The Theory of Concatenation over Finite Models

Dominik D. Freydenberger
Loughborough University, UK

Liat Peterfreund
DI ENS, ENS Paris, CNRS, PSL University, INRIA, France

Abstract

We propose FC, a new logic on words that combines finite model theory with the theory of concatenation – a first-order logic that is based on word equations. Like the theory of concatenation, FC is built around word equations; in contrast to it, its semantics are defined to only allow finite models, by limiting the universe to a word and all its factors. As a consequence of this, FC has many of the desirable properties of FO on finite models, while being far more expressive than FO$[<]$. Most noteworthy among these desirable properties are sufficient criteria for efficient model checking, and capturing various complexity classes by adding operators for transitive closures or fixed points.

Not only does FC allow us to obtain new insights and techniques for expressive power and efficient evaluation of document spanners, but it also provides a general framework for logic on words that also has potential applications in other areas.

2012 ACM Subject Classification Theory of computation → Database query languages (principles); Theory of computation → Logic and databases; Theory of computation → Finite Model Theory

Keywords and phrases finite model theory, word equations, descriptive complexity, model checking, document spanners

Digital Object Identifier 10.4230/LIPIcs.ICALP.2021.130

Category Track B: Automata, Logic, Semantics, and Theory of Programming

Related Version *Full Version*: `https://arxiv.org/abs/1912.06110`

Funding *Dominik D. Freydenberger*: Supported by EPSRC grant EP/T033762/1.
Liat Peterfreund: Supported by Fondation des Sciences Mathématiques de Paris (FSMP). A part of this work was done while affiliated with IRIF, CNRS, Université de Paris and with Edinburgh University.

Acknowledgements The authors thank Joel D. Day, Manfred Kufleitner, Leonid Libkin, Sam M. Thompson, and the reviewers of the current and previous versions for their helpful comments.

1 Introduction

This paper proposes a finite model version of the theory of concatenation: FC, a new logic that is designed to describe properties of words and to query them. While the idea of using logic on words is by no means new, the advantage of FC is its combination of expressive power and tractable model checking and evaluation.

Logic on words. A common way of using logic on words is *monadic second-order logic* (MSO) over a linear order (e. g. [53]). That is, a word w is seen as a sequence of positions, and predicates $\mathsf{P}_a(x)$ express "letter a at position x of w". This approach comes with two disadvantages for querying. The first is that factors (continuous subwords) cannot be expressed directly. Consider the query "return all factors of w". As variables refer to positions, the query would not return a factor u directly, but represent it as a set (or tuple) of positions that describe a specific occurrence of u in w. If u occurs more than once, the query result would contain each occurrence – unless the logic is powerful enough to prevent this.

This leads us to the second disadvantage, namely that MSO cannot compare factors of unbounded length. That is, while MSO can express queries like "return the positions of factors of length k that occur twice in w" for a fixed length k, it is impossible to express "return the positions of factors that occur twice in w"; or non-regular languages, like that of all words ww with $w \in \{\mathtt{a}, \mathtt{b}\}^*$. As a result, many natural relations on factors of w are inexpressible in MSO, in particular the concatenation $x = yz$.

Another approach to logic on words is the *theory of concatenation* (short: C). First defined by Quine [48], this logic is built on *word equations*, that is, equations of the form $xx \doteq yyy$, where variables like x and y stand for words over a finite alphabet Σ. While less prominent than FO or MSO, the theory of concatenation has been studied extensively since the 1970s, with particular emphasis on word equations. A fairly recent survey on the satisfiability of word equations is [14]. More current research on word equations and the theory of concatenation can be found in e. g. [7, 10, 11, 12, 44, 50].

In contrast to MSO, the logic C allows us to treat words as words (instead of intervals of positions) and a position in w can be expressed as the corresponding prefix of w. More importantly, C can express properties like "u is a factor of v" or concatenation like $x = yz$. This expressive power comes at a price – even limited use of negation leads to an undecidable theory (i. e., satisfiability is undecidable, see [17, 48]). Contrast this to *first-order logic* (FO) over finite models: By Trakhtenbrot's theorem, satisfiability is undecidable; but the model checking problem is not just decidable, but can even be made tractable (see e. g. [18, 36]).

Another situation where using queries for words together with an open universe causes problems occurs in *string databases*, see [4, 5, 27, 28]. These query languages treat words as entries of the database instead of operating on a single word. Furthermore, they offer transformation operations that assume an infinite universe. As a result, these query language usually express Turing-complete functions from words to words.

Introducing FC. The new logic FC aims to bring the advantages of FO on finite models to the theory of concatenation. The universe for C is usually assumed to be Σ^*, which means that there is no meaningful distinction between satisfiability and model checking. The key idea of FC is changing universe from Σ^* to the set of all factors of a word w; comparable to how the universe for MSO consists of all positions of a word w.

As FC-formulas are based on word equations, concatenation is straightforward to use. For example, "return all factors that occur twice in w" can be expressed as

$$\varphi_1(x) := \exists p_1, p_2, s_1, s_2 \colon \big(\mathfrak{u} \doteq p_1\, x\, s_1 \land \mathfrak{u} \doteq p_2\, x\, s_2 \land \neg p_1 \doteq p_2\big),$$

where $\mathfrak{u}$ represents w. In detail, φ_1 expresses that there are two different ways of decomposing w into $w = p\, x\, s$. If we also wanted to know the positions of these occurrences, we could return p_1 and p_2 (by removing their quantifiers), as these encode the start of each occurrence in w. This formula does not rely on the requirement that variables can only be mapped to factors of w, as the $\mathfrak{u}$ on the left side of the equations ensures this already. Instead, consider

$$\varphi_2(x) := \exists y, z \colon y \doteq x\, z\, x,$$

which returns all factors x that have two non-overlapping occurrences in w. As we not need to know where in w the factor xzx occurs, $\mathfrak{u}$ is not needed in the formula.

The restriction to a finite universe allows us to translate various classical results from FO to FC. Most importantly, model checking becomes not only decidable, but upper bounds can be lowered in the same way as for FO (Section 4.1). In fact, FC can be extended with iteration operators to characterize complexity classes from L to PSPACE, analogously to FO on ordered structures; which allows us to define a version of $\mathsf{Datalog}$ on words that includes concatenation (Section 4.3). Furthermore, Section 5 also describes how FC can be extended with *constraints* (aka *predicates*), while still keeping model checking tractable.

Spanners. An immediate application of FC is as a logic for *document spanners* (or just *spanners*). Spanners are a rule-based framework for information extraction that was proposed by Fagin, Kimelfeld, Reiss, and Vansummeren [19] to study the formal properties of the query language AQL of IBM's SystemT for information extraction. They can be understood as a combination of regular expressions and relational algebra.

In the last years, spanners have received considerable attention in the database theory community. The two main areas of interest are expressive power [19, 22, 23, 41, 43, 45, 47, 52] and efficient evaluation [3, 21, 24, 41, 42, 45, 46, 52]; further topics include updates [3, 25, 37], cleaning [20], distributed query planning [15], and a weighted variant [16].

But most of these articles do not focus on the full class of spanners that was introduced by Fagin et al. (called *core spanners*, as they describe the core of AQL), but a much smaller subclass, the *regular spanners*. The difference between these is that regular spanners cannot express equality of factors. Hence, techniques for finite automata and MSO often work on regular spanners; but they rarely work for core spanners. Furthermore, although spanners are conceptually similar to relational algebra, many canonical approaches for relational databases and the underlying FO are not viable in the spanner setting. In particular, while *acyclic conjuctive queries* are well-known to be tractable for FO (see e. g. [1]), this does not hold for the corresponding class of spanners (see [24]).

Although "pure" FC is not powerful enough to express core spanners, extending it with constraints that decide regular languages results in a logic that captures core spanners (Section 5.2). In addition to providing us with a rich and natural class of tractable spanners, this connection also allows us to develop a new inexpressibility method (Section 5.3).

2 Preliminaries

Let ε denote the *empty word*. We use $|x|$ for the length of a word, a formula, or a regular expression x, or the number of elements of a finite set x. A word v is a *factor* of a word w, written $v \sqsubseteq w$, if there exists (possibly empty) words p, s with $w = pvs$. For words x and y, let $x \sqsubseteq_{\mathsf{p}} y$ (x is a *prefix* of y) if $y = xs$ for some s, and $x \sqsubset_{\mathsf{p}} y$ if $x \sqsubseteq_{\mathsf{p}} y$ and $x \neq y$.

