ of $\tilde{\alpha} \in \mathsf{BPat}$, we can say $\Psi_{\tilde{\alpha}}$ is x-localized without referring to the concatenation tree of $\Psi_{\tilde{\alpha}}$.

▶ **Example 4.10.** Consider the pattern $\alpha := x_1 x_2 x_1 x_2$ and the following two bracketings:

$$\tilde{\alpha}_1 := ((x_1 \cdot x_2) \cdot (x_1 \cdot x_2)) \text{ and } \tilde{\alpha}_2 := (((x_1 \cdot x_2) \cdot x_1) \cdot x_2).$$

The bracketing $\tilde{\alpha}_1$ is decomposed into $\Psi_1 := \mathsf{Ans}() \leftarrow (z_1 \doteq x_1 \cdot x_2) \wedge (\mathfrak{u} \doteq z_1 \cdot z_1)$ and $\tilde{\alpha}_2$ is decomposed into $\Psi_2 := \mathsf{Ans}() \leftarrow (z_1 \doteq x_1 \cdot x_2) \wedge (z_2 \doteq z_1 \cdot x_1) \wedge (\mathfrak{u} \doteq z_2 \cdot x_2)$. The concatenation trees for Ψ_1 and Ψ_2 are given in Figure 1. The label for each node is given in parentheses next to the corresponding node. We can see that $\mathsf{atom}(v_2) = (z_1 \doteq x_1 \cdot x_2)$. It follows that Ψ_2 is x_1-localized, but Ψ_2 is not x_2-localized. Observe that $v_3 \ll v_2$, since v_2 appears to the left of v_3. Therefore, v_3 does not have any descendants, since it is a *redundant node*.

Utilizing concatenation trees for the decomposition $\Psi_{\tilde{\alpha}}$ of $\tilde{\alpha} \in \mathsf{BPat}(\alpha)$, and the notion of $\Psi_{\tilde{\alpha}}$ being x-localized for $x \in \mathsf{var}(\Psi_{\tilde{\alpha}})$, we are now able to state sufficient and necessary conditions for $\alpha \in \Xi^+$ to be acyclic.

▶ **Lemma 4.11.** *The decomposition $\Psi_{\tilde{\alpha}} \in \mathsf{2FC\text{-}CQ}$ of $\tilde{\alpha} \in \mathsf{BPat}(\alpha)$ is acyclic if and only if $\Psi_{\tilde{\alpha}}$ is x-localized for every $x \in \mathsf{var}(\Psi_{\tilde{\alpha}})$.*

The proof of the if-direction is rather straightforward: Take the concatenation tree of $\Psi_{\tilde{\alpha}}$, replace each non-leaf node $v \in \mathcal{V}$ with $\mathsf{atom}(v)$, then remove all leaf nodes from the concatenation tree of $\Psi_{\tilde{\alpha}}$. This gives us a join tree for $\Psi_{\tilde{\alpha}}$. The only-if direction for Lemma 4.11 is somewhat more technical. This is because we need to prove this direction for the most general join tree of $\Psi_{\tilde{\alpha}}$. We prove this by contradiction, showing that there does not exist a valid label for certain non-leaf nodes of the concatenation tree if $\Psi_{\tilde{\alpha}}$ is not x-localized for some variable $x \in \mathsf{var}(\Psi_{\tilde{\alpha}})$.

Refering back to Example 4.10, we see that Ψ_2 is not x_2-localized and therefore Ψ_2 is cyclic, whereas we have that Ψ_1 is x-localized for all $x \in \mathsf{var}(\Psi_1)$ and hence Ψ_1 is acyclic.

▶ **Theorem 4.12.** *Whether $\alpha \in \Xi^+$ is acyclic can be decided in time $\mathcal{O}(|\alpha|^7)$.*

We prove Theorem 4.12 by giving a bottom-up algorithm that continuously adds larger acyclic subpatterns of α to a set. To determine whether concatenating two acyclic subpatterns results in a larger acyclic subpattern, we also keep an edge relation and check whether x is localized, see Lemma 4.11. We terminate the algorithm when the edge relation has reached a fixed-point. In the proof of Theorem 4.12, we also show that if α is acyclic, then we can construct a concatenation tree for a decomposition for $\tilde{\alpha} \in \mathsf{BPat}(\alpha)$ in $\mathcal{O}(|\alpha|^7)$ time.

5 Acyclic FC-CQs

In this section, we generalize from decomposing patterns to decomposing FC-CQs. The main result of this section is a polynomial-time algorithm to determine whether an FC-CQ can be decomposed into an acyclic 2FC-CQ. We do this to find a notion of acyclicity for FC-CQs such that the resulting fragment is tractable.

Decomposing a word equation $(x \doteq \alpha)$ where $x \in \Xi$ and $\alpha \in (\Xi \setminus \{x\})^+$ is analogous to decomposing α, but whereas $\mathfrak{u}$ is the root variable when decomposing a pattern, we use x as the root variable when decomposing $(x \doteq \alpha)$.

