FC-Datalog as a Framework for Efficient String Querying

Owen M. Bell
Loughborough University, UK

Joel D. Day
Loughborough University, UK

Dominik D. Freydenberger
Loughborough University, UK

Abstract

Core spanners are a class of document spanners that capture the core functionality of IBM's AQL. FC is a logic on strings built around word equations that when extended with constraints for regular languages can be seen as a logic for core spanners. The recently introduced FC-Datalog extends FC with recursion, which allows us to define recursive relations for core spanners. Additionally, as FC-Datalog captures P, it is also a tractable version of Datalog on strings. This presents an opportunity for optimization.

We propose a series of FC-Datalog fragments with desirable properties in terms of complexity of model checking, expressive power, and efficiency of checking membership in the fragment. This leads to a range of fragments that all capture LOGSPACE, which we further restrict to obtain linear combined complexity. This gives us a framework to tailor fragments for particular applications. To showcase this, we simulate deterministic regex in a tailored fragment of FC-Datalog.

2012 ACM Subject Classification Theory of computation → Logic and databases

Keywords and phrases Information extraction, word equations, datalog, document spanners, regex

Digital Object Identifier 10.4230/LIPIcs.ICDT.2025.29

Related Version *Full Version*: `https://arxiv.org/abs/2501.10344`

Supplementary Material Data Accessibility: No data was generated or analyzed during this study.

Funding This work was supported by EPSRC grant EP/T033762/1.

Acknowledgements The authors would like to thank the anonymous reviewers of the current and previous versions for their detailed and helpful feedback.

1 Introduction

As a vast amount of valuable information is stored in unstructured textual data, the operation of extracting structured information from such data is crucial. This operation is the classical task known as Information Extraction (IE), and has applications from healthcare (see e.g. [39]) to social media analytics (see e.g. [5]). A popular approach to this task is the rule-based technique, which can be understood as querying a text as one queries a relational database. Document spanners are a framework for rule-based information extraction. We consider a recursive model connected to document spanners called FC-Datalog, which is based on the logic on strings FC and the query language Datalog. Thus, we first discuss the latter two, and the connection to document spanners, before moving to the former.

FC. The *theory of concatenation* (short: C) is a logic on strings with the infinite universe Σ^*. Introduced by Freydenberger and Peterfreund [16], FC is a finite model version of C which has a finite universe, a single word and all of its factors (contiguous subwords). As a result of this

restriction, FC has decidable model checking and evaluation (see [16]). FC is built on *word equations*, equations of the form $xx \doteq yyy$, where variables x and y represent words over a finite alphabet Σ. As a result, in FC we can reason directly over factors rather than intervals of positions as is the case for other logics on strings such as *monadic second order logic* (MSO) over a linear order. Furthermore, in FC we can compare factors of unbounded length, something which is not possible in MSO (see [16] for details). Word equations themselves are a natural way of expressing many typical properties of a string such as containing a square or being imprimitive (see e.g. [24]), and have previously been used for data management in other areas such as graph databases (see [6]).

Document Spanners and FC. Introduced by Fagin, Kimelfeld, Reiss and Vansummeren [12], *document spanners* (or *spanners*) are a rule-based framework for Information Extraction. Spanners were introduced to capture the core functionality of AQL, a query language used for IBM's SystemT. Informally, spanners are functions that take a text document as input and output a relation over intervals (called *spans*) from the document. Intuitively, primitive extractors (which are commonly regular expressions with capture variables called *regex formulas*) extract relations of spans, and these are then combined using relational algebra.

Many works on spanners, particularly in the area of enumeration (see e.g. [3, 13, 35]), have been concerned with the subclass of *regular spanners*, which are regex formulas extended with projection (π), union ($\cup$) and natural join ($\bowtie$). However, [12] showed that this subclass cannot express more than recognizable relations. The full class of spanners introduced by Fagin et al. [12] are called the *core spanners* as these achieve the original motivation of capturing the core functionality of AQL. Core spanners extend the regular spanners with string equality (denoted $\zeta^=$), an operation necessary to perform fundamental tasks such as reading multiple occurrences of a string. This added expressibility comes at the cost of reduced efficiency (see e.g. [14, 15]). Further extending the core spanners with set difference gives the class of *generalized core spanners*.

The logic FC has a tight connection to core spanners. In particular, the extension FC[REG], which extends FC with constraints that decide membership of regular languages, captures the expressive power of generalized core spanners, and the existential-positive fragment EP-FC[REG] captures the expressive power of core spanners. Furthermore, there are polynomial time conversions between FC[REG] and generalized core spanners, and EP-FC[REG] and core spanners (see Section 5.2 of [16]). While spanners reason over intervals of positions, FC reasons directly over factors. When dealing with factors, unlike with intervals of positions, the default is not to distinguish between duplicates. However, simulating different intervals containing the same factor is easily done: we can simply store in addition to the factor, the prefix preceding it. On the other hand, eliminating duplicates when working with intervals, such as in spanners, is not so easily achieved. Due to the connection between the two models, we can use FC to gain insights into spanners; previous examples of work on FC that has produced results for spanners are [18] for tractability and [38] for inexpressibility.

Datalog. A query language for relational databases, Datalog was introduced to perform operations that were not possible in earlier database languages such as graph transitive closure (see e.g. [1]). A Datalog program has a database of prepopulated *extensional* relations and a set of recursive rules that define new *intensional* relations. Semi-positive Datalog, which allows negation for atoms with extensional relation symbols, captures P on ordered structures (see e.g. [28]). Linear Datalog permits at most one atom with an intensional relation symbol in the body of every rule, and semi-positive linear FC-Datalog captures NLOGSPACE on ordered structures (see e.g. [20]). A more general definition of linear Datalog permits, in the

body of every rule, at most one atom with an intensional relation symbol mutually recursive with the head relation symbol (see e.g [1]). We can evaluate body atoms that have other intensional relation symbols as subroutines, and so this extended definition does not affect the complexity. Linear Datalog also captures how recursion is done in SQL (see [30]).

The combined complexity of model checking Datalog is EXP-complete, even if the input database is empty, the universe is made up of only two elements, and the program has only a single rule (see [10, 20]). For linear Datalog it is PSPACE-complete, again even with an empty input database, a two-element universe, and a single rule (see [20]).

An earlier approach to adapting Datalog for querying strings is Sequence Datalog (see [7]), but this has an undecidable model checking problem. Furthermore, in the spanner setting Peterfreund, ten Cate, Fagin and Kimelfeld [33] introduced RGXlog, Datalog over regex formulas. RGXlog was motivated by the SystemT developers' interest in recursion (for example, to implement context free grammars for natural language processing), and captures the complexity class P. As introduced in [32], Spannerlog$\langle$RGX$\rangle$ generalizes the spanner and relational model and has recently been implemented in [29]. Spannerlog$\langle$RGX$\rangle$ with stratified negation and restricted to string extensional relations also captures P (see Section 6 of [33]).

FC-Datalog. Together with FC, Freydenberger and Peterfreund [16] also introduced FC-Datalog, which extends existential-positive FC (EP-FC) with recursion analogously to how Datalog extends existential-positive FO. It is worth pointing out that EP-FC is able to express the inequality of two strings (see e.g. Example 5.3 in [14]). As in FC, we have the finite universe of a single word and all of its factors. FC-Datalog has word equations instead of extensional relations. That is, FC-Datalog atoms are word equations or relations. In FC-Datalog we adopt the *fixed point* Datalog semantics.

▶ **Example 1.1.** The rules of an FC-Datalog program P:

$$\begin{aligned} \mathrm{Ans}() &\leftarrow \mathfrak{u} \doteq yz,\ \ E(y,z);\\ E(x,y) &\leftarrow x \doteq \varepsilon,\ \ y \doteq \varepsilon;\\ E(x,y) &\leftarrow x \doteq \mathsf{a}u,\ \ y \doteq \mathsf{b}v,\ \ E(u,v). \end{aligned}$$

P defines the language $\mathcal{L}(P) := \{\mathsf{a}^n\mathsf{b}^n \mid n \in \mathbb{N}\}$. See Definition 2.1 for the semantics.

From Theorem 4.11 of [16], FC-Datalog captures the complexity class P. When considering efficiency, we are primarily interested in model checking, which relates practically to deciding if a tuple is in a relation. We can see FC-Datalog as a language for expressing relations that can be used in spanner selections. Because spanners reason over intervals of positions, expressing relations can become cumbersome as a relation holds such intervals, including all those that represent the same factor. In contrast, we can neatly express relations in FC-Datalog as we reason directly over factors and so a relation holds the factors themselves.

▶ **Example 1.2.** In FC-Datalog, we would express a relation that contains all factors that are squares with $R(x) \leftarrow x \doteq yy$. In core spanners, we would express a relation that contains the positions of all factors that are squares with $\pi_x(\zeta^{=}_{y_1,y_2}(\Sigma^* x\{y_1\{\Sigma^*\}y_2\{\Sigma^*\}\}\Sigma^*))$. See [12] for the full definition of core spanners.