For alphabets A, B, a *morphism* is a function $h \colon A^* \to B^*$ with $h(u \cdot v) = h(u) \cdot h(v)$ for all $u, v \in A^*$. To define h, it suffices to define $h(a)$ for all $a \in A$. Let Σ be a finite *terminal alphabet*, and let Ξ be an infinite *variable alphabet* with $\Sigma \cap \Xi = \emptyset$. We assume Σ is fixed and $|\Sigma| \geq 2$, unless stated otherwise. As a convention, we use typewriter letters (like $\mathtt{a}$ and $\mathtt{b}$) for terminals.

Patterns and the theory of concatenation. A *pattern* is a word from $(\Sigma \cup \Xi)^*$. For every pattern $\eta \in (\Sigma \cup \Xi)^*$, let $\mathsf{Var}(\eta)$ denote the set of variables that occur in η. A *pattern substitution* (or just *substitution*) is a partial morphism $\sigma \colon (\Sigma \cup \Xi)^* \to \Sigma^*$ with $\sigma(\mathtt{a}) = \mathtt{a}$ for all $\mathtt{a} \in \Sigma$. When applying a substitution σ to a pattern η, we assume σ is defined on $\mathsf{Var}(\eta)$, that is, $\mathsf{Dom}(\sigma) \supseteq \mathsf{Var}(\eta)$. A *word equation* is a pair of patterns, that is, a pair (η_L, η_R) with $\eta_L, \eta_R \in (\Sigma \cup \Xi)^*$. We also write $\eta_L \doteq \eta_R$, and call η_L and η_R the *left side* and the *right side* of the equation. A *solution* of $\eta_L \doteq \eta_R$ is a substitution σ with $\sigma(\eta_L) = \sigma(\eta_R)$.

The theory of concatenation combines word equations with first-order logic. First the syntax: The set C of formulas of the *theory of concatenation* uses word equations $(\eta_L \doteq \eta_R)$ with $\eta_L, \eta_R \in (\Sigma \cup \Xi)^*$ as atoms. The connectives are conjunction, disjunction, negation, and quantifiers with variables from Ξ. For every $\varphi \in \mathsf{C}$, we define its set of *free variables* $\mathsf{free}(\varphi)$ by $\mathsf{free}(\eta_L \doteq \eta_R) := \mathsf{Var}(\eta_L) \cup \mathsf{Var}(\eta_R)$; extending this canonically.

The semantics build on solutions of word equations: For all $\varphi \in \mathsf{C}$ and all pattern substitutions σ with $\mathsf{Dom}(\sigma) \supseteq \mathsf{free}(\varphi)$, we define $\sigma \models \varphi$ as follows: Let $\sigma \models (\eta_L \doteq \eta_R)$ if $\sigma(\eta_L) = \sigma(\eta_R)$. For the quantifiers, we say $\sigma \models \exists x \colon \varphi$ (or $\sigma \models \forall x \colon \varphi$) if $\sigma_{x \mapsto w} \models \varphi$ holds for an (or all) $w \in \Sigma^*$, where $\sigma_{x \mapsto w}$ is defined by $\sigma_{x \mapsto w}(x) := w$ and $\sigma_{x \mapsto w}(y) := \sigma(y)$ for all $y \in (\Sigma \cup \Xi) - \{x\}$. The connectives' semantics are defined canonically.

▶ **Example 2.1.** Let $\varphi := x\mathtt{abc}y \doteq y\mathtt{bca}x \wedge \neg(x \doteq \varepsilon \vee y \doteq \varepsilon)$. Then $\sigma \models \varphi$ if and only if $\sigma(x\mathtt{abc}y) = \sigma(y\mathtt{bca}x)$, $\sigma(x) \neq \varepsilon$, and $\sigma(y) \neq \varepsilon$. For example, if $\sigma(x) = \mathtt{abca}$ and $\sigma(y) = \mathtt{a}$.

We freely add and omit parentheses as long as the meaning stays clear. $\mathsf{E\text{-}C}$, the *existential fragment* of C, consists of those formulas that do not use universal quantifiers and that apply negation only to word equations. The *existential-positive fragment* $\mathsf{EP\text{-}C}$ allows neither universal quantifiers, nor negation. We also use this notation for other logics that we define.

3 Finite models in the theory of concatenation

The new logic *finite model version of the theory of concatenation*, namely FC, is built around word equations; similarly to the theory of concatenation C. The latter can be understood as first-order logic over the universe Σ^* with concatenation – see for example [29], which refers to C as $\mathsf{FO}(A^*, \cdot)$. In other words, for C, we can consider the universe to be fixed (for a given terminal alphabet Σ). The key idea of FC is to replace the universe Σ^* with a single word and all its factors. In the formulas, this word is represented by a distinguished variable:

▶ **Definition 3.1.** *We distinguish a variable* $\mathfrak{u} \in \Xi$ *and call it the* universe variable.

As the universe variable represents the universe (hence its name), it has a special role in both syntax and semantics of FC. The *syntax of FC* restricts the syntax of C in two ways:

▶ **Definition 3.2.** *The set* FC *of* FC-formulas *is defined recursively: The atoms are word equations* $(\eta_L \doteq \eta_R)$ *with* $\eta_L \in \Xi$ *and* $\eta_R \in (\Sigma \cup \Xi)^*$*. These can be combined using disjunction* $(\varphi \vee \psi)$*, conjunction* $(\varphi \wedge \psi)$*, negation* $\neg\varphi$*, and quantifiers* $\exists x \colon \varphi$ *and* $\forall x \colon \varphi$ *with* $x \in \Xi - \{\mathfrak{u}\}$.

In other words, firstly, every word equation has a single variable on its left side. Secondly, the universe variable $\mathfrak{u}$ may not be bound by quantifiers. The reason for the first restriction is a bit subtle; we shall discuss it after defining the semantics. But the other follows immediately from the intuition that $\mathfrak{u}$ shall represent the universe – hence, binding it would make no sense. For the same reason, we also exclude $\mathfrak{u}$ from the free variables of FC-formulas:

▶ **Definition 3.3.** *The set* $\mathsf{free}(\varphi)$ *of free variables of an* FC*-formula* φ *is defined as for* C*-formulas, with the exception that* $\mathfrak{u}$ *is not considered a free variable.*

The *semantics of FC* combine those of C with the additional condition that the universe consists only of factors of the content of the universe variable $\mathfrak{u}$:

▶ **Definition 3.4.** *For* $\varphi \in \mathsf{FC}$ *and a pattern substitution* σ *with* $\mathsf{Dom}(\sigma) \supseteq \mathsf{free}(\varphi) \cup \{\mathfrak{u}\}$*, we define* $\sigma \models \varphi$ *as for* C*, but with the additional condition that* $\sigma(x) \sqsubseteq \sigma(\mathfrak{u})$ *for all* $x \in \mathsf{Dom}(\sigma)$.

To highlight the special role of $\mathfrak{u}$, we also write $(w, \sigma) \models \varphi$ if $\sigma \models \varphi$ and $w = \sigma(\mathfrak{u})$. We may shorten this to $w \models \varphi$ if φ is a *sentence* – that is, if $\mathsf{free}(\varphi) = \emptyset$. We write $\varphi(\vec{x})$ to denote that $\vec{x}$ is a tuple of free variables of φ.

▶ **Example 3.5.** Define $\varphi_1(y) := \exists x\colon x \doteq \mathtt{papaya}\, y\, \mathtt{banana}$ and $\varphi_2 := \exists x\colon (x \doteq \mathtt{papaya} \lor x \doteq \mathtt{banana})$. Then $(w, \sigma) \models \varphi_1$ if and only if $\sigma(y)$ occurs in w between $\mathtt{papaya}$ and $\mathtt{banana}$, and $w \models \varphi_2$ if and only if w contains $\mathtt{papaya}$ or $\mathtt{banana}$ as factor. Finally, let $\varphi_3(x) := \exists p, s\colon \big(\mathfrak{u} \doteq p\, x\, s \land \neg\exists \hat{p}, \hat{s}\colon (\mathfrak{u} \doteq \hat{p}\, x\, \hat{s} \land \neg \hat{p} \doteq p)\big)$. Then $(w, \sigma) \models \varphi_3$ if and only if $\sigma(x)$ occurs exactly once in w.

When applying σ to an FC-formula, $\sigma(\mathfrak{u})$ always needs to be defined – otherwise, we would have no universe to work with. But FC-formulas do not need to contain $\mathfrak{u}$. As a rule of thumb, $\mathfrak{u}$ is only required when referring to some "global" property of w. If we describe properties that are more "local" (as in the next example), we usually do not need to use $\mathfrak{u}$.