If every atom of $\varphi \in$ FC-CQ is acyclic, then φ does not necessarily have tractable model checking. If this were the case, then any decomposition $\Psi_{\tilde{\alpha}} \in$ 2FC-CQ of some $\tilde{\alpha} \in \mathsf{BPat}$ would have tractable model checking (because every word equation of the form $z \doteq x \cdot y$ is acyclic). This would imply that the membership problem for patterns can be solved in polynomial time, which contradicts [6], unless P = NP. Furthermore, if we define $\varphi \in$ FC-CQ to be acyclic if there exists a join tree for φ where every word equation is an atom, then model checking for φ is not tractable. To show this, consider $\varphi := \mathsf{Ans}() \leftarrow (\mathfrak{u} \doteq \alpha)$. Model checking for φ is equivalent to the membership problem for α, which is NP-complete [6]. Therefore, we require a more refined notion of acyclicity for FC-CQs.

In Section 4, we studied the decomposition of terminal-free patterns. If φ is an FC-CQ with the body $\mathsf{Ans}(\vec{x}) \leftarrow \bigwedge_{i=1}^{n} \eta_i$, then the right-hand side of some η_i may not be terminal-free. Therefore, before defining the decomposition of FC[REG]-CQs, we define a way to *normalize* FC[REG]-CQs in order to better utilize the techniques of Section 4.

▶ **Definition 5.1.** *We call an* FC-CQ *with body $\bigwedge_{i=1}^{n}(x_i \doteq \alpha_i)$* normalized *if for all $i, j \in [n]$, we have $\alpha_i \in \Xi^+$, $x_i \notin \mathsf{var}(\alpha_i)$, $\mathfrak{u} \notin \mathsf{var}(\alpha_i)$, and $\alpha_i = \alpha_j$ if and only if $i = j$.*

An FC[REG]-CQ *with body $\bigwedge_{i=1}^{n}(x_i \doteq \alpha_i) \wedge \bigwedge_{j=1}^{m}(y_j \,\dot{\in}\, \gamma)$ is* normalized *if the subformula $\bigwedge_{i=1}^{n}(x_i \doteq \alpha_i)$ is normalized.*

Since we are interested in polynomial time algorithms, the following lemma allows us to assume that all FC-CQs are normalized without affecting any claims about complexity.

▶ **Lemma 5.2.** *Given $\varphi \in$ FC[REG]-CQ, we can construct an equivalent, normalized* FC[REG]-CQ *in time $\mathcal{O}(|\varphi|^2)$.*

To prove Lemma 5.2 we use a simple re-writing procedure. We replace every terminal factor in our formula with a new variable, and use a regular constraint to determine which terminal word that variable represents. If σ is a morphism that satisfies $(x \doteq \alpha)$ for some $\alpha \in \Xi$, then $|\sigma(x)| = |\sigma(\alpha)|$. Therefore, if $x \in \alpha$, then $|\sigma(x)| = |\sigma(\alpha_1)| + |\sigma(x)| + |\sigma(\alpha_2)|$ where $\alpha = \alpha_1 \cdot x \cdot \alpha_2$. We can then determine that $\sigma(\alpha_1) \cdot \sigma(\alpha_2) = \varepsilon$. Hence, $x \doteq \alpha$ can be replaced with $(x \doteq y) \wedge \bigwedge_{z \in \mathsf{var}(\alpha_1 \cdot \alpha_2)}(z \,\dot{\in}\, \varepsilon)$ where y is a new and unique variable. An analogous method is used if $\mathfrak{u} \in \mathsf{var}(\alpha)$.

▶ **Example 5.3.** We define an FC[REG]-CQ along with an equivalent normalized FC[REG]-CQ:

$$\varphi := \mathsf{Ans}(\vec{x}) \leftarrow (x_1 \doteq x_2 \cdot \mathfrak{u} \cdot x_2) \wedge (x_4 \doteq x_4) \wedge (x_3 \doteq \mathtt{aab}),$$
$$\varphi' := \mathsf{Ans}(\vec{x}) \leftarrow (\mathfrak{u} \doteq x_1) \wedge (x_2 \mathrel{\dot\in} \varepsilon) \wedge (x_4 \doteq z_2) \wedge (x_3 \doteq z_1) \wedge (z_1 \mathrel{\dot\in} \mathtt{aab}).$$

We now generalize the process of decomposing patterns to decomposing FC-CQs. For every FC-CQ $\varphi := \mathsf{Ans}(\vec{x}) \leftarrow \bigwedge_{i=1}^{n} \eta_i$, we say that a 2FC-CQ $\Psi_\varphi := \mathsf{Ans}(\vec{x}) \leftarrow \bigwedge_{i=1}^{n} \Psi_i$ is a *decomposition* of φ if every Ψ_i is a decomposition of η_i and, for all $i, j \in [n]$ with $i \neq j$, the sets of introduced variables for Ψ_i and Ψ_j are disjoint.