Parallel to this, where model checking Sequence Datalog is undecidable, the same problem for FC-Datalog is in P, and so we can also see FC-Datalog as a tractable recursive query language for strings, independent of the connection to spanners. We aim to identify techniques that make recursion less expensive. We thus look to define restrictions that lead to more efficient fragments of FC-Datalog which also retain other desirable properties.

We focus on the model checking problem primarily through two different lenses: *data complexity* and *combined complexity*. In many cases for query languages it is reasonable to use data complexity as the queries are often significantly smaller than the data. In text-based settings on the other hand, features such as regular expressions can make queries large. Consequently, combined complexity remains an important consideration.

Deterministic Regex. As the regular languages are often not enough to express what is required in practice, almost all modern programming languages (such as e.g. PERL, Python, and Java) do not implement only classical *regular expressions*, as introduced by Kleene [26], but *regex*, regular expressions extended with *back-references*. These are operators that match a repetition of a previously matched string, and whilst they do increase expressibility, they also lead to intractability of membership (see [2]). Freydenberger and Schmid [17] combined regex with the notion of determinism to define DRX, the class of *deterministic regex*, to obtain a tractable class of regex with more expressive power than deterministic regular expressions.

Other Related Models. As well as FC-Datalog, there also exist other related models that capture P. These include positive Range Concatenation Grammars (PRCG) (see e.g. [8]), and Hereditary Elementary Formal Systems (HEFS) (see e.g. [31]), which do not have finite model semantics. The key difference of these models to FC-Datalog is in FC-Datalog's use of word equations, which are not present in other formalisms and are crucial for our restrictions that lead to efficient fragments.

Contributions of this Paper. The only previously known complexity result for FC-Datalog is that it captures P. We first show that combined complexity of FC-Datalog is EXP-complete, and then perform an evaluation of different restrictions on FC-Datalog to identify fragments: with more efficient complexity of model checking, for both data and combined complexity; that do not overly sacrifice expressive power; and where membership in the fragment can be checked efficiently. As the semi-positive linear fragment of classical Datalog captures NLOGSPACE (on ordered structures), our first restriction is to adapt linearity to FC-Datalog. We show that *linear* FC-Datalog also captures NLOGSPACE and has PSPACE-complete combined complexity. Our second restriction is to remove nondeterminism. Here, we define *deterministic linear* FC-Datalog which captures LOGSPACE. But, checking whether a linear program is deterministic is as hard as satisfiability for word equations, a problem which is known to be NP-hard (see [4, 27]) and in nondeterministic linear space (see [23]). Therefore, we employ another restriction that we call *one letter lookahead* (OLLA) on the permitted word equations. Deterministic OLLA (DOLLA) FC-Datalog captures LOGSPACE and checking if an OLLA program is deterministic can be done in polynomial time, but its combined complexity is still PSPACE-complete. We thus make a final restriction that we call *strictly decreasing* (SD) and define SD-DOLLA FC-Datalog, which has linear combined complexity.

We therefore establish the endpoints of a range of fragments that all capture LOGSPACE; at one end is deterministic linear FC-Datalog at at the other is DOLLA FC-Datalog. We establish a trade-off in this range between how rich the fragment's syntax is and how easy it is to check membership in the fragment, although fully mapping this range is left for future work. Furthermore, we show that we can obtain an FC-Datalog fragment with linear combined complexity, namely SD-DOLLA FC-Datalog. Consequently, we have paved the way to design tailored fragments for particular applications.

We then explain how we can view FC-Datalog programs from our range as generalized nondeterministic and deterministic multi-headed two-way finite automata, which are equivalent to nondeterministic and deterministic LOGSPACE Turing machines, respectively (see [25]). FC-Datalog fragments from our range allow for more flexibility than these automata models, for example DOLLA+ FC-Datalog, tailored to simulate DRX, can be viewed as a generalization that permits performing nonregular string computations in the transitions.

Where in [17], to check if a regex matches a word we have to construct a technically involved automata model, we show that we can model this simply in SD-DOLLA+ FC-Datalog. We also show how this tailored fragment allows us to conveniently and naturally write programs that are more concise. DOLLA+ FC-Datalog permits additional deterministic components, but despite this maintains all of the desirable properties of DOLLA FC-Datalog: it captures LOGSPACE, determinism can be checked in polynomial time, and its strictly decreasing variant SD-DOLLA+ FC-Datalog has linear combined complexity. Finally, we show that as we can simulate DRX, another example of deterministic components that can be added to these tailored fragments are constraints that match DRX.

2 Preliminaries

Let $\mathbb{N} := \{0, 1, 2, \ldots\}$ and $\mathbb{N}_+ := \{1, 2, \ldots\}$. We denote the *empty set* by $\emptyset$ and the *cardinality* of a set S by $|S|$. Let $S \setminus S'$ be the *set difference* of S and S', and let $\mathcal{P}(S)$ be the *power set* of S. Let A be some alphabet. For any two $u, v \in A^*$, we use $u \cdot v$, or simply uv, for the *concatenation* of u and v. Let $|u|$ denote the length of $u \in A^*$. Let Σ be a finite alphabet that we call *terminal symbols* (or just *terminals*). We call any $w \in \Sigma^*$ a *word* (or *string*). We use ε to denote the *empty word* and let $\Sigma^+ = \Sigma^* \setminus \{\varepsilon\}$. A *factor* of a word w is a word t such that there exist $u, v \in \Sigma^*$ with $w = u \cdot t \cdot v$. In literature, a factor is sometimes called a contiguous subword.

For a tuple $\vec{t}$, let its *size* $\left|\vec{t}\right|$ be the number of components in $\vec{t}$ and $x \in \vec{t}$ denote that x is a component of $\vec{t}$. We refer to tuples where $\left|\vec{t}\right| = 1$ as *singletons* and omit the brackets in this case. A *relation* is a set of tuples of the same size, and is represented by a *relation symbol*.

Let Ξ be a countably infinite set of *variables* where $\Sigma \cap \Xi = \emptyset$. A *pattern* is a word $\alpha \in (\Sigma \cup \Xi)^*$. We define $\mathrm{Var}(\alpha)$ as the set of variables in α. For two alphabets A and B, a *homomorphism* is a function $g : A^* \to B^*$ where $g(u) \cdot g(v) = g(u \cdot v)$ holds for all $u, v \in A^*$. A *pattern substitution* θ is a homomorphism $\theta : (\Sigma \cup \Xi)^* \to (\Sigma \cup \Xi)^*$ where $\theta(\mathsf{a}) = \mathsf{a}$ for all $\mathsf{a} \in \Sigma$. We usually refer to a pattern substitution as simply a *substitution*. We denote the image of a pattern α under a substitution θ by $\theta(\alpha)$. If $\theta(x) = \varepsilon$ for some variable x, we say θ is *erasing*. For a tuple $\vec{t} = (\alpha_1, \ldots, \alpha_n)$ of patterns $\alpha_1, \ldots, \alpha_n$ and a substitution θ, let $\theta(\vec{t}) = (\theta(\alpha_1), \ldots, \theta(\alpha_n))$.

For two patterns $\alpha, \beta \in (\Sigma \cup \Xi)^*$, an equation of the form $\alpha \doteq \beta$ is called a *word equation*. Under a substitution θ, the word equation $\varphi := \alpha \doteq \beta$ holds if $\theta(\alpha) = \theta(\beta)$, and we then say θ is a *solution* of φ. If there exists a solution for φ, then we say φ is *satisfiable*. We say a conjunction of two word equations $\varphi_1 \wedge \varphi_2$ is satisfiable if there exists a substitution θ that is a solution to both φ_1 and φ_2. We say two word equations φ_1 and φ_2 *contradict* each other if their conjunction is not satisfiable. We say a *pattern equation* is a word equation of the form $x \doteq \alpha$, where $x \in \Xi$ and $\alpha \in (\Sigma \cup \Xi)^*$. For every pattern equation $\varphi = x \doteq \alpha$, we define $\mathrm{Var}(\varphi) := \{x\} \cup \mathrm{Var}(\alpha)$. The expressive power of conjunctions of pattern equations is the same as that for conjunctions of word equations as we can simulate a word equation $\alpha_1 \doteq \alpha_2$, for $\alpha_1, \alpha_2 \in (\Sigma \cup \Xi)^*$, using two pattern equations $x \doteq \alpha_1$ and $x \doteq \alpha_2$ for a new variable $x \in \Xi$.

FC–Datalog. An FC-Datalog *atom* is either a pattern equation, or a so-called *relation atom* of the form $R(y_1, \ldots, y_{\text{ar}(R)})$ where R is a relation symbol that has an arity $\text{ar}(R) \geq 0$ and $y_1, \ldots, y_{\text{ar}(R)} \in \Xi$. Without loss of generality, we can assume that for every pattern equation atom $x \doteq \alpha$, the variable x does not occur in α because such equations reduce to trivial scenarios, as in Lemma 3.1 of [37]. For a relation atom $\varphi = R(y_1, \ldots, y_{\text{ar}(R)})$, we define $\text{Var}(\varphi) = \{y_1, \ldots, y_{\text{ar}(R)}\}$. For now, we assume each relation symbol R has a corresponding relation R. How relations are updated is specified in the following semantics.