▶ **Example 3.6.** Let $\varphi^{\sqsubseteq_{\mathsf{p}}}(x, y) := \exists z\colon y \doteq xz$. Then $\sigma \models \varphi$ if and only if $\sigma(x)$ and $\sigma(y)$ are factors of $\sigma(\mathfrak{u})$ with $\sigma(x) \sqsubseteq_{\mathsf{p}} \sigma(y)$. In other words, $\varphi^{\sqsubseteq_{\mathsf{p}}}$ expresses $x \sqsubseteq_{\mathsf{p}} y$. Consequently, we can express $x \sqsubset_{\mathsf{p}} y$ through $\varphi^{\sqsubset_{\mathsf{p}}}(x, y) := \varphi^{\sqsubseteq_{\mathsf{p}}}(x, y) \land \neg x \doteq y$.

In fact, $\sqsubset_{\mathsf{p}}$ and inequality can be expressed without negation (or universal quantifiers). First, define $\varphi(x)^{\neq\varepsilon} := \exists y\colon \bigvee_{\mathtt{a}\in\Sigma} x \doteq \mathtt{a}\, y$ to express $x \neq \varepsilon$ – that is, $\sigma \models \varphi^{\neq\varepsilon}$ if and only if $\sigma(x) \sqsubseteq \sigma(\mathfrak{u})$ and $\sigma(x) \neq \varepsilon$. We use this in $\psi^{\sqsubset_{\mathsf{p}}}(x, y) := \exists z\colon (y \doteq x\, z \land \varphi^{\neq\varepsilon}(z))$. Like $\varphi^{\sqsubset_{\mathsf{p}}}$, this expresses $x \sqsubset_{\mathsf{p}} y$; but without negation.

Finally, let $\varphi^{\neq}(x, y) := \psi^{\sqsubset_{\mathsf{p}}}(x, y) \lor \psi^{\sqsubset_{\mathsf{p}}}(y, x) \lor \bigvee_{\mathtt{a},\mathtt{b}\in\Sigma, \mathtt{a}\neq\mathtt{b}} \exists x_1, y_1, z\colon (x \doteq z\, \mathtt{a}\, x_1 \land y \doteq z\, \mathtt{b}\, y_1)$. This states that $x \sqsubset_{\mathsf{p}} y$, $y \sqsubset_{\mathsf{p}} x$, or x and y differ after a common prefix z – that is, $x \neq y$.

We say $\varphi, \psi \in \mathsf{FC}$ are *equivalent*, written $\varphi \equiv \psi$, if for all σ with $\mathsf{Dom}(\sigma) \supseteq \mathsf{free}(\varphi) \cup \mathsf{free}(\psi) \cup \{\mathfrak{u}\}$, we have that $\sigma \models \varphi$ holds if and only if $\sigma \models \psi$. Thus, in Example 3.6, we have $\varphi^{\sqsubset_{\mathsf{p}}} \equiv \psi^{\sqsubset_{\mathsf{p}}}$. If $\varphi \in \mathsf{FC}$ is a sentence, we define its language as $\mathcal{L}(\varphi) := \{w \mid w \models \varphi\}$.

▶ **Example 3.7.** A language is called *star-free* if it is defined by a regular expression α that is constructed from the empty set $\emptyset$, terminals $\mathtt{a} \in \Sigma$, concatenation $\cdot$, union $\cup$, and complement $\overline{\alpha}$. Given such an α, we define $\varphi^{\alpha} := \exists x\colon (\mathfrak{u} \doteq x \land \psi^{\alpha}(x))$, where $\psi^{\alpha}(x)$ is defined recursively by $\psi^{\emptyset}(x) := \neg(x \doteq x)$, $\psi^{\mathtt{a}}(x) := (x \doteq \mathtt{a})$, $\psi^{(\alpha_1\cdot\alpha_2)}(x) := \exists x_1, x_2\colon \big(x \doteq x_1\, x_2 \land \psi^{\alpha_1}(x_1) \land \psi^{\alpha_2}(x_2)\big)$, $\psi^{(\alpha_1\cup\alpha_2)}(x) := \psi^{\alpha_1}(x) \lor \psi^{\alpha_1}(x)$, and $\psi^{\overline{\alpha}}(x) := \neg\psi^{\alpha}(x)$. Then $\sigma \models \psi^{\alpha}$ if and only if $\sigma(x) \in \mathcal{L}(\alpha)$ and $\sigma(x) \sqsubseteq \sigma(\mathfrak{u})$. Thus, $\mathcal{L}(\varphi^{\alpha}) = \mathcal{L}(\alpha)$.

We are now ready to discuss why Definition 3.2 restricts the left sides of word equations to single variables. Assume we allowed, for instance, the word equation $xy \doteq yx$ in an FC-formula, and consider the case of $\sigma(\mathfrak{u}) = \mathtt{a}^3$ and $\sigma(x) = \sigma(y) = \mathtt{a}^2$. Then $\sigma(x) \sqsubseteq \sigma(\mathfrak{u})$, $\sigma(y) \sqsubseteq \sigma(\mathfrak{u})$, and $\sigma(xy) = \sigma(yx)$ hold, but $\sigma(xy) = \mathtt{a}^4$ is not a factor of $\sigma(\mathfrak{u})$, which means that it is not in the universe.

There are two straightforward ways of allowing arbitrary word equations $\eta_L \doteq \eta_R$ in FC without changing the underlying universe. The first is adding the additional requirements $\sigma(\eta_L) \sqsubseteq \sigma(\mathfrak{u})$ and $\sigma(\eta_R) \sqsubseteq \sigma(\mathfrak{u})$ to the definition of $\sigma \models (\eta_L \doteq \eta_R)$. This can also be understood as declaring the concatenation as undefined if its result is not a factor of $\sigma(\mathfrak{u})$. The second is interpreting $\eta_L \doteq \eta_R$ as syntactic sugar for $\exists x\colon (x \doteq \eta_L \land x \doteq \eta_R)$, where x is a new variable.

On the other hand, this re-interpretation of the solutions of word equations can be considered non-intuitive, which makes formulas that rely on these easy to misunderstand. To avoid these issues, this paper restricts every left side to a single variables, even though this is not strictly necessary.

4 Properties of FC

In this section we analyze FC dynamically by discussing its *evaluation problem* – given a formula $\varphi \in \mathsf{FC}$ and a pattern substitution σ, decide whether $\sigma \models \varphi$. We call the special case where φ is a sentence the *model checking problem.* We also consider the *satisfiability problem* – given $\varphi \in \mathsf{FC}$, decide whether there is a pattern substitution σ with $\sigma \models \varphi$. After that, we also consider aspects of optimization of formulas.

4.1 Model checking vs satisfiability

For C, the model checking problem is undecidable. This is due to two reasons. Firstly, the satisfiability problem for C is undecidable by Quine [48]. Secondly, for C, satisfiability reduces to model checking – from a given $\varphi \in \mathsf{C}$, we can construct a C-sentence φ' by binding all free variables of φ existentially. Then φ is satisfiable if and only if $\sigma \models \varphi'$, no matter which substitution σ we choose. In contrast to this, the finite universe of FC drastically reduces the complexity of model checking.

▶ **Theorem 4.1.** *Evaluation is* PSPACE*-complete for* FC *and* NP*-complete for* $\mathsf{EP\text{-}FC}$*.*

In fact, the proof shows that the lower bounds hold even in the special case of model checking $\mathtt{a} \models \varphi$. Both the drop in complexity and the fact a very simple structure suffice are comparable to FO on finite relations (see e. g. [36]), and the proofs are equally straightforward. For both logics, the hardness of the problem comes from parameters of the formula and not of the word or the relational structure. FO provides us with another parameter to lower the complexity of model checking. We define the *width* $\mathsf{wd}(\varphi)$ of a formula φ as the maximum number of free variables in any of its subformulas.

▶ **Theorem 4.2.** *Model checking for* FC *can be solved in time* $O(k|\varphi|n^{2k})$*, for* $k := \mathsf{wd}(\varphi)$ *and* $n := |\sigma(\mathfrak{u})|$*.*

The proof also shows that this is only a rough upper bound; taking properties of variables into account lowers the exponent. In principle, we can apply various structure parameters for first-order formulas (see e. g. Adler and Weyer [2]) to FC. This assumes that we treat word equations as atomic formulas, which is certainly possible – but we can do better than that.