▶ **Example 5.4.** Let $\varphi \in$ FC-CQ be defined as follows:

$$\varphi := \mathsf{Ans}(\vec{x}) \leftarrow (x_1 \doteq y_1 \cdot y_2 \cdot y_3) \wedge (x_2 \doteq y_2 \cdot y_3 \cdot y_3 \cdot y_4).$$

We now consider the following decompositions for each word equation of φ:

$$\Psi_1 := (x_1 \doteq y_1 \cdot z_1) \wedge (z_1 \doteq y_2 \cdot y_3), \text{ and } \Psi_2 := (x_2 \doteq z_2 \cdot y_4) \wedge (z_2 \doteq z_3 \cdot y_3) \wedge (z_3 \doteq y_2 \cdot y_3).$$

Therefore, $\Psi_\varphi := \mathsf{Ans}(\vec{x}) \leftarrow \Psi_1 \wedge \Psi_2$ is a decomposition of φ.

▶ **Definition 5.5** (Acyclic FC-CQs). *If $\Psi_\varphi \in$ 2FC-CQ is a decomposition of $\varphi \in$ FC-CQ, we say that Ψ_φ is* acyclic *if there exists a join tree for Ψ_φ. Otherwise, Ψ_φ is* cyclic. *If there exists an acyclic decomposition of φ, then we say that φ is* acyclic. *Otherwise, φ is* cyclic.

Recall that, since $\mathfrak{u}$ is always mapped to w, we can consider $\mathfrak{u}$ a constant symbol. Therefore, if $T := (V, E)$ is a join tree for some decomposition of φ, then there can exist two nodes that both contain $\mathfrak{u}$, yet it is not necessary for all nodes on the path between these two nodes to also contain $\mathfrak{u}$. Referring back to Example 5.4, we can see that φ is acyclic by executing the GYO algorithm on the decomposition (see Chapter 6 of [1] for more information on acyclic joins). Our next focus is to study which FC-CQs are acyclic, and which are not.

▶ **Lemma 5.6.** *If $\Psi_\varphi \in$ 2FC-CQ is a decomposition of $\varphi := \mathsf{Ans}(\vec{x}) \leftarrow \bigwedge_{i=1}^{n} \eta_i$, and we have a join tree $T := (V, E)$ for Ψ_φ, then we can partition T into $T^1, T^2, \dots T^n$ such that for each $i \in [n]$, we have that T^i is a join tree for a decomposition of η_i.*

To prove Lemma 5.6, we consider a join tree $T := (V, E)$ for the acyclic decomposition $\Psi_\varphi \in$ 2FC-CQ of $\varphi \in$ FC-CQ, along with the induced subgraph of T on the set of atoms for a decomposition of a single atom of φ. We show that this subgraph is connected, and since the introduced variables are disjoint for separate atoms of φ, this forms a partition on T.

Let $\varphi := \mathsf{Ans}(\vec{x}) \leftarrow \bigwedge_{i=1}^{n} \eta_i$ be a normalized FC-CQ. A join tree $T := (V, E)$ for φ where $V = \{\eta_i \mid i \in [n]\}$ is called a *weak join tree.* If there exists a weak join tree for φ, then we say that φ is *weakly acyclic.* Otherwise, φ is *weakly cyclic.* Clearly weak acyclicity is not sufficient for tractability, as discussed at the start of the current section.

▶ **Example 5.7.** Consider the following normalized FC-CQ:

$$\varphi := \mathsf{Ans}(\vec{x}) \leftarrow (\mathfrak{u} \doteq x_1 \cdot x_2 \cdot x_1 \cdot x_3 \cdot x_1) \wedge (x_1 \doteq x_4 \cdot x_5 \cdot x_5) \wedge (x_6 \doteq x_7 \cdot x_7 \cdot x_7).$$

Using the GYO algorithm, we can see that φ is weakly acyclic.

Let $\varphi := \mathsf{Ans}(\vec{x}) \leftarrow \bigwedge_{i=1}^{n} \eta_i$ be an FC-CQ, and let Ψ_φ be an acyclic decomposition of φ. If $T := (V, E)$ is a join tree of Ψ_φ, then for each $i \in [n]$, we use $T^i := (V^i, E^i)$ to denote the subtree of T that is a join tree for the decomposition of η_i. We know that T^i and T^j are disjoint for all $i, j \in [n]$ where $i \neq j$, see Lemma 5.6.