Let $w \in \Sigma^*$ be a word and let θ be a substitution. For a pattern equation atom $\varphi = x \doteq \alpha$, we say $(w, \theta) \models \varphi$ if $\theta(x) = \theta(\alpha)$ and both $\theta(x)$ and $\theta(\alpha)$ are factors of w. For a relation atom $\varphi = R(y_1, \ldots, y_{\text{ar}(R)})$, we say $(w, \theta) \models \varphi$ if $(\theta(y_1), \ldots, \theta(y_{\text{ar}(R)})) \in R$ and $\theta(y_1), \ldots, \theta(y_{\text{ar}(R)})$ are factors of w. We represent the word w that defines the universe with the distinguished universe variable $\mathfrak{u}$, and use this as an input to an FC-Datalog program.

Let σ be a relational vocabulary (see e.g. [11, 28] for a definition of FO and vocabularies). A *conjunctive query* is an $\text{FO}[\sigma]$ formula of the form $\rho(\vec{x}) \coloneqq \exists \vec{y}\colon \bigwedge_{i=1}^{n} \eta_i$, where $\eta_i \coloneqq R_i(\vec{t_i})$, each R_i is a relation symbol in σ, each $\vec{t_i}$ is a σ-term, and $n \geq 1$. We usually write this as $\rho \coloneqq \text{Ans}(\vec{x}) \leftarrow \bigwedge_{i=1}^{n} \eta_i$ where $\vec{x}$ is the tuple of free variables. We call $\text{Ans}(\vec{x})$ the *head* of ρ and $\bigwedge_{i=1}^{n} \eta_i$ the *body* of ρ. If there are no free variables then ρ is *Boolean*. We call the set of conjunctive queries CQ. We also use the output relation symbol Ans in FC-Datalog.

▶ **Definition 2.1.** *An* FC-Datalog *program is a 3-tuple* $P \coloneqq (\mathfrak{u}, \mathcal{R}, \Phi)$*, where:*

- $\mathfrak{u} \in \Xi$ *is the universe variable,*
- $\mathcal{R} \supseteq \{\text{Ans}\}$ *is a finite set of relation symbols, where each* $R \in \mathcal{R}$ *has an arity* $ar(R) \geq 0$*,*
- Φ *is a finite set of rules of the form* $R(x_1, \ldots, x_{ar(R)}) \leftarrow \varphi_1, \ldots, \varphi_m$ *for some* $m \geq 1$*, some* $R \in \mathcal{R}$*, where for* $1 \leq i \leq m$ *every* φ_i *is an* FC-Datalog *atom, and for* $0 \leq j \leq ar(R)$ *every* x_j *occurs in some* $\text{Var}(\varphi_i)$*.*

Each element of Φ can be intuitively seen as a conjunctive query over FC-Datalog atoms and as for CQ, when $\text{ar}(\text{Ans}) = 0$, the program is Boolean. For brevity, just Φ is used to represent the whole tuple if $\mathfrak{u}$ and $\mathcal{R}$ are clear from context. For a relation symbol $R \in \mathcal{R}$, let $\Phi_R \subseteq \Phi$ be the subset of rules with head relation symbol R. For an FC-Datalog rule ρ we define $\mathsf{pe}(\rho)$ as the conjunction of all the pattern equations in ρ.

▶ **Example 2.2.** For the FC-Datalog program $P = (\mathfrak{u}, \{\text{Ans}, E\}, \Phi)$ where Φ are the rules defined in Example 1.1, we have the two subsets: $\Phi_{\text{Ans}} \coloneqq \{\text{Ans}() \leftarrow \mathfrak{u} \doteq yz,\ E(y, z)\}$ and $\Phi_E \coloneqq \Phi \setminus \Phi_{\text{Ans}}$. Let $\rho = E(x, y) \leftarrow x \doteq \mathsf{a}u,\ y \doteq \mathsf{b}v,\ E(u, v)$. Then $\mathsf{pe}(\rho) = x \doteq \mathsf{a}u \wedge y \doteq \mathsf{b}v$.

Let $\rho = R(x_1, \ldots, x_{\text{ar}(R)}) \leftarrow \varphi_1, \ldots, \varphi_m$ be an FC-Datalog rule and let $V \coloneqq \bigcup_{i=1}^{m} \text{Var}(\varphi_i)$. For a word $w \in \Sigma^*$, a *w-substitution* is a substitution θ where $\theta(\mathfrak{u}) = w$ and $\theta(x)$ is a factor of w for each $x \in V$. Using only w-substitutions ensures that the universe is restricted to w and its factors. For the rule ρ and w-substitution θ, we say $(w, \theta) \models \rho$ if for all $1 \leq i \leq m$ we have $(w, \theta) \models \varphi_i$. As we only consider the finite universe setting here, we are thus only considering w-substitutions, and so refer to these as just substitutions for brevity.

We treat an FC-Datalog program $P = (\mathfrak{u}, \mathcal{R}, \Phi)$ as implicitly defining a vocabulary and define corresponding structures. An FC-Datalog structure $\mathfrak{A}_P$ consists of a fixed universe $w \in \Sigma^*$ and all its factors, and an interpretation function $f^{\mathfrak{A}_P}$ that maps every relation symbol $R \in \mathcal{R}$ to an interpretation $R^{\mathfrak{A}_P}$. For convenience, we also refer to $R^{\mathfrak{A}_P}$ as R.

A program P and a word w define a structure $[\![P]\!](w)$ incrementally. First, all $R \in \mathcal{R}$ are initialized to $\emptyset$. Then, for each rule $R(x_1, \ldots, x_{\text{ar}(R)}) \leftarrow \varphi_1, \ldots, \varphi_m$ and $V \coloneqq \bigcup_{i=1}^{m} \text{Var}(\varphi_i)$, for every w-substitution θ where $(w, \theta) \models \varphi_i$, for all $1 \leq i \leq m$, add $(\theta(x_1), \ldots, \theta(x_{\text{ar}(R)}))$ to R. We repeat this until all $R \in \mathcal{R}$ have stabilized. Then $[\![P]\!](w)$ is the content of the Ans relation. This "filling up" of relations mirrors the fixed point semantics of classical Datalog.

The defined function P that maps w to ar(Ans)-ary tuples over factors of w is well-defined (thus the fixed point is unique) and the "filling up" process terminates for every given w. If P is Boolean, $[\![P]\!](w)$ is either $\emptyset$ or $\{()\}$. We interpret these as Reject and Accept (resp.) and use this to define a language $\mathcal{L}(P)$.

▶ **Example 2.3.** The FC-Datalog program $P = (\mathfrak{u}, \{\text{Ans}, R\}, \Phi)$ where Φ is:

$$\text{Ans}() \leftarrow \mathfrak{u} \doteq yy,\ \ R(y); \qquad R(x) \leftarrow x \doteq y\mathsf{a},\ \ R(y);$$
$$R(x) \leftarrow x \doteq \varepsilon; \qquad R(x) \leftarrow x \doteq y\mathsf{b},\ \ R(y).$$

defines the language $\mathcal{L}(P) := \{v \cdot v \mid v \in \{\mathsf{a}, \mathsf{b}\}^*\}$. Note that without the three rules that have R in the head, we would define the language $\{v \cdot v \mid v \in \Sigma^*\}$.

We look at the *model checking* problem for FC-Datalog: given a Boolean FC-Datalog program P and a word w, is $w \in \mathcal{L}(P)$? We consider three perspectives: *data complexity*, where the program P is fixed and only the word w is considered input, *expression complexity*, where the word w is fixed and only the program P is considered input, and *combined complexity*, where both the word w and program P are considered input (for details, see [28]).

We call a subset of FC-Datalog programs a *fragment*. We say that a fragment $\mathcal{F}$ *captures* a complexity class $\mathbb{C}$ if $\mathbb{C} = \{\mathcal{L}(P) \mid P \in \mathcal{F}\}$. Note that, following directly from the definitions, if $\mathcal{F}$ captures $\mathbb{C}$, then the data complexity of $\mathcal{F}$ is $\mathbb{C}$. The only complexity result that is known for FC-Datalog is that FC-Datalog captures P (see Theorem 4.11 of [16]), however, there are results for other versions of FC, such as those discussed in Section 1.

3 Efficient FC-Datalog

As P is often not considered efficient for data complexity, this presents an opportunity for optimization. We show that this is also the case for combined complexity.

▶ **Theorem 3.1.** *Combined complexity of* FC-Datalog *is* EXP*-complete.*

It is straightforward to see that as our universe is finite, the number of tuples we can add to the relations is exponential. We therefore have a naive evaluation algorithm of a loop with an exponential bound on the number of iterations, and an exponential number of steps in each iteration. We can also straightforwardly reduce classical Datalog evaluation to show EXP-hardness. Our main goal in this section is to identify fragments with lower data and combined complexity, and where membership in the fragment can be easily checked.

3.1 Linearity

There exists a fragment of classical Datalog that captures the complexity class NLOGSPACE on ordered structures (see e.g. [20]), namely the fragment of all semi-positive *linear* programs. As discussed in Section 1, there is a more general definition for linearity (see e.g. [1]) that does not affect complexity. Here we translate the more general linearity restriction from the relational setting to the text setting.