Decomposing patterns. Using a word equation $x \doteq \alpha$ as an atom results in a formula that has a width of at least $|\mathsf{Var}(\alpha)|$. Our goal is to lower that bound, by decomposing the pattern into a formula. Technically, a pattern $\alpha = \alpha_1 \cdots \alpha_n$ with $\alpha_i \in (\Xi \cup \Sigma)$ is a term $f(\alpha_1, \ldots, \alpha_n)$, where the function f is the n-ary concatenation. But there is a syntactic criterion that allows us to decompose α into a conjunction of binary concatenations. This builds on a result from combinatorics on words and formal languages, where a pattern α is also treated as generators of the *pattern languages* $\mathcal{L}(\alpha)$; the set of images of α under pattern substitutions. In this context, Reidenbach and Schmid [49] started a series of articles on classes of *pattern languages* with a polynomial time membership problem (surveyed in [40]), most of which rely on the following definition (see [8] for the definition of treewidth).

▶ **Definition 4.3.** *The* standard graph *of a pattern* $\alpha = \alpha_1 \cdots \alpha_n$ *with* $n \geq 1$ *and* $\alpha_i \in (\Sigma \cup \Xi)$ *is* $\mathcal{G}_\alpha := (V_\alpha, E_\alpha)$ *with* $V_\alpha := \{1, \ldots, n\}$ *and* $E_\alpha := E_\alpha^< \cup E_\alpha^=$*, where* $E_\alpha^<$ *is the set of all* $\{i, i+1\}$ *with* $1 \leq i < n$*, and* $E_\alpha^=$ *is the set of all* $\{i, j\}$ *such that* α_i *is some* $x \in \Xi$*, and* α_j *is the next occurrence of* x *in* α*. Then* $\mathsf{tw}(\alpha)$*, the* treewidth *of* α*, is the treewidth of* $\mathcal{G}_\alpha$*.*

As artificial (but simple) example, consider the sequence of patterns $\alpha_n := x_1 x_1 x_2 x_2 \cdots x_n x_n$. Then $|\mathsf{Var}(\alpha_n)| = n$, affecting the width of formulas that use α_n accordingly, but $\mathsf{tw}(\alpha_n) = 1$. Using tree decompositions, we can rewrite patterns of bounded treewidth into formulas of bounded width (similar to the proof of Kolaitis and Vardi [34] for variable bounded FO).

▶ **Theorem 4.4.** *Let $\varphi := \exists x_1, \ldots, x_m \colon y \doteq \alpha$. Then there exists $\psi \in$ EP-FC with $\psi \equiv \varphi$ and $\mathsf{wd}(\psi) \leq 2\mathsf{tw}(\alpha) + v$, where $v = 2 + |\mathsf{free}(y \doteq \alpha) - \{x_1, \ldots, x_m\}|$.*

For every fixed k, given φ with $\mathsf{tw}(\alpha) \leq k$, we can compute ψ in polynomial time.

Combining Theorems 4.2 and 4.4 yields a (slightly) different proof of the polynomial time decidability of the membership problem for classes of patterns with bounded treewidth from [49]. As pointed out in [9], bounded treewidth does not cover all pattern languages with a polynomial time membership problem, like e. g. patterns α^k where $\mathsf{tw}(\alpha)$ is bounded. But these languages can be expressed by $\exists x \colon (\mathfrak{u} \doteq x^k \wedge \varphi_\alpha(x))$, where $\varphi_\alpha(x)$ is a formula that expresses $x \in \mathcal{L}(\alpha)$, thus increasing the width by one. We leave a systematic examination whether all criteria for patterns with a tractable membership problem map to FC-formulas of bounded width for future work.

Satisfiability. Another parallel to FO is that satisfiability is undecidable for FC, even if we use only few variables. Let FC^k denote the set of formulas with width at most k.

▶ **Proposition 4.5.** *Satisfiability for FC^3 is undecidable if $|\Sigma| \geq 2$.*

The problem is trivial for FC^0 (see the proof of Theorem 4.8) and open for FC^1 and FC^2.

4.2 Static optimization

Apart from the width, Theorem 4.2 highlights the length of a formula as another parameter that influences the complexity of model checking. While the length of the patterns in the word equations might not seem to be factor that is overly important, there are patterns where straightforward optimizations can lead to an exponential advantage.

▶ **Example 4.6.** For $k \geq 1$, let $\varphi_k(y) := \exists x \colon y \doteq x^{2^k}$. Then $\varphi_k \equiv \psi_k := \exists x_1, \ldots, x_k \colon \big(y \doteq x_1 x_1 \wedge \bigwedge_{i=1}^{k-1} x_i \doteq x_{i+1} x_{i+1}\big)$, and $|\varphi_k|$ is exponential in k, while $|\psi_k|$ is linear in k. We can also rewrite ψ_k into a formula of width 3 by pulling quantifiers inwards and reusing variables. More specifically, we first rewrite each ψ_k into the equivalent formula

$$\exists x_1 \colon (y \doteq x_1 x_1 \wedge (\exists x_2 \colon x_1 \doteq x_2 x_2 \wedge \cdots (\exists x_{k-1} \colon x_k \doteq x_{k-1} x_{k-1}) \cdots)).$$

Then we replace every variable x_i with x_1 if i is odd or x_2 if i is even. The resulting formula has width 3 (due to y), is equivalent to φ_k, and has the same length.

This raises the questions whether we can computably minimize formulas and whether some fragments are more succinct than others. We address these questions in order.

▶ **Theorem 4.7.** *There is no algorithm that, given $\varphi \in$ FC, computes an equivalent ψ such that $|\psi|$ is minimal. This holds even if we restrict this to minimization within $\mathsf{EP\text{-}FC}^4$.*

This leaves open the decidability of, given $\varphi \in$ FC (or $\varphi \in$ EP-FC) and $k > 0$, is there an equivalent $\psi \in \mathsf{FC}^k$. But without suitable inexpressibility methods (see Section 5.3), we cannot even show that a language is inexpressible in FC^k for some $k > 0$, which complicates tackling this problem. The proof of Theorem 4.7 is actually more general and also demonstrates the undecidability of other common problems, like containment and equivalence.

Via Hartmanis' [30] meta theorem, certain undecidability results provide insights into the relative succinctness of models (see [35] or e. g. [23] for details). For two logics $\mathcal{F}_1$ and $\mathcal{F}_2$, the *tradeoff* from $\mathcal{F}_1$ to $\mathcal{F}_2$ is *non-recursive* if, for every computable $f\colon \mathbb{N} \to \mathbb{N}$, there exists some $\varphi \in \mathcal{F}_1$ that is expressible in $\mathcal{F}_2$, but $|\psi| \geq f(|\varphi|)$ holds for every $\psi \in \mathcal{F}_2$ with $\psi \equiv \varphi$.

▶ **Theorem 4.8.** *There are non-recursive tradeoffs from* $\mathsf{EP\text{-}FC}^4$ *to regular expressions and* FC^0*; and from* FC^4 *to* $\mathsf{EP\text{-}FC}$*, patterns, and singleton sets* $\{w\}$*.*

Note in particular that patterns can be parts of word equations. Hence, where Example 4.6 showed an exponential advantage in the rewriting, Theorem 4.8 shows FC^4 can obtain far larger advantages on certain classes of patterns.

4.3 Iteration and recursion

Iteration and recursion have been extensively studied in finite model theory and database theory. In particular, $\mathsf{FO}[<]$ that is extended with operators for transitive closure or fixed points captures various complexity classes (see e. g. [18, 36]). This is also closely connected to the recursive query language Datalog (see e. g. [1]). In this section, we shall define FC-Datalog, an FC-analog of Datalog. On the way, we shall also see that using transitive closures or fixed points on FC instead of $\mathsf{FO}[<]$ characterizes the same complexity classes.

Iteration. The definitions of these operators and the resulting extensions of FC are straightforward adaptions of their $\mathsf{FO}[<]$-versions (see e. g. [36] or [18]). But as they are also rather lengthy, we only give the intuitions behind them (detailed definitions can be found in the full version of this paper).

For $k \geq 1$, an FC-formula with $2k$ free variables can be viewed as generator of a relation over $R \subseteq S^k \times S^k$, where S is the universe (the set of factors of $\mathfrak{u}$). The operator tc computes the *transitive closure* $\mathsf{tc}(R)$ of R. If we view R as edge set of a directed graph over S^k, then tc computes the reachability relation in this graph. The *deterministic transitive closure* dtc is defined analogously, with the additional restriction that dtc stops at nodes with more than one outgoing edge. $\mathsf{FC}^{\mathsf{tc}}$ and $\mathsf{FC}^{\mathsf{dtc}}$ extend FC with tc and dtc, respectively.