▶ **Lemma 5.8.** *Let* $\varphi := \mathsf{Ans}(\vec{x}) \leftarrow \bigwedge_{i=1}^{n} \eta_i$ *be a normalized* FC-CQ*. If any of the following conditions holds, then* φ *is cyclic:*

1. φ *is weakly cyclic,*
2. η_i *is cyclic for any* $i \in [n]$*,*
3. $|\mathsf{var}(\eta_i) \cap \mathsf{var}(\eta_j)| > 3$ *for any* $i, j \in [n]$ *where* $i \neq j$*, or*
4. $|\mathsf{var}(\eta_i) \cap \mathsf{var}(\eta_j)| = 3$*, and* $|\eta_i| > 3$ *or* $|\eta_j| > 3$ *for any* $i, j \in [n]$ *where* $i \neq j$*.*

Condition 1 can be proven by simply replacing T^i with a single node η_i for all $i \in [n]$. Condition 2 follows directly from Lemma 5.6. Conditions 3 and 4 can be proven by a contradiction: Consider the shortest path from any atom of the decomposition of η_i to any atom of the decomposition of η_j. Since the end points of these paths cannot contain all the variables that η_i and η_j share, it follows that $T := (V, E)$ is not a join tree.

While Conditions 3 and 4 might seem strict, we can pre-factor common subpatterns. For example, the conjunction $(x_1 \doteq \alpha_1 \cdot \alpha_2 \cdot \alpha_3) \wedge (x_2 \doteq \alpha_4 \cdot \alpha_2 \cdot \alpha_5)$, where $\alpha_i \in \Xi^+$ for $i \in [5]$, can be written as $(x_1 \doteq \alpha_1 \cdot z \cdot \alpha_3) \wedge (x_2 \doteq \alpha_4 \cdot z \cdot \alpha_5) \wedge (z \doteq \alpha_2)$ where $z \in \Xi$ is a new variable. We illustrate this further in the following example.

▶ **Example 5.9.** Consider the following FC-CQ:

$$\varphi := \mathsf{Ans}() \leftarrow (x_1 \doteq y_1 \cdot y_2 \cdot y_3 \cdot y_4 \cdot y_5) \wedge (x_2 \doteq y_6 \cdot y_2 \cdot y_3 \cdot y_4 \cdot y_5).$$

Using Lemma 5.8, we can see that φ is cyclic. However, since the right-hand side of the two word equations share a common subpattern, we can rewrite φ as

$$\varphi' := \mathsf{Ans}() \leftarrow (x_1 \doteq y_1 \cdot z) \wedge (x_2 \doteq y_6 \cdot z) \wedge (z \doteq y_2 \cdot y_3 \cdot y_4 \cdot y_5).$$

One could alter our definition of FC-CQ decomposition so that if two atoms share a bracketing, then the bracketing is replaced with the same variable (analogously to how decompositions are defined on patterns). The authors believe it is likely that such a definition of FC-CQ decomposition is equivalent to our definition of FC-CQ decomposition after "factoring out" common subpatterns between atoms.

Our next consideration is how the structure of a join tree for a decomposition of an acyclic query $\varphi \in$ FC[REG]-CQ relates to the structure of a weak join tree for φ.

▶ **Definition 5.10** (Skeleton Tree)**.** *Let* $\Psi_\varphi \in$ 2FC-CQ *be an acyclic decomposition of the query* $\varphi := \mathsf{Ans}(\vec{x}) \leftarrow \bigwedge_{i=1}^{n} \eta_i$*, and let* $T := (V, E)$ *be a join tree for* Ψ_φ*. We say that a weak join tree* $T_w := (V_w, E_w)$ *is the* skeleton tree *of* T *if there exists an edge in* E *from a node in* V^i *to a node in* V^j *if and only if* $\{\eta_i, \eta_j\} \in E_w$*.*

In the proof of Lemma 5.8 (Condition 1), we show that every join tree for a decomposition has a corresponding skeleton tree. We shall leverage the fact that every join tree of a decomposition of an acyclic FC[REG]-CQ has a skeleton tree in the algorithm given in the proof of Theorem 5.14.

▶ **Example 5.11.** We define $\varphi \in$ FC-CQ and a decomposition Ψ_φ as follows:

$$\varphi := \mathsf{Ans}(\vec{x}) \leftarrow (x_1 \doteq x_2 \cdot x_3 \cdot x_2) \wedge (x_2 \doteq x_4 \cdot x_4 \cdot x_5),$$
$$\Psi_\varphi := \mathsf{Ans}(\vec{x}) \leftarrow (x_1 \doteq x_2 \cdot z_1) \wedge (z_1 \doteq x_3 \cdot x_2) \wedge (x_2 \doteq z_2 \cdot x_5) \wedge (z_2 \doteq x_4 \cdot x_4).$$

The skeleton tree along with the join tree of Ψ_φ are given in Figure 2.

One might assume that some skeleton trees are more "desirable" than others in terms of using it for finding an acyclic decomposition of an FC[REG]-CQ. However, as we observe next, any skeleton tree is sufficient.