We define a relation $R \leftarrow R'$ over relation symbols R and R' if there exists a rule ρ where R is the head relation symbol and R' appears in the body. We denote the transitive closure of $\leftarrow$ by $\overset{+}{\leftarrow}$, and two relation symbols R and R' are *mutually recursive* if $R \overset{+}{\leftarrow} R'$ and $R' \overset{+}{\leftarrow} R$.

▶ **Definition 3.2.** *Let P be an* FC-Datalog *program with rule set Φ. A rule $\rho \in \Phi$ is* linear *with respect to P if at most one atom in the body of ρ has a relation symbol with which the head relation symbol of ρ is mutually recursive. If every $\rho \in \Phi$ is linear, then P is* linear.

Checking if a given FC-Datalog program is linear can be done in polynomial time with respect to the program's number of rules, as it amounts to determining which pairs of relation symbols are mutually recursive, which is a syntactic criterion.

▶ **Example 3.3.** The FC-Datalog programs given in Example 1.1 and Example 2.3 are both linear FC-Datalog programs. An example FC-Datalog program that is not linear is the following program that retrieves all even length factors of the input word:

$$\text{Ans}(z) \leftarrow z \doteq xy,\ \ \text{Ans}(x),\ \ \text{Ans}(y);$$
$$\text{Ans}(z) \leftarrow z \doteq xy,\ \ L(x),\ \ L(y);$$

and a rule $L(x) \leftarrow x \doteq \mathsf{a}$ for each $\mathsf{a} \in \Sigma$. This is not linear as Ans is mutually recursive with Ans, and the top rule has Ans in the head and two occurrences of Ans in the body.

While unrestricted FC-Datalog captures P, the restriction to linear FC-Datalog has substantially improved data complexity (Theorem 3.4) as well as combined complexity (Theorem 3.5).

▶ **Theorem 3.4.** *Linear* FC-Datalog *captures* NLOGSPACE.

We capture NLOGSPACE as we can simulate multi-headed two-way nondeterministic finite automata, which are equivalent to nondeterministic Turing machines with logarithmic space (see [25]). We will further discuss the connection to automata in Section 3.5.

▶ **Theorem 3.5.** *Combined and expression complexity of linear* FC-Datalog *are* PSPACE*-complete.*

On the other hand, we would like to lower both data complexity further than NLOGSPACE and combined complexity further than PSPACE. Furthermore, PSPACE-completeness occurs even on a single-character input. As such, we look for more efficient fragments.

3.2 Determinism

As linear FC-Datalog captures NLOGSPACE, it is natural to look at causes of nondeterminism and see if there is a corresponding fragment that captures LOGSPACE. For model checking, we evaluate FC-Datalog programs top-down.

▶ **Example 3.6.** We show a top-down evaluation of the program P in Example 2.3 on the input word $w = \mathsf{abab}$. The only rule we can apply first is $\text{Ans}() \leftarrow \mathfrak{u} \doteq yy,\ \ R(y)$. As $\theta(\mathfrak{u}) = w$, we see that $\theta(y) = \mathsf{ab}$, and we pass this into the relation R. We can then only apply the rule $R(x) \leftarrow x \doteq y\mathsf{b},\ \ R(y)$. As $\theta(x) = \mathsf{ab}$, we have $\theta(y) = \mathsf{a}$ and we recurse on this value of y. We then apply $R(x) \leftarrow x \doteq y\mathsf{a},\ \ R(y)$ and recurse on $\theta(y) = \varepsilon$. We can then apply $R(x) \leftarrow x \doteq \varepsilon$, and as this holds, we accept.

Based on this top-down evaluation model, we define two categories of variables.

▶ **Definition 3.7.** *Let $\rho = R_1(x_1, \dots x_m) \leftarrow R_2(y_1, \dots y_n), \varphi_1, \dots, \varphi_\ell$ be a linear* FC-Datalog *rule where R_1 is mutually recursive with R_2, and φ_i is either a pattern equation or a relation atom with a relation symbol that is not mutually recursive with R_1, for $1 \leq i \leq \ell$. We say ρ's* top variables $\mathsf{top}(\rho)$ *are* $(x_1, \dots, x_m, \mathfrak{u})$ *and ρ's* bottom variables $\mathsf{bottom}(\rho)$ *are* $(y_1, \dots, y_n)$.

In top-down evaluation, a rule's top variables contain values that were passed down from a preceding rule's evaluation, and its bottom variables contain the values that are passed down. A variable can be both a top and bottom variable.

▶ **Example 3.8.** Let ρ be the linear FC-Datalog rule $R(x) \leftarrow x \doteq y\mathsf{a},\ R(y)$ from Example 2.3. Then $\mathsf{top}(\rho) = (x, \mathfrak{u})$ and $\mathsf{bottom}(\rho) = (y)$.

When evaluating a linear program, nondeterminism can occur in two ways: in choosing the values of the variables when processing a rule, which we call *local nondeterminism*, and in choosing which rule to process, which we call *global nondeterminism*. In order to remove all nondeterminism, we must ensure every program has both *local* and *global determinism*.

Let w be the input word and let ρ be a linear FC-Datalog rule. We define the relation W_ρ to be all pairs $(\theta(\mathsf{top}(\rho)), \theta(\mathsf{bottom}(\rho)))$, for all substitutions θ such that $(w, \theta) \models \mathsf{pe}(\rho)$. For some $(\vec{t}, \vec{t'}) \in W_\rho$, we call $\vec{t}$ a *top tuple* and $\vec{t'}$ a *bottom tuple*. Recall that as that we are working under finite model semantics, each value is a factor of the word w. We define two criteria for determinism.

▶ **Definition 3.9.** *A linear* FC-Datalog *program P with rule set Φ is* locally deterministic *if for each $\rho \in \Phi$, the relation W_ρ is a partial function. P is* globally deterministic *if for every relation symbol R, for all pairs of distinct rules $\rho, \chi \in \Phi_R$, there is no top tuple $\vec{t}$ such that $(\vec{t}, \vec{t'})$ in W_ρ and $(\vec{t}, \vec{t''})$ in W_χ, for some bottom tuples t' and t''.*

If W_ρ is a partial function for some rule ρ, then top-down evaluation of ρ is locally deterministic as for every top tuple there is at most one bottom tuple. If a top tuple is a valid input for only one rule ρ in Φ_R, for ρ's head relation symbol R, then we have global determinism as ρ is the only rule we can process.

▶ **Definition 3.10.** Deterministic linear FC-Datalog *is the set of linear* FC-Datalog *programs that are locally and globally deterministic.*

Our condition is sufficiently strong as we precisely capture LOGSPACE.

▶ **Theorem 3.11.** *Deterministic linear* FC-Datalog *captures* LOGSPACE.

We capture LOGSPACE as we can simulate multi-headed two-way deterministic finite automata, which are equivalent to deterministic Turing machines with logarithmic space (see [25]). Unfortunately, where linearity for FC-Datalog can be checked in polynomial time, checking determinism for linear FC-Datalog is considerably more expensive, as the criteria we have to check is semantic rather than syntactic. A problem p is *word equations-hard* if there exists a polynomial time reduction from word equation satisfiability to p.

▶ **Proposition 3.12.** *Checking local or global determinism of linear* FC-Datalog *is word equations-hard.*

Word equation satisfiability is NP-hard [4, 27] and in nondeterministic linear space [23]. Closing this gap is a longstanding open problem. Thus, although we have reduced the complexity, we have lost efficient checking of membership in the fragment.

We now show that, in contrast to regular spanners, under standard complexity theoretic assumptions we cannot expect to have constant delay enumeration algorithms in our setting. A constant delay enumeration algorithm has a preprocessing phase, which often runs in linear time, and an enumeration phase that outputs solutions one by one, with constant time between any consecutive outputted solutions (see e.g. [36]).

▶ **Proposition 3.13.** *If there exists a constant delay enumeration algorithm with polynomial preprocessing for deterministic linear* FC-Datalog*, then* P = NP.

As such, we do not look further into constant delay enumeration, and instead focus on identifying a fragment that has both reduced complexity and efficient membership checking, without compromising on expressive power.

3.3 One Letter Lookahead

In this subsection, we introduce a second fragment that captures LOGSPACE, but where we can check membership in the fragment in polynomial time. This fragment has a severely restricted syntax but can still simulate multi-headed two-way deterministic finite automata, demonstrating how little of FC-Datalog is required to retain this level of expressibility.