For fixed points, we introduce special relation symbols as part of inductive definitions. Inside a fixed point operator, a formula φ may use a symbol $\dot{R}$ and at the same time also define the relation R inductively. We start with $R_0 := \emptyset$ and let R_1 be the relation that is defined by φ if $\dot{R}$ represents R_0. This is repeated, each R_i giving rise to R_{i+1}, until a fixed point is reached. For *least fixed points*, we ensure $R_i \subseteq R_{i+1}$ for all i. For *partial fixed points*, this is not required. We use $\mathsf{FC}^{\mathsf{lfp}}$ and $\mathsf{FC}^{\mathsf{pfp}}$ for the respective extensions of FC.

Complexity classes are commonly defined as classes of languages; and as we can treat FC and its extensions as language generators, connecting these two worlds is straightforward. We say that a logic $\mathcal{F}$ *captures* a complexity class $\mathbb{C}$ if $\mathbb{C}$ is the class of languages that are $\mathcal{F}$-definable – that is, $\mathbb{C} = \{\mathcal{L}(\varphi) \mid \varphi \in \mathcal{F}\}$.

The following result mirrors that for the respective extensions of $\mathsf{FO}[<]$:

▶ **Theorem 4.9.** $\mathsf{FC}^{\mathsf{dtc}}$, $\mathsf{FC}^{\mathsf{tc}}$, $\mathsf{FC}^{\mathsf{lfp}}$, $\mathsf{FC}^{\mathsf{pfp}}$ *capture* L, NL, P, PSPACE, *respectively.*

The result holds even if the formulas are required to be existential-positive. Thus, FC and even EP-FC behave under fixed-points and transitive closures like $\mathsf{FO}[<]$.

Recursion. This connection immediately suggests another: Recall that FO with least-fixed point operators can be used to define Datalog (see e. g. Part D of [1]). Analogously, we define FC-Datalog, a version of Datalog that is based on word equations.

An FC-Datalog-program is a tuple $P := (\mathcal{R}, \Phi, \mathsf{Ans})$, where $\mathcal{R}$ is a set of relation symbols that contains a special output symbol Ans, each $R \in \mathcal{R}$ has an arity $\mathsf{ar}(R)$, and Φ is a finite set of rules $R(\vec{x}) \leftarrow \varphi_1(\vec{y}_1), \ldots, \varphi_m(\vec{y}_m)$ with $R \in \mathcal{R}$, $m \geq 1$, each φ_i is an FC-word equation, and each x of $\vec{x}$ appears in some $\vec{y}_i$.

We define $[\![P]\!](w)$ incrementally, initializing the relations of all $R \in \mathcal{R}$ to $\emptyset$. For each rule $R(\vec{x}) \leftarrow \varphi_1(\vec{y}_1), \ldots, \varphi_m(\vec{y}_m)$, we enumerate all σ with $\sigma(\mathfrak{u}) = w$ and check if $\sigma \models \exists \vec{y} \colon \bigwedge_{i=1}^{m} \varphi_i$, where $\vec{y} := (\bigcup_{i=1}^{m} \vec{y}_i) - \vec{x}$. If this holds, we add $\sigma(\vec{x})$ to R. This is repeated until all relations have stabilized. Then $[\![P]\!](w)$ is the content of the relation Ans.

▶ **Example 4.10.** Define an FC-Datalog-program $(\{\mathsf{Ans}, E\}, \Phi)$, where $\mathsf{ar}(\mathsf{Ans}) = 0$, $\mathsf{ar}(E) = 3$, and Φ consists of the rules $\mathsf{Ans}() \leftarrow \mathfrak{u} \doteq xyz, E(x,y,z)$, and $E(x,y,z) \leftarrow x \doteq \varepsilon, y \doteq \varepsilon, z \doteq \varepsilon$, and $E(x,y,z) \leftarrow x \doteq \hat{x}\mathtt{a}, y \doteq \hat{y}\mathtt{b}, z \doteq \hat{z}\mathtt{c}, E(\hat{x},\hat{y},\hat{z})$. This defines the language $\{\mathtt{a}^n\mathtt{b}^n\mathtt{c}^n \mid n \geq 0\}$.

▶ **Theorem 4.11.** FC-Datalog *captures* P.

This is unsurprising, considering Datalog on ordered structures captures P, see e. g. [36], and the analogous result for spanners with recursion [47]. But it allows us to use word equations as a basis for Datalog on words. This provides potential applications for future insights into acyclicity for patterns, which could be combined with existing techniques for Datalog.

FC-Datalog can also be seen as a generalization of *range concatenation grammars (RCGs)*, see [6, 31], to use outputs and relations. There has been some work on parsing of RCGs (see [32] and its references). In the future, these might help identify tractable fragments of FC-Datalog. Vice versa, insights into the latter might lead to new approaches to RCG-parsing.

5 FC as a logic for document spanners

Fagin et al. [19] introduced *document spanners* (or just *spanners*) as a formal model of information extraction that is based on relational algebra (see e. g. [1]). This section connects spanners to FC. After stating the necessary definitions (Section 5.1), we extend FC into a logic for spanners (Section 5.2) and then use this for an inexpressibility proof (Section 5.3).

5.1 Spans and document spanners

A *span* of $w := \mathtt{a}_1 \cdots \mathtt{a}_n$ with $n \geq 1$ is an interval $[i, j\rangle$ with $1 \leq i \leq j \leq n+1$. It describes the factor $w_{[i,j\rangle} = \mathtt{a}_i \cdots \mathtt{a}_{j-1}$. For finite $V \subset \Xi$ and $w \in \Sigma^*$, a (V, w)*-tuple* is a function μ that maps each variable in V to a span of w. A *spanner* with variables V is a function P that maps every $w \in \Sigma^*$ to a set $P(w)$ of (V, w)-tuples. We use $\mathsf{Var}(P)$ for the variables of a spanner P. Accordingly, a spanner P is a function that takes an input word w and computes a relation $P(w)$ of $(\mathsf{Var}(P), w)$-tuples.

Like [19], we base spanners on *regex formulas*; regular expressions with variable bindings $x\{\alpha\}$. This matches the same words as the expression α and assigns the corresponding span of w to the variable x. For the purpose of this article, this informal definition shall suffice. Detailed definitions of the syntax and semantics of regex formulas can be found in [19] (the original definition that uses parse trees) and [22] (a more lightweight definition that uses the ref-words from Schmid [51]).

A regex formula is *functional* if on every word, every match has exactly one assignment for each variable. The set of functional regex formulas is RGX. For $\alpha \in \mathsf{RGX}$, we define the spanner $[\![\alpha]\!]$ as follows. Every match on $w \in \Sigma^*$ defines a $(\mathsf{Var}(\alpha), w)$-tuple μ, where each $\mu(x)$ is the span assigned to x; and $[\![\alpha]\!](w)$ is the set of all these μ.

► **Example 5.1.** We consider the regex formula $\alpha := \Sigma^*\big(x\{\mathtt{banana}\} \cup x\{\mathtt{papaya}\}\big)\Sigma^*$, which matches every word w that contains an occurrence of banana or papaya. The corresponding spanner $[\![\alpha]\!](w)$ contains all spans $[i,j\rangle$ with $w_{[i,j\rangle} \in \{\mathtt{banana}, \mathtt{papaya}\}$. Next, we define $\beta := \Sigma^* x\{\Sigma^*\}\Sigma^* y\{\Sigma^*\}\Sigma^*$. For every $w \in \Sigma^*$, we have that $[\![\beta]\!](w)$ contains those μ where $\mu(x)$ refers to a span to the left of $\mu(y)$.

We use the spanner operations union $\cup$, natural join $\bowtie$, projection π, set difference $-$, and equality selection $\zeta^=$. For spanners P_1 and P_2 with $\mathsf{Var}(P_1) = \mathsf{Var}(P_2)$, *union* and *set difference* are defined by $(P_1 \cup P_2)(w) := P_1(w) \cup P_2(w)$ and $(P_1 - P_2)(w) := P_1(w) - P_2(w)$ on all $w \in \Sigma^*$. Furthermore, the *projection* $\pi_V P$ for a spanner P and a set of variables $V \subset \mathsf{Var}(P)$ is obtained for every $w \in \Sigma^*$ by restricting the domain of every $\mu \in P(w)$ to V.