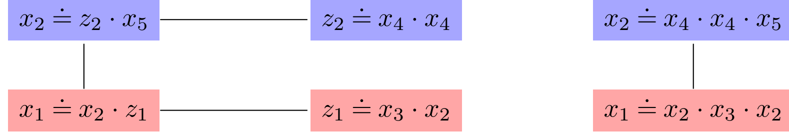

Figure 2 The join tree (left) and the skeleton tree of the join tree (right) for Example 5.11.

▶ **Lemma 5.12.** *Let $\Psi_\varphi \in$ 2FC-CQ be a decomposition of $\varphi \in$ FC-CQ. If Ψ_φ is acyclic, then any weak join tree can be used as the skeleton tree.*

Given a weak join tree of an acyclic query φ, the proof of Lemma 5.12 transforms the join tree of Ψ_φ so that the resulting join tree has the given weak join tree as its skeleton tree. Thus, we can use any weak join tree as a "template" for the eventual join tree of the decomposition (under the assumption that the query is acyclic).

While Lemma 5.8 and Lemma 5.12 give some insights and necessary conditions for deciding whether $\varphi \in$ FC-CQ is acyclic, these conditions are not sufficient. We therefore give the following lemma which is needed in the proof of Theorem 5.14 to find an acyclic decomposition of φ.

▶ **Lemma 5.13.** *Given a normalized* FC-CQ *of the form $\varphi := \mathsf{Ans}(\vec{x}) \leftarrow (z \doteq \alpha)$ and a set $C \subseteq \{\{x,y\} \mid x,y \in \mathsf{var}(z \doteq \alpha)$ and $x \neq y\}$, we can decide whether there is an acyclic decomposition $\Psi \in$ 2FC-CQ of φ such that for every $\{x,y\} \in C$, there is an atom of Ψ that contains both x and y in time $\mathcal{O}(|\alpha|^7)$.*

We prove Lemma 5.13 using a variant of the algorithm given in the proof of Theorem 4.12. The purposes of Lemma 5.13 should become clearer after giving the following necessary and sufficient criteria for an FC[REG]-CQ to be acyclic: Let $\varphi := \bigwedge_{i=1}^{m}(x_i \doteq \alpha_i) \wedge \bigwedge_{j=1}^{n}(y_j \mathrel{\dot\in} \gamma_j)$ be a normalized FC[REG]-CQ. Then, there exists an acyclic decomposition $\Psi \in$ 2FC[REG]-CQ of φ if and only if the following conditions hold:

1. φ is weakly acyclic,
2. for all $i \in [m]$ the pattern α_i is acyclic, and
3. for every $i \in [m]$, there is a decomposition Ψ_i of $x_i \doteq \alpha_i$ such that for all $j \in [m] \setminus \{i\}$ there is a decomposition Ψ_j of $x_j \doteq \alpha_j$ where there exists an atom χ_i of Ψ_i and an atom χ_j of Ψ_j that satisfies $\mathsf{var}(\chi_i) \cap \mathsf{var}(\chi_j) = \mathsf{var}(x_i \doteq \alpha_i) \cap \mathsf{var}(x_j \doteq \alpha_j)$.

We are now ready to give the main result of the paper.

▶ **Theorem 5.14.** *Whether $\varphi \in$ FC[REG]-CQ is acyclic can be decided in time $\mathcal{O}(|\varphi|^8)$.*

To prove Theorem 5.14, we first check whether $\varphi \in$ FC-CQ has any of the conditions from Lemma 5.8. If so, then we know that φ is cyclic. Then, we construct a weak join tree for φ. If there is an edge $\{\eta_i, \eta_j\}$ of the weak join tree such that η_i and η_j share exactly two variables, then we use Lemma 5.13 to decompose η_i and η_j such that there is an atom of the decomposition (of η_i and η_j), which contains the variables that η_i and η_j share. In the full proof, we show that if such decompositions do not exist, then φ is cyclic. For all other atoms of φ we can use any decomposition. The resulting acyclic decomposition is the conjunction of the decompositions of each atom. The proof of Theorem 5.14 also shows that if φ is acyclic, an acyclic decomposition can be constructed in polynomial time.

▶ **Example 5.15.** We revisit the FC[REG]-CQ that was given in the introduction:

$$\varphi := \mathsf{Ans}(x,y) \leftarrow (z \doteq z_2 \cdot x \cdot z_3 \cdot x \cdot z_4) \wedge (z \doteq z_5 \cdot y \cdot z_6) \wedge (z \mathrel{\dot\in} \gamma_{\mathsf{sen}}) \wedge (x \mathrel{\dot\in} \gamma_{\mathsf{prod}}) \wedge (y \mathrel{\dot\in} \gamma_{\mathsf{pos}}).$$