▶ **Definition 3.14.** *Let ρ be a linear* FC-Datalog *rule. Let $x, \in \mathsf{top}(\rho)$, $y \in \mathsf{bottom}(\rho)$, and $\mathsf{a} \in \Sigma \cup \{\varepsilon\}$. The rule ρ is a* one letter lookahead (OLLA) FC-Datalog *rule if each pattern equation has either of the below forms:*

- $x \doteq \varepsilon$ *or* $x \doteq x'$ *where* $x' \in \mathsf{top}(\rho)$.
- $x \doteq y\mathsf{a}$ *or* $x \doteq \mathsf{a}y$ *(deleting a letter from a top variable).*
- $y \doteq x\mathsf{a}$ *or* $y \doteq \mathsf{a}x$ *(adding a letter to a top variable).*
- $\mathfrak{u} \doteq x\mathsf{a}z$ *for* $z \in \Xi$ *where* $z \notin \mathsf{top}(\rho) \cup \mathsf{bottom}(\rho)$ *and z does not occur elsewhere in ρ (matching the next letter after a top variable in the input word).*
- $\mathfrak{u} \doteq z\mathsf{a}x$ *for* $z \in \Xi$ *where* $z \notin \mathsf{top}(\rho) \cup \mathsf{bottom}(\rho)$ *and z does not occur elsewhere in ρ (matching the first letter before a top variable in the input word).*

We call an equation $x \doteq \varepsilon$ or $x \doteq x'$ or where $\mathsf{a} = \varepsilon$ an ε-OLLA pattern equation. Otherwise, we call an equation where a is on the right side of x or y a *right* OLLA pattern equation, and an equation where a is on the left side of x or y a *left* OLLA pattern equation.

▶ **Definition 3.15.** *Let P be a linear* FC-Datalog *program with rule set Φ. We say P is a one letter lookahead* (OLLA) FC-Datalog *program if all $\rho \in \Phi$ are* OLLA FC-Datalog *rules, and for every relation symbol R, for every rule $\rho \in \Phi_R$, every variable $x \in \mathsf{top}(\rho) \cup \mathsf{bottom}(\rho)$ is not used in both left* OLLA *and right* OLLA *pattern equations. An* OLLA FC-Datalog *program is* deterministic (a DOLLA FC-Datalog program) *if it is both locally and globally deterministic.*

As such, in DOLLA FC-Datalog, for every rule $\rho \in \Phi_R$ we can only use equations of the form $x \doteq x'$ for $x, x' \in \mathsf{top}(\rho)$ if $|\Phi_R| = 1$ or there exists a equation of the form $x = \varepsilon$ or $x = \mathsf{a}y$ or $x = y\mathsf{a}$ in $\mathsf{pe}(\rho)$, for $y \in \mathsf{bottom}(\rho)$. We call such a program *guarded*.

▶ **Example 3.16.** The linear FC-Datalog program P given in Example 2.3 is not a DOLLA FC-Datalog program as the rule $\mathrm{Ans}() \leftarrow \mathfrak{u} \doteq yy,\ R(y)$ contains $\mathfrak{u} \doteq yy$, which does not fit any of the permitted forms of pattern equations for an OLLA FC-Datalog program.

The simplest DOLLA FC-Datalog program the authors could find that models the language of squares given in Example 2.3 is the one that models the multi-headed two-way DFA; this program first identifies the middle of the string and processes the two halves letter by letter. We now see that despite restricting the programs substantially, our new fragment DOLLA FC-Datalog retains the expressive power of deterministic linear FC-Datalog.

▶ **Theorem 3.17.** DOLLA FC-Datalog *captures* LOGSPACE.

To check if a program is an OLLA FC-Datalog program is straightforward. Checking if an OLLA FC-Datalog program is deterministic is easier than in the general case, as the pattern equations are restricted to only match one letter at a time. When checking both local and global determinism, we only need to consider the case where rules do not contain contradicting pattern equations; if we have a rule with two pattern equations φ_1 and φ_2 that contradict each other, then as there are no solutions to $\varphi_1 \wedge \varphi_2$, this rule can never be applied. To check if a program is locally deterministic, we check that for every rule ρ without pattern equations that contradict each other, every $x \in \mathsf{top}(\rho)$ appears in some pattern equation

in $\mathsf{pe}(\rho)$ or $x \in \mathsf{bottom}(\rho)$, and every $y \in \mathsf{bottom}(\rho)$ appears in some pattern equation in $\mathsf{pe}(\rho)$ or $y \in \mathsf{top}(\rho)$. If so, then W_ρ is a partial function. To check if a program is globally deterministic, we use *profiles* for its rules.

▶ **Definition 3.18.** *Let ρ be an* OLLA FC-Datalog *rule without pattern equations that contradict each other. We define a* profile *for ρ as a function $pro_\rho \colon \mathsf{top}(\rho) \to \Sigma \cup \{\varepsilon, \bot\}$ where:*

$$pro_\rho(x) = \begin{cases} \mathsf{a} \text{ if there exists a pattern equation } x \doteq y\mathsf{a} \text{ or } x \doteq \mathsf{a}y, \text{ for } \mathsf{a} \in \Sigma, \\ \varepsilon \text{ if there exists a pattern equation } x \doteq \varepsilon, \\ \bot \text{ otherwise.} \end{cases}$$

For $p, p' \in \Sigma \cup \{\varepsilon, \bot\}$, we say p and p' are in conflict *if $p \neq \bot$ and $p' \neq \bot$, and $p \neq p'$. Let ρ and χ be linear* FC-Datalog *rules with the same head relation symbol, with respective profiles pro_ρ and pro_χ. Let $\mathsf{top}(\rho) = (x_1, \ldots, x_k)$. Note that $\mathsf{top}(\rho) = \mathsf{top}(\chi)$. We say pro_ρ and pro_χ are* in conflict *if there exists some $x \in \mathsf{top}(\rho)$ where $pro_\rho(x)$ is in conflict with $pro_\chi(x)$.*

The function pro_ρ represents how each $x \in \mathsf{top}(\rho)$ is processed in ρ. In OLLA FC-Datalog, all we need to consider is the leftmost or rightmost letter of each x, depending on whether x is used in left or right OLLA pattern equations for rules with this head relation symbol.

▶ **Lemma 3.19.** *A guarded* OLLA FC-Datalog *program P is globally deterministic if and only if for every relation symbol R, for every pair of distinct rules $\rho, \chi \in \Phi_R$, we have that pro_ρ is in conflict with pro_χ.*

This allows us to check if a given program is deterministic efficiently.

▶ **Proposition 3.20.** *Local and global determinism for* OLLA FC-Datalog *can be decided in polynomial time.*

We thus have a fragment where we have reduced data complexity to LOGSPACE, and without losing efficient checking of membership in the fragment. However, the combined complexity for DOLLA FC-Datalog is the same as for linear FC-Datalog.

▶ **Theorem 3.21.** *Combined complexity of* DOLLA FC-Datalog *is* PSPACE*-complete.*

3.4 Strictly Decreasing

Our next goal is more efficient combined complexity. We introduce a further restriction called *strictly decreasing*, and show that this leads to linear combined complexity.

▶ **Definition 3.22.** *Let P be a* DOLLA FC-Datalog *program with rule set Φ, and let $\Phi' \subseteq \Phi$ be the rules containing both an atom with a relation symbol mutually recursive with the head relation symbol, and at least one pattern equation. Let w be the input word. We say P is* strictly decreasing (SD) *if for every rule $\rho \in \Phi'$ that has head relation symbol R:*

1. *there exist $x \in \mathsf{top}(\rho)$ and $y \in \mathsf{bottom}(\rho)$ such that y occurs in a pattern equation with x and for all substitutions θ such that $(w, \theta) \models \rho$ we have $|\theta(y)| < |\theta(x)|$, and*
2. *if there exists a relation symbol R' in the body of ρ that is mutually recursive with R, for every rule $\chi \in \Phi'$ with head relation symbol R', condition 1 holds for some $x' \in \mathsf{top}(\chi)$ and some $y' \in \mathsf{bottom}(\chi)$, and $\theta(y) = \theta'(x')$ for all substitutions θ such that $(w, \theta) \models \rho$ and all substitutions θ' such that $(w, \theta') \models \chi$.*

We now see that this restriction significantly improves the combined complexity. In fact, if the maximum relation symbol arity k is fixed (or assumed to be much smaller than $|w|$), then this is linear. Furthermore, the preprocessing is not dependent on the word w.

▶ **Theorem 3.23.** *Given a word w and an* SD-DOLLA FC-Datalog *program P with n relation symbols and maximum relation symbol arity k, we can decide $w \in \mathcal{L}(P)$ in $\mathcal{O}(|w|k)$ time after $\mathcal{O}(n|\Sigma|)$ preprocessing.*

In this section, we have defined the endpoints of an infinite range of fragments that all capture LOGSPACE. At one end, DOLLA FC-Datalog has just enough syntax to simulate multi-headed two-way deterministic finite automata, but has easy checking of determinism. At the other end, deterministic linear FC-Datalog has a richer syntax but has more difficult checking of determinism. From this range, there is therefore the opportunity to extend DOLLA FC-Datalog to make writing programs more natural and convenient. We will demonstrate such a situation in Section 4 when designing an appropriate fragment to model deterministic regex. Furthermore, although the fragments in this range do not reduce combined complexity any more than is the case for linear FC-Datalog, when we add the extra dimension of strictly decreasing, we can reduce the combined complexity to linear time.