The *natural join* $P_1 \bowtie P_2$ combines spanner results by merging tuples that agree on the common variables. That is, $(P_1 \bowtie P_2)(w)$ contains those $(\mathsf{Var}(P_1) \cup \mathsf{Var}(P_2), w)$-tuples μ for which there exist $\mu_1 \in P_1(w)$ and $\mu_2 \in P_2(w)$ such that $\mu_1(x) = \mu_2(x)$ for all $x \in \mathsf{Var}(P_1) \cap \mathsf{Var}(P_2)$. An important consequence of this definition is that join is defined using spans (and thereby positions in the input word), not using the factors that occur in the spans. To compare factors, we use the *equality selection* $\zeta^=_{x,y}P$ with $x, y \in \mathsf{Var}(P)$. This is defined by $\zeta^=_{x,y}P(w) := \{\mu \in P(w) \mid w_{\mu(x)} = w_{\mu(y)}\}$ for $w \in \Sigma^*$.

By combining regex formulas with symbols for spanner operations, we obtain *spanner representations*; and their semantics are defined by applying the operations. The class of *generalized core spanner representations* $\mathsf{RGX}^{\mathsf{gcore}}$ consists of combinations of RGX and any of the five operators; the *core spanner representations* $\mathsf{RGX}^{\mathsf{core}}$ exclude set difference. According to Fagin et al. [19], "core spanners" capture the core functionality of IBM's SystemT.

► **Example 5.2.** Let α and β be the regex formulas from Example 5.1. We define the spanner representation $\varrho_1 := \alpha(x) \bowtie \alpha(y) \bowtie \beta(x,y)$. Then $\mathsf{Var}(\varrho_1) = \{x, y\}$, and $[\![\varrho_1]\!](w)$ contains those μ where $\mu(x)$ occurs before $\mu(y)$ in w and each of $w_{\mu(x)}$ and $w_{\mu(y)}$ is banana or papaya. Now let $\varrho_2 := \zeta^=_{x,y}\varrho_1$. Then $[\![\varrho_2]\!](w)$ is the subset of $[\![\varrho_1]\!](w)$ that also has $w_{\mu(x)} = w_{\mu(y)}$.

We identify spanners and their representations; e. g. by referring to a representation ϱ as a spanner (technically, $[\![\varrho]\!]$ is the spanner) or by calling the elements of $\mathsf{RGX}^{\mathsf{core}}$ *core spanners*.

5.2 Adding expressive power to FC

As core spanners are based on regular expressions, they can define all regular languages. This makes them more powerful than EP-FC. To prove this, we first connect FC to C.

► **Lemma 5.3.** *Given $\varphi \in \mathsf{FC}$, we can construct in polynomial time $\psi \in \mathsf{C}$ such that $\sigma \models \varphi$ if and only if $\sigma \models \psi$. This also preserves the properties existential and existential-positive.*

Hence, EP-FC is not more expressive than EP-C, which cannot express all regular languages – not even comparatively "harmless" languages like e. g. $\{\mathtt{a}, \mathtt{b}\}^*\mathtt{c}$ (see Karhumäki, Mignosi, Plandowski [33]). While we could define this specific language using negation, we shall address the issue in a way that generalizes far beyond regular languages and that does not require us to leave the existential-positive fragment (and its friendlier upper bounds). Complexity is also a reason why we do not use MSO or define a second-order version of FC.

Instead, take inspiration from C (see Diekert [13]). The *theory of concatenation with regular constraints*, C[REG], extends C by allowing *regular constraints* $x \mathrel{\dot\in} \alpha$ as atoms, where $x \in \Xi$, α is a regular expression, and $\sigma \models x \mathrel{\dot\in} \alpha$ if $\sigma(x) \in \mathcal{L}(\alpha)$. We define FC[REG] analogously, where $\sigma \models x \mathrel{\dot\in} \alpha$ has the additional condition that $\sigma(x) \sqsubseteq \sigma(\mathfrak{u})$ must hold.

▶ **Theorem 5.4.** *Theorem 4.1 and Theorem 4.2 also hold if we replace* FC *with* FC[REG] *and* EP-FC *with* EP-FC[REG].

In other words, evaluation is PSPACE-complete for FC[REG] and NP-complete for EP-FC[REG], and formula width can be used as parameter to bound model checking for FC[REG]. This generalizes to all constraints that can be decided in polynomial time, which allows us to adapt FC to other settings as well.

For example, string solvers often use *length constraints*. There are predicates that compare words by applying arithmetic to their lengths, like $|x| + |y| = |z|$. While the applications of EP-C in a string solver context usually rely on deciding satisfiability, cases where model checking suffices could benefit from using FC with appropriate constraints.

Regarding prior work, the C[REG]-fragments SpLog and SpLog$^\neg$ were introduced in [22] as alternatives to RGX$^{\mathsf{core}}$ and RGX$^{\mathsf{gcore}}$, respectively. As these ensure the finite universe purely through syntax, they are more cumbersome than FC and do not generalize as nicely.

Before we connect FC[REG] to spanners, we take a brief look at restricted regular expressions that can be expressed in FC. We call a regular expression *simple* if the operator * is only applied to terminal words or to Σ (a shorthand for $\bigcup_{\mathtt{a}\in\Sigma}\mathtt{a}$). That is, if $\Sigma = \{\mathtt{a},\mathtt{b},\mathtt{c}\}$, then $(\mathtt{abc})^*\Sigma^*$ is simple, but $(\mathtt{a}\cup\mathtt{b})^*$ and $(\mathtt{a}(\mathtt{b})^*)^*$ are not.

▶ **Lemma 5.5.** *For every simple regular expression α, there is $\varphi^\alpha(x) \in$ EP-FC such that $(w,\sigma) \models \varphi^\alpha$ if and only if $\sigma(x) \in \mathcal{L}(\alpha)$ and $\sigma(x) \sqsubseteq w$.*

The proof uses the characterization of commuting words (see e.g. Lothaire [38]). We shall use this Lemma in the proof of Theorem 5.14, to replace regular constraints.

FC[REG] and Spanners. As we want to use FC[REG] for spanners, we still need to close a formal gap, namely that spanners reason over positions in a word, while FC[REG] reasons over words. We bridge this gap through the notion of one *realizing* the other, which [22] introduced for the logic SpLog. We begin with formulas that realize spanners.

▶ **Definition 5.6.** *A substitution σ* expresses *a (V,w)-tuple μ if* $\mathsf{Dom}(\sigma) \supseteq \{x^P, x^C \mid x \in V\}$ *and, for all $x \in V$, we have $\sigma(x^P) = w_{[1,i\rangle}$ and $\sigma(x^C) = w_{[i,j\rangle}$ for $[i,j\rangle = \mu(x)$.*

A formula $\varphi \in$ FC[REG] realizes *a spanner P if* $\mathsf{free}(\varphi) = \{x^P, x^C \mid x \in \mathsf{Var}(P)\}$ *and, for all $w \in \Sigma^*$, we have $(w,\sigma) \models \varphi$ if and only if σ expresses some $\mu \in P(w)$.*

In other words, x^C is $w_{\mu(x)}$ (the content of x), and x^P is the prefix of w before $w_{\mu(x)}$.

▶ **Example 5.7.** In Example 5.1, we defined $\alpha := \Sigma^*\big(x\{\mathtt{banana}\}\cup x\{\mathtt{papaya}\}\big)\Sigma^*$. Its spanner $[\![\alpha]\!]$ is realized by $\varphi(x^P,x^C) := \exists y\colon \mathfrak{u} \doteq x^P x^C y \wedge \big(x^C \doteq \mathtt{banana} \vee x^C \doteq \mathtt{papaya}\big)$.

Then $(w,\sigma) \models \varphi$ if σ expresses some $\mu \in [\![\alpha]\!](w)$. That is, $\sigma(x^C)$ contains $w_{\mu(x)}$ (i.e., $\mathtt{banana}$ or $\mathtt{papaya}$), and $\sigma(x^P)$ contains the prefix in w before it.

To show that FC[REG] cannot express more than the classes of spanners that we consider, we also define the notion of spanners that realize formulas.

▶ **Definition 5.8.** *A spanner P* realizes *$\varphi \in$ FC[REG] if $\mathsf{Var}(P) = \mathsf{free}(\varphi)$ and, for all $w \in \Sigma^*$, we have $\mu \in P(w)$ if and only if $(w,\sigma) \models \varphi$ for the σ with $\sigma(x) := w_{\mu(x)}$ for all $x \in \mathsf{Var}(P)$.*

There are *polynomial-time conversions* from a class of formulas (or spanners) A to a class of spanners (or formulas) B if, given $x \in A$, we can compute in polynomial time $y \in B$ that realizes x. We write $A \equiv_{\mathsf{poly}} B$ if there are polynomial-time conversions from A to B and from B to A.