We can see this is acyclic by considering the following decomposition:

$$\Psi := \mathsf{Ans}(x,y) \leftarrow (y_1 \doteq x \cdot z_3) \wedge (y_2 \doteq y_1 \cdot x) \wedge (y_3 \doteq z_2 \cdot y_2) \wedge (z \doteq y_3 \cdot z_4) \\ \wedge (y_4 \doteq z_5 \cdot y) \wedge (z \doteq y_4 \cdot z_6) \wedge (z \mathrel{\dot\in} \gamma_{\mathsf{sen}}) \wedge (x \mathrel{\dot\in} \gamma_{\mathsf{prod}}) \wedge (y \mathrel{\dot\in} \gamma_{\mathsf{pos}}).$$

Due to the small width of the tables that each word equation of the form $(x \doteq y \cdot z)$ produces, we conclude the following:

▶ **Proposition 5.16.** *If $\Psi \in$ 2FC[REG]-CQ is acyclic, then:*
1. *Given $w \in \Sigma^*$, the model checking problem can be solved in time $\mathcal{O}(|\Psi|^2|w|^3)$.*
2. *Given $w \in \Sigma^*$, we can enumerate $[\![\Psi]\!](w)$ with $\mathcal{O}(|\Psi|^2|w|^3)$ delay.*

For FC[REG]-CQs, we first find an acyclic decomposition $\Psi_\varphi \in$ 2FC[REG]-CQ of φ in $\mathcal{O}(|\varphi|^7)$. Then, the upper bound for model checking follows from [15]. Polynomial-delay enumeration follows from [3], where it was proven that given an acyclic (relational) conjunctive query ψ and a database D, we can enumerate $\psi(D)$ with $\mathcal{O}(|\psi||D|)$ delay. Our "database" is of size $\mathcal{O}(|\varphi| \cdot |w|^3)$ as each atom of the form $(z \doteq x \cdot y)$ defines a relation of size $\mathcal{O}(|w|^3)$.

Considering techniques from [3], it may seem that the results of an acyclic FC[REG]-CQ without projections can be enumerated with constant-delay after polynomial time preprocessing. However this is not the case. New variables, that are not free, are introduced in the decomposition of φ and therefore the resulting 2FC[REG]-CQ may not be free-connex, which is required for the results of a CQ to be enumerated with constant-delay [3].

From FC[REG]-CQs to SERCQs. Combining Lemma 3.6 and Proposition 5.16 gives us a class of SERCQs for which model checking can be solved in polynomial-time, and we can enumerate results with polynomial-delay. The hardness of deciding semantic acyclicity (whether a given SERCQ can be realized by an acyclic FC[REG]-CQ) remains open. The authors believe that semantic acyclicity for SERCQs is undecidable, partly due to the fact that various minimization problems are undecidable for FC [11, 14]. For now, all we have are sufficient critiera for a SERCQ to be realized by an acyclic FC[REG]-CQ.

▶ **Definition 5.17.** *We say that a query of the form $P := \pi_Y\big(\zeta^{=}_{x_1,y_1} \cdots \zeta^{=}_{x_k,y_k}(\gamma_1 \bowtie \cdots \bowtie \gamma_n)\big)$ is* pseudo-acyclic *if for every $i \in [n]$, we have that $\gamma_i := \beta_{i_1} \cdot x_i\{\beta_{i_2}\} \cdot \beta_{i_3}$ where $x_i \in \Xi$, and where β_{i_1}, β_{i_2}, and β_{i_3} are regular expressions.*

We now show that Definition 5.17 gives sufficient criteria for an SERCQ to be realized by an acyclic FC[REG]-CQ.

▶ **Proposition 5.18.** *Given a pseudo-acyclic* SERCQ *query, we can construct in polynomial time an acyclic* FC[REG]-CQ *that realizes P.*

Freydenberger et al. [13] proved that fixing the number of atoms and the number of string equalities in a SERCQ allows for polynomial-delay enumeration of results. In contrast to this, Proposition 5.18 allows an unbounded number of joins and string equality selection operators. However, in order to have this tractability result, the expressive power of each regex formula is restricted to only allow one variable. While Proposition 5.18 gives sufficient criteria for a SERCQ to be represented by an acyclic FC[REG]-CQ, many other such classes of SERCQs likely exist. Research into finding large classes of SERCQs that map to acyclic FC[REG]-CQs seems like a promising direction for future work.

6 A Note on k-ary Decompositions

We now generalize the notion of pattern decomposition so that the length of the right-hand side of the resulting formula is less than or equal to some $k \geq 2$. While the binary decompositions might be considered the natural case, we show that generalizing to higher arities increases the expressive power of acyclic patterns. By $k\mathsf{FC\text{-}CQ}$ we denote the set of $\mathsf{FC\text{-}CQ}$ formulas that have a right-hand side of at most length k. We write BPat_k for the set of k-ary bracketed patterns over Ξ. We define BPat_k formally using the following recursive definition: For all $x \in \Xi$ we have that $x \in \mathsf{BPat}_k$, and if $\alpha_1, \alpha_2, \ldots, \alpha_i \in \mathsf{BPat}_k$ where $i \leq k$, then $(\tilde{\alpha}_1 \cdot \tilde{\alpha}_2 \cdots \tilde{\alpha}_i) \in \mathsf{BPat}_k$. We write $\tilde{\alpha} \in \mathsf{BPat}_k(\alpha)$ for some $\alpha \in \Xi^+$ if the underlying, unbracketed pattern of $\tilde{\alpha}$ is α. We can convert $\tilde{\alpha} \in \mathsf{BPat}_k$ into an equivalent $k\mathsf{FC\text{-}CQ}$ analogously to the binary case, see Definition 4.3.