3.5 FC-Datalog as Generalized Automata

As DOLLA FC-Datalog, by design, permits just enough to simulate multi-headed two-way deterministic finite automata, we can naturally see a DOLLA FC-Datalog program as a generalization of such an automaton: every relation acts as a state, every rule acts as a transition, and every string variable acts a head which can be moved, added and deleted (as the number of heads can change per state). In each transition we read one letter and move, add, or delete the heads accordingly.

The fragments of FC-Datalog discussed in this paper can be seen as further generalizations. As the syntax grows, the connection to automata becomes less natural and checking determinism becomes harder. For example, Section 4 introduces DOLLA+ FC-Datalog which allows word equations such as $x \doteq yy$. Instead of just moving a head by one letter, this is an operation that either splits a memory in half or doubles it. We can then see a DOLLA+ FC-Datalog program as an generalized multi-headed two-way DFA where heads can read words rather than letters and that can perform nonregular string computations in the transitions.

We can also see an OLLA FC-Datalog program as a generalized multi-headed two-way NFA, and a linear FC-Datalog program as a further generalization. There, reading letters at head positions is replaced with a declarative programming language that checks if the rule applies and computes the operations. The increasing gap to traditional automata is reflected in the fact that determinism is now expensive to verify. Finally, for FC-Datalog, we can understand every relation symbol in the body of a rule as a call of a subroutine, which splits the automaton into parallel copies, each of which must terminate. At this general level, however, the gap to traditional automata is so large the connection is no longer natural.

4 Applying the Framework: Simulating DRX

In this section, we tailor a fragment from our range to a specific application. We apply our model to simulate deterministic regex, introduced by Freydenberger and Schmid [17].

We define regular expressions as usual (see e.g. [1]) and extend these with back-references to define *regex*. For $x \in \Xi$ and a regular expression δ, we thus add the expressions x and $\langle x\colon \delta\rangle$ to our syntax. The expression $\langle x\colon \delta\rangle$ matches δ and saves the string that is matched by δ in the *memory* x. All further occurrences of x are *recalls* of memory x, which we match using the content of the variable that was saved earlier. For readability, we use $\cdot$ for concatenation.

▶ **Example 4.1.** Let $\gamma \coloneqq \langle x \colon (\mathsf{a} \vee \mathsf{b})^+ \rangle \cdot \mathsf{d} \cdot x$. Then γ matches all words $u\mathsf{d}u$ where $u \in \{\mathsf{a}, \mathsf{b}\}^+$.

Every regular expression can be converted into a finite automaton using the classical construction from Glushkov [19]. If the result of this construction is deterministic then the regular expression is deterministic. Deterministic regular expressions define a strict subclass of the regular languages, and have a more efficient membership problem than general regular expressions. For a deterministic regular expression γ and a word w, we can decide membership in $\mathcal{O}(|\Sigma||\gamma| + |w|)$ (see [9, 34]) or $\mathcal{O}(|\gamma| + |w| \log\log|\gamma|)$ (see [21]).

Freydenberger and Schmid [17] combined regex and deterministic regular expressions to define *deterministic regex*, which can define nonregular languages, and have an efficient membership problem; this can be decided in $\mathcal{O}(|\Sigma||\gamma|n{+}k|w|)$ for a word w and a deterministic regex γ with k distinct variables and n total occurrences of terminal symbols or variable references (see Theorem 5 of [17]). Similarly to the Glushkov [19] construction, every regex γ has a corresponding *memory finite automaton with trap state* (DTMFA) M_γ (where the trap state handles memory recall failures), and γ is deterministic if M_γ is deterministic (see [17] for details). We call the class of all deterministic regex DRX.

▶ **Example 4.2.** Let γ be the regex defined in Example 4.1, and let $\gamma' \coloneqq \langle x \colon (\mathsf{a} \vee \mathsf{b})^* \rangle \cdot x$. Then $\mathcal{L}(\gamma) \coloneqq \{u\mathsf{d}u \mid u \in \{\mathsf{a}, \mathsf{b}\}^*\}$ and $\mathcal{L}(\gamma') \coloneqq \{uu \mid u \in \{\mathsf{a}, \mathsf{b}\}^*\}$. Then $\gamma \in$ DRX and $\gamma' \notin$ DRX, as M_γ is deterministic and $M_{\gamma'}$ is not deterministic. Intuitively, this is because γ has only one choice of when to stop matching the first u, whereas γ' does not.

As in [17], we can assume w. l. o. g. that a regex does not recall empty memories, does not start to save into a memory that is already being saved to, and does not reset a memory to its initial value. We first show that for any $\gamma \in$ DRX, we can express $\mathcal{L}(\gamma)$ in SD-DOLLA FC-Datalog, but with a large number of rules. We can parameterize such a program using the number of memories, terminal symbols and memory recalls in γ. Here we use ε-semantics[1].

▶ **Theorem 4.3.** *Let $\gamma \in$ DRX have k memories. Let the total number of terminal symbols and memory recalls in γ be n. We can express $\mathcal{L}(\gamma)$ with an* SD-DOLLA FC-Datalog *program P that has at most $k + n + 2$ relation symbols and at most $k(|\Sigma| + 1) + n(n + 3) + 1$ rules.*

The program P that is the result of our construction requires a rule for each pair (a, ℓ), for $\mathsf{a} \in \Sigma$ and $0 \leq \ell \leq k - 1$. This is because the syntax permits matching only one letter at a time, and so the program requires many rules to read memories letter by letter. We can achieve this with a much more concise program if we relax the syntax slightly, to what we call DOLLA+ FC-Datalog. This sits between DOLLA FC-Datalog and deterministic linear FC-Datalog, and it retains the polynomial time checking of determinism. To define this, we introduce the concept of a symbol being *uniquely defined* for top-down evaluation.

▶ **Definition 4.4.** *Let ρ be an* FC-Datalog *rule. We inductively define* uniquely defined *symbols: As base rules, the universe variable* $\mathfrak{u}$*, every $\mathsf{a} \in \Sigma$ and every $x \in \mathsf{top}(\rho)$ are uniquely defined. Then, if we have some pattern equation φ where $\mathrm{Var}(\varphi) = \{x_1, \dots x_k\}$ and exactly one variable x_i is not uniquely defined, then x_i becomes uniquely defined.*

Example 3.6 illustrates how top variables are uniquely defined for top-down evaluation.

▶ **Example 4.5.** Let $\rho = R_1(x_1, x_2) \leftarrow \varphi_1, \varphi_2, R_2(y_1, y_2)$ be a linear FC-Datalog rule where $\varphi_1 = x_1 \doteq x_2 y_1$ and $\varphi_2 = y_1 \doteq y_2 y_2$. From the base rules, x_1 and x_2 are uniquely defined. In the first iteration, as all $z \in \mathrm{Var}(\varphi_1) \setminus \{y_1\}$ are uniquely defined, y_1 is uniquely defined. In the second iteration, as all $z \in \mathrm{Var}(\varphi_2) \setminus \{y_2\}$ are uniquely defined, y_2 is uniquely defined.

[1] In ε-semantics we treat everything not initialized as ε (see e.g. Section 8.2.1 in [17]).

In deterministic linear and DOLLA FC-Datalog, by Definition 3.9, we ensure local determinism by requiring the relation W_ρ to be a partial function for every rule ρ, implicitly requiring every variable to be uniquely defined. We ensure local determinism in DOLLA+ FC-Datalog by reasoning directly over uniquely defined variables.

▶ **Definition 4.6.** *Let P be a linear* FC-Datalog *program with rule set Φ. We say P is a* DOLLA+ FC-Datalog *program if it is globally deterministic and for every rule $\rho \in \Phi$, every variable that occurs in ρ is uniquely defined.*

▶ **Example 4.7.** The FC-Datalog program P given in Example 2.3 that is not a DOLLA FC-Datalog program (see Example 3.16) is in fact a DOLLA+ FC-Datalog program.

We define SD for DOLLA+ the same way as for DOLLA (Definition 3.22). SD-DOLLA+ FC-Datalog generalizes SD-DOLLA FC-Datalog. We now show that despite being extensions, DOLLA+ FC-Datalog retains the low data complexity (Theorem 4.8) and the efficient checking of determinism (Proposition 4.9) of DOLLA FC-Datalog, and SD-DOLLA+ FC-Datalog retains the low combined complexity of SD-DOLLA FC-Datalog (Theorem 4.8).

▶ **Theorem 4.8.** DOLLA+ FC-Datalog *captures* LOGSPACE*, and given a word w and an* SD-DOLLA+ FC-Datalog *program P with n relation symbols and maximum relation symbol arity k, we can decide $w \in \mathcal{L}(P)$ in $\mathcal{O}(|w|k)$ time after $\mathcal{O}(n|\Sigma|)$ preprocessing.*

▶ **Proposition 4.9.** *Membership of* DOLLA+ FC-Datalog *can be decided in polynomial time.*

Where a DOLLA FC-Datalog program is required to match letter-by-letter, this is no longer the case for DOLLA+ FC-Datalog, and a consequence of this is that we can simulate a deterministic regex with a much more concise SD-DOLLA+ FC-Datalog program.