▶ **Theorem 5.9.** FC[REG] $\equiv_{\mathsf{poly}}$ RGX$^{\mathsf{gcore}}$ *and* EP-FC[REG] $\equiv_{\mathsf{poly}}$ RGX$^{\mathsf{core}}$.

5.3 Inexpressibility for FC, FC[REG], and spanners

There are currently only few inexpressibility methods for FC and FC[REG], as there are only few such methods for related models like spanners or the theory of concatenation. A detailed discussion from the point of view of $\mathsf{RGX}^{\mathsf{core}}$ and SpLog can be found in Section 6 of [22]. These techniques do not account for negation, which makes them inapplicable for FC or FC[REG]. A standard tool for FO-inexpressibility are Ehrenfeucht–Fraïssé games (e. g. [36]). But as concatenation acts as a generalized addition, using these for FC or FO[EQ] is far from straightforward. Another standard tool is the Feferman-Vaught theorem (see [39]). While this can be used for FC, the factor universe of FC makes decomposing the structure into disjoint sets inconvenient. Instead of following down this road, we introduce FO[EQ], an extension of FO[<] that has the same expressive power as FC.

Connecting FC to FO[<]. In this section, we establish connections between FC and "classical" relational first-order logic. It is probably safe to say that in finite model theory, the most common way of applying first-order logic to words is the logic FO[<] (and the more general MSO). This uses the equality $\doteq$ and a vocabulary that consists of a binary relation symbol $<$ and unary relation symbols $\mathsf{P_a}$ for each $\mathsf{a} \in \Sigma$. Every word $w = a_1 \cdots a_n \in \Sigma^+$ with $n \geq 1$ is represented by a structure $\mathcal{A}_w$ with universe $\{1, \ldots, n\}$. For every $\mathsf{a} \in \Sigma$, the relation $\mathsf{P_a}$ consists of those i that have $a_i = \mathsf{a}$. To simplify dealing with ε, we slightly deviate from this standard structure. For every $w \in \Sigma^*$, we extend $\mathcal{A}_w$ to $\mathcal{A}'_w$ by adding an additional "letter-less" node $|w| + 1$ that occurs in no $\mathsf{P_a}$. Then we have a one-to-one correspondence between pairs (i, j) with $i \leq j$ from the universe of $\mathcal{A}'_w$ and the spans $[i, j\rangle$ of w (see Section 5.1), and $w = \varepsilon$ does not require a special case.

▶ **Definition 5.10.** FO[EQ] *extends* FO[<] *with constants* min *and* max*, the binary relation symbol* succ*, and the 4-ary relation symbol* Eq*. For every $w \in \Sigma^*$ and the corresponding structure $\mathcal{A}'_w$, these symbols express $\mathsf{min} = 1$, $\mathsf{max} = |w| + 1$, $\mathsf{succ} = \{(i, i+1) \mid 1 \leq i \leq |w|\}$, and* Eq *contains those (i_1, j_1, i_2, j_2) with $i_1 \leq j_1$ and $i_2 \leq j_2$ such that $w_{[i_1,j_1\rangle} = w_{[i_2,j_2\rangle}$. We write $(w, \alpha) \models \varphi$ to denote that α is a satisfying assignment for φ on $\mathcal{A}'_w$.*

▶ **Example 5.11.** The FO[EQ]-formula $\exists x\colon \mathsf{Eq}(\mathsf{min}, x, x, \mathsf{max})$ defines $\{ww \mid w \in \Sigma^*\}$.

Technically, we do not need the symbols min, max, or succ, as these can be directly expressed in FO[<]. But these constants allows us to better preserve the structural similarities when converting between various fragments of FC and FO[EQ].

When comparing FC to FO[EQ], we need to address that one operates on words and the other on positions. We can handle this in a way that is similar to the situation between FC and spanners; and this can be used to show that there are polynomial time conversions between FC and FO[EQ] that preserve the properties existential and existential-positive, and only marginally increase the width of the formulas.

In fact, these transformations show that one could choose FO[EQ] over FC as a logic for words (or for spanners, if one extends FO[EQ] with regular constraints or generalizes it to MSO with Eq). This is a valid choice, if one prefers writing $\exists x_1, \ldots, x_6\colon \big(\mathsf{P_p}(x_1) \wedge \mathsf{P_a}(x_2) \wedge \mathsf{P_p}(x_3) \wedge \mathsf{P_a}(x_4) \wedge \mathsf{P_y}(x_5) \wedge \mathsf{P_a}(x_6) \wedge \bigwedge_{i=1}^{5} \mathsf{succ}(x_i, x_{i+1})\big)$ over $\exists x\colon x \doteq \mathsf{papaya}$ or if one wants to express $x \doteq yz$ as $\exists x^m\colon \big(\mathsf{Eq}(x^o, x^m, y^o, y^c) \wedge \mathsf{Eq}(x^m, x^c, z^o, z^c)\big)$ instead.

Details on these conversions (and the required definitions) can be found in the full version of this paper. For the sake of finding an inexpressibility result, we only require the following.

▶ **Lemma 5.12.** *A language is definable in* FC *if and only if it is definable in* FO[EQ]*.*

Proving inexpressibility. Lemma 5.12 allows us to use Feferman-Vaught theorem, at least when considering languages that are restricted enough.

▶ **Lemma 5.13.** *There is no* FC*-formula that defines* $\{\mathtt{a}^n\mathtt{b}^n \mid n \geq 1\}$.

Moreover, we can show that regular constraints offer no help for defining this language.

▶ **Theorem 5.14.** FC[REG] *cannot express the equal length relation* $|x| = |y|$.

As FC[REG] has the same expressive power as $\mathsf{RGX}^{\mathsf{gcore}}$, this is the first inexpressible result for $\mathsf{RGX}^{\mathsf{gcore}}$ on non-unary alphabets. The proof has two parts, which both rely on the limited structure of the language $\mathtt{a}^n\mathtt{b}^n$. One part is using Lemma 5.13, wich applies the Feferman-Vaught theorem. The other is using Lemma 5.5 to eliminate the regular constraints, which is based on combinatorics on words. The authors expect that a more general inexpressibility method for FC (or even FC[REG]) would need to combine more advanced techniques from combinatorics on words (like those in [33]) with methods from logic.

6 Conclusions and future work

On words, concatenation is one of the most natural operations. But as seen for C, using concatenation with first-order logic quickly becomes undecidable. Restricting the universe to a word and all its factors changes the situation drastically. In contrast to C, the resulting logic FC has a meaningful distinction between satisfiability and model checking; and the latter is not only decidable, but we can use the structure of the formula to derive upper bounds in the same way as for FO over finite structures. In addition to this, FC can also replace FO[<] as "base" logic for characterizing complexity classes. Hence, while one might certainly make a case against the claim that FC is *the* finite model version of the theory of concatenation, the results leave little doubt that it is at least *a* valid approach.

FC also provides an extendable framework for querying and model checking words, in particular for scenarios that rely on expressing that factors appear multiple times. If more expressive power is needed, FC is easily extended with constraints, without affecting the lower bounds on evaluation and model checking. In particular, we can translate core and generalized core spanners to FC[REG] and then analyze or optimize these formulas with respect to parameters like width. To a degree, this was also possible the spanner logic SpLog, but FC is more elegant, easier to use, and behaves much more like FO on relational databases.

Future work

Many fundamental questions remain open, in particular for model checking and related problems, like evaluation and enumeration.

Compilation into tractable fragments. One promising direction is the compiling of formulas into equivalent formulas of a fragment where these problems can be solved more efficiently. For example, Theorem 4.2 shows that bounding the width of the formulas leads to tractable model checking. Theorem 4.4 then provides us with a sufficient criterion for formulas that can be rewritten into formulas with a lower width, by decomposing the pattern of word equations. It is likely that this approach can be further refined by not just rewriting single patterns, but taking the larger formula into account. This approach can also be used with other structure parameter for formulas (like acyclicity and bounded tree width), by developing a corresponding variant of Theorem 4.4. One example of this is [26], which adapts the concept of acyclic conjunctive queries to FC.

A more fundamental question is whether all tractable fragments of FC can be explained through the criterion of bounded width (or are subset of a larger tractable fragment that is explained through it). A good starting point for this line of investigation is the question whether all classes of pattern languages with a polynomial time membership problem can be explained through bounded width.

Model checking as parsing. Another approach – that is not investigated in the present paper – is the connection to parsing algorithms. This is a natural question, as model-checking EP-FC-formulas can be understood as a parsing problem (where variables are mapped to factors of the input word). Promising starting points for this are parsing algorithms for RCGs (recall Section 4.3) and related grammars, and the extraction grammars from [45].