▶ **Example 6.1.** Consider the following 4-ary bracketing:

$$\tilde{\alpha} := (((x_1 \cdot x_2 \cdot x_3) \cdot (x_4 \cdot x_2 \cdot x_4) \cdot (x_1 \cdot x_2) \cdot (x_5 \cdot x_5)) \cdot x_2).$$

As with the 2-ary case, we decompose $\tilde{\alpha}$ to get the following $4\mathsf{FC\text{-}CQ}$:

$$\Psi_{\tilde{\alpha}} := \mathsf{Ans}() \leftarrow (z_1 \doteq x_1 \cdot x_2) \wedge (z_2 \doteq x_5 \cdot x_5) \wedge (z_3 \doteq x_4 \cdot x_2 \cdot x_4) \\ \wedge (z_4 \doteq x_1 \cdot x_2 \cdot x_3) \wedge (z_5 \doteq z_4 \cdot z_3 \cdot z_1 \cdot z_2) \wedge (\mathfrak{u} \doteq z_5 \cdot x_2).$$

The definition of k-ary concatenation tree for a decomposition $\Psi_{\tilde{\alpha}} \in k\mathsf{FC\text{-}CQ}$ of $\tilde{\alpha} \in \mathsf{BPat}_k$ follows analogously to the concatenation trees for 2-ary decompositions, see Definition 4.8. The concatenation tree of the decomposition $\Psi_{\tilde{\alpha}} \in k\mathsf{FC\text{-}CQ}$ is a rooted, labeled, undirected tree $\mathcal{T} := (\mathcal{V}, \mathcal{E}, <, \Gamma, \tau, v_r)$, where $\mathcal{V}$ is the set of nodes, the relation $\mathcal{E}$ is the edge relation, and $<$ is used to denote the order of children of a node (from left to right). We have that $\Gamma := \mathsf{var}(\Psi_{\tilde{\alpha}})$ is the alphabet of labels and $\tau \colon \mathcal{V} \to \Gamma$ is the labeling function. The semantics of a k-ary concatenation tree are defined by considering the natural generalization of Definition 4.8. We say that $\Psi_{\tilde{\alpha}}$ is *x-localized* if all nodes which exist on a path between two x-parents (of $\mathcal{T}$) are also x-parents.

▶ **Proposition 6.2.** *There exists $\tilde{\alpha} \in \mathsf{BPat}_3$ such that the decomposition $\Psi \in 3\mathsf{FC\text{-}CQ}$ of $\tilde{\alpha}$ is acyclic, but there exists $x \in \mathsf{var}(\Psi)$ such that Ψ is not x-localized.*

Proof. Consider $\tilde{\alpha} := ((x_3 \cdot x_3) \cdot ((x_3 \cdot x_3) \cdot x_2) \cdot (x_1 \cdot ((x_3 \cdot x_3) \cdot x_2)))$. The bracketing $\tilde{\alpha}$ is decomposed into $\Psi_{\tilde{\alpha}} \in 3\mathsf{FC\text{-}CQ}$, which is defined as

$$\Psi_{\tilde{\alpha}} := \mathsf{Ans}() \leftarrow (z_1 \doteq x_3 \cdot x_3) \wedge (z_2 \doteq z_1 \cdot x_2) \wedge (z_3 \doteq x_1 \cdot z_2) \wedge (\mathfrak{u} \doteq z_1 \cdot z_2 \cdot z_3).$$

The formula $\Psi_{\tilde{\alpha}}$ can be verified to be acyclic. However, $\Psi_{\tilde{\alpha}}$ is not z_1-localized. ◀

In this section, we have briefly examined k-ary decompositions, and have shown that there exists $\tilde{\alpha} \in \mathsf{BPat}_3$ such that the decomposition $\Psi \in 3\mathsf{FC\text{-}CQ}$ of $\tilde{\alpha}$ is acyclic, but Ψ is not x-localized for some $x \in \mathsf{var}(\Psi)$. The authors note that the if-direction in the proof of Lemma 4.11 implies that x-locality for all variables is a sufficient criterion for a k-ary decomposition to be acyclic. A systematic study into k-ary acyclic decompositions may yield more expressive spanners, and could be useful for pattern languages, which have been linked to FC-formulas with bounded width [14]. However, more general approaches such as bounded treewidth for binary decompositions appear to be a more promising direction for future work. Furthermore, the membership problem for a pattern α parameterized by $|\alpha|$ is $\mathsf{W}[1]$-hard [8]. Since every pattern is trivially $|\alpha|$-ary acyclic, the authors believe it to be likely that the parameterized problem of model checking for k-ary acyclic decompositions is $\mathsf{W}[1]$-hard.