▶ **Theorem 4.10.** *Let $\gamma \in$ DRX have k memories. Let the total number of terminal symbols and memory recalls in γ be n. We can express $\mathcal{L}(\gamma)$ with an* SD-DOLLA+ FC-Datalog *program P that has at most $n+2$ relation symbols and at most $n(n+3)+1$ rules.*

As the following example shows, the extra flexibility permitted in the syntax of DOLLA+ FC-Datalog allows us to write programs much more conveniently than for DOLLA FC-Datalog.

▶ **Example 4.11.** In DOLLA+ FC-Datalog, we can process a memory x'_n, with a rule:

$$Q'(u, x'_1, \dots, x'_k) \leftarrow Q(v, x_1, \dots, x_k),\ \ u \doteq x'_n v,\ \ x_1 \doteq x'_1 x'_n,\ \ \dots,\ \ x_k \doteq x'_k x'_n.$$

In DOLLA FC-Datalog, to process a memory x'_n, we use a new relation symbol R_ℓ and rules:

$$Q'(u, x'_1, \dots, x'_k) \leftarrow Q(v, x_1, \dots, x_k),\ \ R_\ell(u, v, x_n, x'_1, x_1, \dots, x'\ell, x_\ell);$$
$$R_\ell(u, v, x'_1, x_1, \dots, x'_\ell, x_\ell) \leftarrow x_n \doteq \varepsilon,\ \ u \doteq v,\ \ x'_1 \doteq x_1,\ \ \dots,\ \ x'_\ell \doteq x_\ell;$$

and for every $\mathsf{a} \in \Sigma$ a rule:

$$R_\ell(u, v, x_n, x'_1, x_1, \dots, x'_\ell, x_\ell) \leftarrow x_n \doteq \mathsf{a}x'_n,\ \ u \doteq \mathsf{a}u',$$
$$x''_1 \doteq x'_1\mathsf{a},\ \ \dots,\ \ x''_\ell \doteq x'_\ell\mathsf{a},\ \ R_\ell(u', v, x'_n, x''_1, x_1, \dots, x''_\ell, x_\ell).$$

We have thus demonstrated the design of a tailored fragment in our range spanned by DOLLA and deterministic linear FC-Datalog, in this case by adding word equations with uniquely defined variables. The DOLLA+ and SD-DOLLA+ fragments retain the desirable properties of the restrictive DOLLA and SD-DOLLA fragments. As discussed in Section 3.5, we can see DOLLA+ FC-Datalog as a generalization of multi-headed DFAs which is then not restricted to left-to-right parsing. As the next example shows, it can express context-free languages which cannot be recognized by a DPDA (see e.g [22]).

▶ **Example 4.12.** The palindrome language can be expressed deterministically by the DOLLA+ FC-Datalog program with the rules $\mathrm{Ans}() \leftarrow R(\mathfrak{u})$; $R(x) \leftarrow x \doteq \varepsilon$ and the rules $R(x) \leftarrow x \doteq \mathsf{a}y\mathsf{a},\ R(y)$; $R(x) \leftarrow x \doteq \mathsf{a}$ for each $\mathsf{a} \in \Sigma$. This evaluates in $\lceil |w|/2 \rceil + 1$ steps, which is linear.

Furthermore, as we can simulate DRX, another feature that can be added to these tailored fragments are atoms that match DRX, which we can solve as a subroutine. We say a DRX-*constraint* is an expression $(x \dot{\in} \gamma)$ for $x \in \Xi$ and a deterministic regex γ, that denotes x is mapped to an element $u \in \mathcal{L}(\gamma)$ and u is a factor of the input word w.

To stay in LOGSPACE, we must ensure programs remain locally and globally deterministic. As DRX-constrains only check if a variable matches a deterministic regex, we can decide local determinism as we would without the DRX-constraints. To retain global determinism (Definition 3.9), if using deterministic regexes $\gamma_1, \ldots, \gamma_n$ in multiple rules with the same head relation symbol, we must ensure that always at most one rule can accept. We thus need to decide if $\bigcap_{i=1}^{n} \mathcal{L}(\gamma_i) = \emptyset$. Unfortunately, intersection-emptiness for DRX is undecidable (see Theorem 9 of [17]). We say $\gamma \in \mathrm{DRX}$ is variable-star-free if each of its sub-regexes under a Kleene-star or Kleene-plus do not contain any variable operations. Intersection-emptiness for variable-star-free DRX is at least word equations-hard (see Proposition 8 of [17]). Thus, adding DRX-constraints means we lose efficient checking of determinism, another example of the trade-off between richer syntax and efficient determinism checking.

We can keep efficient determinism checking if we limit where we include DRX-constraints. If $|\Phi_R| = 1$ for some relation symbol R, global determinism for Φ_R is inherent. If we limit inclusions of DRX-constraints to rules $\rho \in \Phi_R$ where $|\Phi_R| = 1$, we can verify determinism by verifying every DRX-constraint is in such rules. We can decide this in polynomial time. Combining this with Proposition 4.9, we can check the whole program's determinism in polynomial time.

▶ **Example 4.13.** The FC-Datalog program P:

$$\mathrm{Ans}() \leftarrow R_1(\mathfrak{u}); \qquad R_1(x) \leftarrow x \doteq \mathsf{a}y\mathsf{c},\ \ R_1(y);$$
$$R_2(x) \leftarrow (x \dot{\in} \langle y \colon (\mathsf{c} \vee \mathsf{d})^+ \rangle \cdot \mathsf{a} \cdot y); \qquad R_1(x) \leftarrow x \doteq \mathsf{b}y\mathsf{b},\ \ R_2(y).$$

defines the language $\{\mathsf{a}^z\mathsf{b}w\mathsf{a}w\mathsf{b}\mathsf{c}^z \mid z \in \mathbb{N} \wedge w \in \{\mathsf{c},\mathsf{d}\}^*\}$, and $|\Phi_{R_2}| = 1$, where R_2 is the only head relation symbol that has rules containing DRX-constraints.

▶ Remark 4.14. Another extension of FC is FC[REG], which captures the expressive power of generalized core spanners and also has LOGSPACE date complexity. However, FC[REG] does not capture LOGSPACE, as it cannot express the language $\{a^n b^n \mid n \in \mathbb{N}\}$ (see [16]). We can naturally express this language in DOLLA+ FC-Datalog:

▶ **Example 4.15.** We can express $\{a^n b^n \mid n \in \mathbb{N}\}$ with the DOLLA+ FC-Datalog program that has the rules: $\mathrm{Ans}() \leftarrow R(\mathfrak{u})$; $R(x) \leftarrow x \doteq \varepsilon$; and $R(x) \leftarrow x \doteq \mathsf{a}y\mathsf{b},\ R(y)$.

5 Conclusions and Future Work

Freydenberger and Peterfreund [16] recently proposed FC-Datalog as an extension of the logic FC with recursion. FC-Datalog captures the complexity class P and can be seen as a language for expressing relations that can be used in spanner selections. Furthermore, independent of the spanner connection, FC-Datalog can also be seen as a version of Datalog on strings with a decidable model checking problem, where the same problem for a previous approach towards Datalog on strings is undecidable. We first showed that combined complexity of FC-Datalog

is EXP-complete, presenting an opportunity for optimization of model checking. In this paper, we identified fragments of FC-Datalog with an efficient model checking problem for both data and combined complexity by performing an analysis of four restrictions: linearity, determinism, one letter lookahead and strictly decreasing.

In Section 3.1 we proposed linear FC-Datalog, a fragment that captures NLOGSPACE. In Section 3.2 we identified and removed nondeterminism. Thus, deterministic linear FC-Datalog captures LOGSPACE. However, this restriction cannot be checked efficiently.

In Section 3.3 we imposed the one letter lookahead restriction on a program's word equations to obtain deterministic OLLA (DOLLA) FC-Datalog that has both desirable properties: it captures LOGSPACE and determinism can be checked in polynomial time. We also showed that the combined complexity of model checking for both linear FC-Datalog and DOLLA FC-Datalog is PSPACE-complete. Therefore, we added the further restriction of strictly decreasing (SD), and showed that SD-DOLLA FC-Datalog has linear combined complexity. We thus established the endpoints of a range of FC-Datalog fragments that all capture LOGSPACE, and showed how we can restrict this further to reduce combined complexity to linear time. Hence, we have paved the way to construct further fragments which can be tailored for particular applications.

We illustrated tailoring a fragment in Section 4, where we constructed DOLLA+ FC-Datalog. This allows us to straightforwardly model deterministic regex without the need for a technically involved automata model. Yet, as for DOLLA FC-Datalog: we capture LOGSPACE, determinism can be checked in polynomial time, and strictly decreasing programs have linear combined complexity. We then showed how deterministic regex can be used as atoms, and how we can restrict where they are used so as to not compromise on these desirable properties.

As LOGSPACE is closed under complement, we can add stratified negation without affecting the combined complexity. Other convenient additions could include predicates that express properties of a string such as "is a letter", and functions that build on these to express properties such as "is the first/last letter". We could also add regular constraints: atoms that work in the same way as DRX-constraints, but match a classical regular expression rather than a deterministic regex, as in FC[REG]. Using these or other syntactic additions, we could further map the range between DOLLA and deterministic linear FC-Datalog, and investigate how these additions affect checking membership in the fragment. Furthermore, we could also look to find sufficient conditions to ensure membership can be easily checked.