Data structures. Model checking algorithms will likely benefit from specialized data structures. For example, a naive representation of all factors of a word of length n would contain about $O(n^2)$ elements, and if these are just represented directly as words, this would take $O(n^3)$ memory. But using data structures like suffix trees and suffix arrays, one can create in time $O(n)$ a data structure that allows us to enumerate all factors with constant delay (see [26], which also examines small word equations).

While these optimizations do not matter if one considers polynomial time efficient enough, it would be very useful to know which fragments can be model-checked in time $O(n^k)$ for small k, or even in sub-quadratic time.

Inexpressibility and satisfiability. Our results on inexpressibility also leave many questions open. Lemma 5.13 heavily relies on the limited structure of the language. This is the same situation as in Section 6.1 of [22], which describes an inexpressibility technique for EP-FC[REG]. Although these two approaches provide us with some means of proving inexpressibility, they only cover special cases, and much remains to be done. It seems likely that a more general method will need to combine approaches from finite model theory (like the Feferman-Vaught theorem that we used for Lemma 5.13) with techniques from combinatorics on words (like those in [33] that [22] uses). A related problem that is still open is whether EP-FC has the same expressive power as EP-C.

Of particular interest is finding a method to prove inexpressibility in FC^k for some $k > 0$. This problem relates to the open questions whether there are algorithms that minimize the width of a formula, and for which k the fragment FC^{k+1} is more expressive than FC^k. The authors conjecture that this holds for all $k \geq 0$, which would contrast with $\mathsf{FO}[<]$, where the fragment of formulas with width three has the same expressive power as the full logic. Finally, it remains open whether satisfiability is decidable for FC^1 or FC^2.

Beyond FC. Using FC as a logic for spanners (and other models, potentially) raises further questions. For example, while every tractable fragment of FC[REG] maps to a tractable fragment of core spanners (namely, those that are obtained by converting the formulas), there is no guarantee that the obtained fragment is natural. Hence, a more detailed investigation into conversions between FC[REG] and $\mathsf{RGX}^{\mathsf{core}}$ is justified.

There are many other possible directions. For example, one could easily define a second-order version of FC and adapt various results from SO. Moreover, FC could be examined from an algebra point of view, or related to rational and regular relations.

References

1 Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995. URL: `http://webdam.inria.fr/Alice/`.

2 Isolde Adler and Mark Weyer. Tree-width for first order formulae. *Log. Methods Comput. Sci.*, 8(1), 2012.

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

4 Michael Benedikt, Leonid Libkin, Thomas Schwentick, and Luc Segoufin. Definable relations and first-order query languages over strings. *J. ACM*, 50(5):694–751, 2003.

5 Anthony J. Bonner and Giansalvatore Mecca. Sequences, datalog, and transducers. *J. Comput. Syst. Sci*, 57(3):234–259, 1998.

6 Pierre Boullier. Range concatenation grammars. In *New developments in parsing technology*, pages 269–289. Springer, 2004.

7 Laura Ciobanu, Volker Diekert, and Murray Elder. Solution sets for equations over free groups are EDT0L languages. *IJAC*, 26(5):843–886, 2016.

8 Marek Cygan, Fedor V. Fomin, Lukasz Kowalik, Daniel Lokshtanov, Dániel Marx, Marcin Pilipczuk, Michal Pilipczuk, and Saket Saurabh. *Parameterized Algorithms.* Springer, 2015.

9 Joel D. Day, Pamela Fleischmann, Florin Manea, Dirk Nowotka, and Markus L. Schmid. On matching generalised repetitive patterns. In *Proc. DLT 2018*, pages 269–281, 2018.

10 Joel D. Day, Vijay Ganesh, Paul He, Florin Manea, and Dirk Nowotka. The satisfiability of word equations: Decidable and undecidable theories. In *Proc. RP 2018*, pages 15–29, 2018.

11 Joel D. Day and Florin Manea. On the structure of solution sets to regular word equations. In *Proc. ICALP 2020*, pages 124:1–124:16, 2020.

12 Joel D. Day, Florin Manea, and Dirk Nowotka. The hardness of solving simple word equations. In *Proc. MFCS 2017*, pages 18:1–18:14, 2017.

13 Volker Diekert. Makanin's Algorithm. In M. Lothaire, editor, *Algebraic Combinatorics on Words*, chapter 12. Cambridge University Press, 2002.

14 Volker Diekert. More than 1700 years of word equations. In *Proc. CAI 2015*, 2015.

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

16 Johannes Doleschal, Benny Kimelfeld, Wim Martens, and Liat Peterfreund. Weight annotation in information extraction. In *Proc. ICDT 2020*, pages 8:1–8:18, 2020.

17 V. G. Durnev. Undecidability of the positive $\forall\exists^3$-theory of a free semigroup. *Siberian Mathematical Journal*, 36(5):917–929, 1995.

18 Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory.* Springer, 2nd edition, 1999.

19 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Document spanners: A formal approach to information extraction. *J. ACM*, 62(2):12:1–12:51, 2015.

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

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

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

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

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

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

26 Dominik D. Freydenberger and Sam M. Thompson. Splitting spanner atoms: A tool for acyclic core spanners, 2021. `arXiv:2104.04758`.

27 Seymour Ginsburg and Xiaoyang Sean Wang. Regular sequence operations and their use in database queries. *J. Comput. Syst. Sci*, 56(1):1–26, 1998.

28 Gösta Grahne, Matti Nykänen, and Esko Ukkonen. Reasoning about strings in databases. *J. Comput. Syst. Sci*, 59(1):116–162, 1999.

29 Simon Halfon, Philippe Schnoebelen, and Georg Zetzsche. Decidability, complexity, and expressiveness of first-order logic over the subword ordering. In *Proc. LICS 2017*, pages 1–12, 2017.

30 Juris Hartmanis. On Gödel speed-up and succinctness of language representations. *Theor. Comput. Sci.*, 26:335–342, 1983.

31 Laura Kallmeyer. *Parsing Beyond Context-Free Grammars.* Cognitive Technologies. Springer, 2010.

32 Laura Kallmeyer, Wolfgang Maier, and Yannick Parmentier. An earley parsing algorithm for range concatenation grammars. In *Proc. ACL-IJCNLP 2009*, pages 9–12, 2009.

33 Juhani Karhumäki, Filippo Mignosi, and Wojciech Plandowski. The expressibility of languages and relations by word equations. *J. ACM*, 47(3):483–505, 2000.

34 Phokion G. Kolaitis and Moshe Y. Vardi. Conjunctive-query containment and constraint satisfaction. *J. Comput. Syst. Sci*, 61(2):302–332, 2000.

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

36 Leonid Libkin. *Elements of Finite Model Theory.* Texts in Theoretical Computer Science. Springer, 2004.

37 Katja Losemann. *Foundations of Regular Languages for Processing RDF and XML.* PhD thesis, University of Bayreuth, 2015. URL: `https://epub.uni-bayreuth.de/2536/`.

38 M. Lothaire. *Combinatorics on Words.* Addison-Wesley, Reading, MA, 1983.

39 Johann A. Makowsky. Algorithmic uses of the Feferman-Vaught theorem. *Annals of Pure and Applied Logic*, 126(1-3):159–213, 2004.

40 Florin Manea and Markus L. Schmid. Matching patterns with variables. In *Proc. WORDS 2019*, pages 1–27, 2019.

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

42 Andrea Morciano, Martin Ugarte, and Stijn Vansummeren. Automata-based evaluation of AQL queries. Technical report, Université Libre de Bruxelles, 2016.

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

44 Dirk Nowotka and Aleksi Saarela. An optimal bound on the solution sets of one-variable word equations and its consequences. In *Proc. ICALP 2018*, pages 136:1–136:13, 2018.

45 Liat Peterfreund. Grammars for document spanners. In *Proc. ICDT 2021*, pages 7:1–7:18, 2021.

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

47 Liat Peterfreund, Balder ten Cate, Ronald Fagin, and Benny Kimelfeld. Recursive programs for document spanners. In *Proc. ICDT 2019*, pages 13:1–13:18, 2019.

48 W. V. Quine. Concatenation as a basis for arithmetic. *J. Symb. Log.*, 11(4):105–114, 1946.

49 Daniel Reidenbach and Markus L. Schmid. Patterns with bounded treewidth. *Inf. Comput.*, 239:87–99, 2014.

50 Aleksi Saarela. Hardness results for constant-free pattern languages and word equations. In *Proc. ICALP 2020*, pages 140:1–140:15, 2020.

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

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

53 Howard Straubing. *Finite Automate, Formal Logic, and Circuit Complexity*. Birkhäuser, 1994.