7 Conclusions

Freydenberger and Peterfreund [14] introduced FC[REG] as a logic for querying and model checking words that behaves similar to relational FO. The present paper develops this connection further by providing a polynomial-time algorithm that either decomposes an FC[REG]-CQ into an acyclic 2FC[REG]-CQ, or determines that this is not possible. These acyclic 2FC[REG]-CQ formulas allow for polynomial-time model checking, and their results can be enumerated with polynomial-delay. Consequently, the present paper establishes a notion of tractable acyclicity for FC-CQs. Due to the close connections between FC[REG] and core spanners, this provides us with a large class of tractable SERCQs.

But this is only the first step in the study of tractable SERCQs and FC[REG]-CQs. It seems likely that more efficient algorithms for model checking and enumeration can be found by utilizing string algorithms rather than materializing the relations for each atom.

Another future direction for research is the consideration of other structural parameters, like treewidth. A systematic study of the decomposition of FC-CQs into 2FC-CQs of bounded treewidth would likely yield a large class of FC-CQs with polynomial-time model checking. As a consequence, one could define a suitable notion of treewidth for core spanners. Determining the exact class of FC-CQs with polynomial-time model checking is likely a hard problem. This is because such a result would solve the open problem in formal languages of determining exactly what patterns have polynomial-time membership.

References

1. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*, volume 8. Addison-Wesley Reading, 1995.
2. Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-delay enumeration for nondeterministic document spanners. *ACM SIGMOD Record*, 49(1):25–32, 2020.
3. Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. On acyclic conjunctive queries and constant delay enumeration. In *Proceedings of CSL 2007*, pages 208–222, 2007.
4. Joachim Bremer and Dominik D. Freydenberger. Inclusion problems for patterns with a bounded number of variables. *Information and Computation*, 220:15–43, 2012.
5. Stefan Burkhardt, Juha Kärkkäinen, and Peter Sanders. Linear work suffix array construction. *Journal of the ACM*, 53(6):918–936, 2006.
6. Andrzej Ehrenfreucht and Grzegorz Rozenberg. Finding a homomorphism between two words is NP-complete. *Information Processing Letters*, 9(2):86–88, 1979.
7. Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Document spanners: A formal approach to information extraction. *Journal of the ACM*, 62(2):12, 2015.
8. Henning Fernau, Markus L Schmid, and Yngve Villanger. On the parameterised complexity of string morphism problems. *Theory of Computing Systems*, 59:24–51, 2016.
9. Fernando Florenzano, Cristian Riveros, Martín Ugarte, Stijn Vansummeren, and Domagoj Vrgoc. Constant delay algorithms for regular document spanners. In *Proceedings of PODS 2018*, pages 165–177, 2018.
10. Dominik D. Freydenberger. A logic for document spanners. *Theory of Computing Systems*, 63(7):1679–1754, 2019.
11. Dominik D. Freydenberger and Mario Holldack. Document spanners: From expressive power to decision problems. *Theory of Computing Systems*, 62(4):854–898, 2018.
12. Dominik D. Freydenberger, Benny Kimelfeld, Markus Kröll, and Liat Peterfreund. Complexity bounds for relational algebra over document spanners. In *Proceedings of PODS 2019*, pages 320–334, 2019.

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

14 Dominik D. Freydenberger and Liat Peterfreund. The theory of concatenation over finite models. In *Proceedings of ICALP 2021*, pages 130:1–130:17, 2021.

15 Georg Gottlob, Nicola Leone, and Francesco Scarcello. The complexity of acyclic conjunctive queries. *Journal of the ACM*, 48(3):431–498, 2001.

16 Dan Gusfield. *Algorithms on Strings, Trees, and Sequences – Computer Science and Computational Biology*. Cambridge University Press, 1997.

17 Markus Lohrey. Algorithmics on SLP-compressed strings: A survey. *Groups-Complexity-Cryptology*, 4(2):241–299, 2012.

18 Gloria Olive. Catalan numbers revisited. *Journal of mathematical analysis and applications*, 111(1):201–235, 1985.

19 Liat Peterfreund. Grammars for document spanners. In *Proceedings of ICDT 2021*, pages 7:1–7:18, 2021.

20 Steven David Prestwich. CNF encodings. *Handbook of satisfiability*, 185:75–97, 2009.

21 Markus L. Schmid and Nicole Schweikardt. A purely regular approach to non-regular core spanners. In *Proceedings of ICDT 2021*, pages 4:1–4:19, 2021.

22 Mihalis Yannakakis. Algorithms for acyclic database schemes. In *Proceedings of VLDB 1981*, pages 82–94, 1981.