There are structure criteria for conjunctive queries in FC (FC-CQ) that improve efficiency such as acyclicity, as demonstrated by Freydenberger and Thompson [18]. Every FC-Datalog rule can be seen as a conjunctive query in FC. We could therefore add such structure criteria to FC-Datalog rules and examine how this improves the complexity of FC-Datalog fragments further. Another direction for research is to look at inexpressibility for these fragments. However, even for FC (as opposed to FC-Datalog), finding inexpressibility results is challenging (see [38]). Another logical next step is to use our restrictions for FC-Datalog to identify natural fragments in the spanner setting with desirable properties.

Finally, as we can also see FC-Datalog as a generalization of range concatenation grammars (RCG) (see [16]), parsing techniques for these grammars such as the algorithm in Boullier [8] could be adapted to FC-Datalog.

References

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

2 Alfred V. Aho. *Algorithms for Finding Patterns in Strings*, pages 255–300. MIT Press, 1991.

3 Antoine Amarilli, Pierre Bourhis, Stefan Mengel, and Matthias Niewerth. Constant-Delay Enumeration for Nondeterministic Document Spanners. *SIGMOD Rec.*, 49(1):25–32, 2020. doi:10.1145/3422648.3422655.

4 Dana Angluin. Finding Patterns Common to a Set of Strings. *J. Comput. Syst. Sci.*, 21(1):46–62, 1980. doi:10.1016/0022-0000(80)90041-0.

5 Mena Badieh Habib Morgan and Maurice van Keulen. Information Extraction for Social Media. In *Proc. SWAIE*, pages 9–16, 2014. doi:10.3115/v1/W14-6202.

6 Pablo Barceló and Pablo Muñoz. Graph Logics with Rational Relations: The Role of Word Combinatorics. *ACM Trans. Comput. Logic*, 18(2), 2017. doi:10.1145/3070822.

7 Anthony J. Bonner and Giansalvatore Mecca. Sequences, Datalog, and Transducers. *J. Comput. Syst. Sci.*, 57(3):234–259, 1998. doi:10.1006/jcss.1998.1562.

8 Pierre Boullier. *Range Concatenation Grammars*, pages 269–289. New Developments in Parsing Technology. Springer, 2004. doi:10.1007/1-4020-2295-6_13.

9 Anne Brüggemann-Klein. Regular expressions into finite automata. *Theoretical Computer Science*, 120(2):197–213, 1993. doi:10.1016/0304-3975(93)90287-4.

10 Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. Complexity and Expressive Power of Logic Programming. *ACM Computing Surveys*, 33(3):374–425, 2001. doi:10.1145/502807.502810.

11 Heinz-Dieter Ebbinghaus and Jörg Flum. *Finite Model Theory*. Springer Monographs in Mathematics, 2nd edition, 1999.

12 Ronald Fagin, Benny Kimelfeld, Frederick Reiss, and Stijn Vansummeren. Document Spanners: A Formal Approach to Information Extraction. *J. ACM*, 62(2), 2015. doi:10.1145/2699442.

13 Fernando Florenzano, Cristian Riveros, Martín Ugarte, Stijn Vansummeren, and Domagoj Vrgoč. Efficient Enumeration Algorithms for Regular Document Spanners. *ACM Trans. Database Syst.*, 45(1), 2020. doi:10.1145/3351451.

14 Dominik D. Freydenberger. A Logic for Document Spanners. *Theory of Computing Systems*, 63(7):1679–1754, 2019. doi:10.1007/s00224-018-9874-1.

15 Dominik D. Freydenberger and Mario Holldack. Document Spanners: From Expressive Power to Decision Problems. *Theory Comput. Syst.*, 62(4):854–898, 2018. doi:10.1007/S00224-017-9770-0.

16 Dominik D. Freydenberger and Liat Peterfreund. The Theory of Concatenation over Finite Models. In *Proc. ICALP 2021*, pages 130:1–130:17, 2021. doi:10.4230/LIPIcs.ICALP.2021.130.

17 Dominik D. Freydenberger and Markus L. Schmid. Deterministic regular expressions with back-references. *J. Comput. Syst. Sci.*, 105:1–39, 2019. doi:10.1016/J.JCSS.2019.04.001.

18 Dominik D. Freydenberger and Sam M. Thompson. Splitting Spanner Atoms: A Tool for Acyclic Core Spanners. In *Proc. ICDT 2022*, pages 10:1–10:18, 2022. doi:10.4230/LIPIcs.ICDT.2022.10.

19 V M Glushkov. The Absract Theory of Automata. *Russian Mathematical Surveys*, 16(5), 1961. doi:10.1070/RM1961v016n05ABEH004112.

20 Georg Gottlob and Christos Papadimitriou. On the complexity of single-rule datalog queries. *Information and Computation*, 183(1):104–122, 2003. doi:10.1016/S0890-5401(03)00012-9.

21 B. Groz and S. Maneth. Efficient testing and matching of deterministic regular expressions. *J. Comp. Syst. Sci.*, 89:372–399, 2017. doi:10.1016/j.jcss.2017.05.013.

22 John E. Hopcroft, Rajeev Motwani, and Jeffrey D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 3rd edition, 2007.

23 Artur Jez. Word Equations in Nondeterministic Linear Space. In *Proc. ICALP 2017*, pages 95:1–95:13, 2017. doi:10.4230/LIPIcs.ICALP.2017.95.

24 Juhani Karhumäki, Filippo Mignosi, and Wojciech Plandowski. The expressibility of languages and relations by word equations. *J. ACM*, 47(3):483–505, 2000. doi:10.1145/337244.337255.

25 K. N. King. Alternating Multihead Finite Automata. *Theoretical Computer Science*, 61(2):149–174, 1988. doi:10.1016/0304-3975(88)90122-3.

26 S. C. Kleene. *Representation of Events in Nerve Nets and Finite Automata*, pages 3–42. Princeton University Press, 1956. doi:10.1515/9781400882618-002.

27 Antoni Koscielski and Leszek Pacholski. Complexity of Makanin's algorithm. *J. ACM*, 43(4):670–684, 1996. doi:10.1145/234533.234543.

28 Leonid Libkin. *Complexity of First-Order Logic*, pages 87–111. Elements of Finite Model Theory. Springer, 2004. doi:10.1007/978-3-662-07003-1_6.

29 Dean Light, Ahmad Aiashi, Mahmoud Diab, Daniel Nachmias, Stijn Vansummeren, and Benny Kimelfeld. SpannerLib: Embedding Declarative Information Extraction in an Imperative Workflow. *Proc. VLDB Endow.*, 17(12):4281–4284, 2024. doi:10.14778/3685800.3685855.

30 Carsten Lutz and Leif Sabellek. Ontology-Mediated Querying with the Description Logic EL: Trichotomy and Linear Datalog Rewritability. In *IJCAI 2017*, pages 1181–1187, 2017. doi:10.24963/ijcai.2017/164.

31 S. Miyano, A. Shinohara, and T. Shinohara. Which classes of Elementary Formal Systems are polynomial-time learnable? In *Proc. ALT 1992*, pages 139–150, 1992.

32 Yoav Nahshon, Liat Peterfreund, and Stijn Vansummeren. Incorporating information extraction in the relational database model. In *Proc. WebDB 2016*, 2016. doi:10.1145/2932194.2932200.

33 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. doi:10.4230/LIPIcs.ICDT.2019.13.

34 J. L. Ponty, D. Ziadi, and J. M. Champarnaud. A new quadratic algorithm to convert a regular expression into an automaton. In *Automata Implementation*, pages 109–119. Springer, 1997.

35 Markus L. Schmid and Nicole Schweikardt. Spanner Evaluation over SLP-Compressed Documents. In *Proc. PODS 2021*, pages 153–165, 2021. doi:10.1145/3452021.3458325.

36 Luc Segoufin. Enumerating with constant delay the answers to a query. In *Proc. ICDT 2013*, pages 10–20, 2013. doi:10.1145/2448496.2448498.

37 Sam M. Thompson and Dominik D. Freydenberger. Languages Generated by Conjunctive Query Fragments of FC[REG]. In *Proc. DLT 2023*, pages 233–245, 2023. doi:10.1007/978-3-031-33264-7_19.

38 Sam M. Thompson and Dominik D. Freydenberger. Generalized Core Spanner Inexpressibility via Ehrenfeucht-Fraïssé Games for FC. *Proc. ACM Manag. Data*, 2(2), 2024. doi:10.1145/3651143.

39 Yanshan Wang, Liwei Wang, Majid Rastegar-Mojarad, Sungrim Moon, Feichen Shen, Naveed Afzal, Sijia Liu, Yuqun Zeng, Saeed Mehrabi, Sunghwan Sohn, and Hongfang Liu. Clinical information extraction applications: A literature review. *Journal of Biomedical Informatics*, 77:34–49, 2018. doi:10.1016/j.jbi.2017.11.011.