Definable Relations and First-Order Query Languages over Strings*

Michael Benedikt† Bell Labs
Leonid Libkin§ U. Toronto
Thomas Schwentick¶ U. Marburg
Luc Segoufin‖ INRIA

Abstract

We study analogs of classical relational calculus in the context of strings. We start by studying string logics. Taking a classical model-theoretic approach, we fix a set of string operations and look at the resulting collection of definable relations. These form an algebra — a class of n-ary relations for every n, closed under projection and Boolean operations. We show that by choosing the string vocabulary carefully, we get string logics that have desirable properties: computable evaluation and normal forms. We identify five distinct models and study the differences in their model-theory and complexity of evaluation. We identify a subset of these models which have additional attractive properties, such as finite VC dimension and quantifier elimination.

Once you have a logic, the addition of free predicate symbols gives you a string query language. The resulting languages have attractive closure properties from a database point of view: while SQL does not allow the full composition of string pattern-matching expressions with relational operators, these logics yield compositional query languages that can capture common string-matching queries while remaining tractable. For each of the logics studied in the first part of the paper, we study properties of the corresponding query languages. We give bounds on the data complexity of queries, extend the normal form results from logics to queries, and show that the languages have corresponding algebras expressing safe queries.

1 Introduction

In the past 40 years, various connections between logic on strings, formal languages and finite automata have been explored in great detail. The standard setting for connecting logical definability with various properties of formal languages is to represent strings over a finite alphabet $\Sigma = \{a_1, \ldots, a_n\}$ as first-order structures in the signature $(P_{a_1}, \ldots, P_{a_n}, <)$, so that the structure M_s for a string s of length k has the universe $\{1, \ldots, k\}$, with $<$ being the usual ordering, and P_{a_i} being the set of the positions l such that the lth character in s is a_i. Then a sentence Φ of some logic $\mathcal{L}$ defines a language $L(\Phi) = \{s \in \Sigma^* \mid M_s \models \Phi\}$. Two classical results on logic and language theory state that languages thus definable in monadic second-order logic (MSO) are precisely the regular

*Part of this work was done while the second and the third authors visited INRIA, and the second and the fourth authors visited Mainz.

†Bell Laboratories, 263 Shuman Blvd, Naperville, IL 60566, USA. E-mail: benedikt@research.bell-labs.com.

§Department of Computer Science, University of Toronto, 6 King's College Road, Toronto, Ontario M5S 3H5, Canada. E-mail: libkin@cs.toronto.edu.

¶Fachbereich Mathematik und Informatik, Philipps-Universität Marburg, D-35032 Marburg, Germany. Email: tick@informatik.uni-marburg.de. Part of this work done while at U. Mainz.

‖INRIA-Rocquencourt, B.P. 105, 78153 Le Chesnay Cedex, France. E-mail: Luc.Segoufin@inria.fr.

languages [20], and the languages definable in first-order logic (FO) are precisely the star-free languages [54]. For a survey, see [65, 67].

An alternative approach to definability of strings, based on classical infinite model theory rather than finite model theory, dates back to the 1960s [20, 19]. One considers an infinite structure M consisting of $\langle \Sigma^*, \Omega \rangle$, where Ω is a set of functions, predicates and constants on Σ^*. One can then look at definable sets, those of the form $\{\vec{a} \mid M \models \varphi(\vec{a})\}$, where φ is a first-order formula in the language of M. A well-known result links definability with traditional formal language theory. Let Ω_{reg} consist of unary functions l_a, $a \in \Sigma$, binary predicates $\text{el}(x, y)$ and $x \preceq y$, where $l_a(x) = x \cdot a$, $\text{el}(x, y)$ states that x and y have the same length, and $x \preceq y$ states that x is a prefix of y. Let $\mathbf{S}_{\text{len}}$ be the model $\langle \Sigma^*, \Omega_{\text{reg}} \rangle$ (we will explain the notation later). Then subsets of Σ^* definable in $\mathbf{S}_{\text{len}}$ are precisely the regular languages [20, 19, 14]; moreover, this implies decidability of the first-order theory of $\mathbf{S}_{\text{len}}$ [45, 14].

The key advantage of the "model-theoretic approach" is that one immediately gets an extension of the notion of recognizability from string languages to n-ary string relations for arbitrary n. One gets an algebra of n-ary string relations for every n, and these algebras automatically have closure under projection and product, in addition to the Boolean operations. In the case of the model $\mathbf{S}_{\text{len}}$ above, this algebra is not new: in fact, the definable n-ary relations are exactly the ones recognizable under a natural notion of automaton running over n-tuples [19, 29]. We will refer to these automata-definable relations as the *regular relations*: the formal definition is given in subsection 3.1.1. We show here that by taking restrictions of the model $\mathbf{S}_{\text{len}}$, one gets new algebras of regular relations which behave better, in many ways, than the full algebra of recognizable relations given by $\mathbf{S}_{\text{len}}$. We introduce four such models here, and show that the definable sets in these models enjoy superior model-theoretic properties relative to the full algebra of recognizable relations associated with $\mathbf{S}_{\text{len}}$.

A key motivation for finding closed algebras of string relations comes from the field of databases, in particular, the study of query languages with interpreted operations [8, 10, 37, 50]. String manipulation facilities have long been recognized as a critical component of a realistic database query language. In SQL, for example, the `WHERE` clause can contain string *pattern-matching expressions*, such as `FACULTY.NAME LIKE 'Nyk%nen'`. These expressions can themselves be seen as queries over string relations: the above clause, for example, can be seen as a selection performed on a projection of the `FACULTY` relation. While the Relational Calculus gives a satisfactory formal model for SQL queries in the absence of built-in datatypes, there has been thus far no satisfactory model that fully accounts for string queries. The lack of an adequate formal model is related to the fact that SQL restricts the interaction of string operations and relational operations in a number of ad-hoc ways: one cannot apply the `LIKE` operator to a subquery to build up a new query, nor can one take the product of two string expressions built with `LIKE`. The natural way to obtain a calculus on string relations where one can freely compose string operations and relational operators is to start with a decidable structure on strings, like those mentioned above, and extend them to query languages by adding free predicate symbols — in the same way that traditional Relational Calculus can be obtained from first-order logic over pure equality. Using this approach we see that corresponding to $\mathbf{S}_{\text{len}}$ and each of the four restricted models mentioned above, we obtain five interesting compositional query languages on strings.

The paper has two main parts. In the first part, we study definable algebras of string relations, that is, model-theoretic structures on Σ^* and definability in these structures. We focus on five structures, of which the model $\mathbf{S}_{\text{len}}$ mentioned above is the richest. In the second part of the paper, we deal with database applications, and study the corresponding query languages for string

databases given by each of the five structures. This can be thought of as definability over model-theoretic structures *and* a finite relational database. Naturally, the results of the first part form the basis for reasoning about string query languages.

We now summarize the developments in both parts of the paper.

As mentioned above, we know that there exists *a* regular string algebra [20, 19, 14], i.e., an algebra which exactly captures the regular sets when restricted to unary relations. An obvious question to ask, then, is whether *new* algebras of string relations arise through the model-theoretic approach. In particular, if we restrict the signature Ω to be less expressive than Ω_{reg}, do we get new relation algebras lying within the recognizable relations?

A natural starting point would be to find a signature that captures properties of the star-free sets. Here again, a simple example leaps out: consider the signature $\Omega_{\text{sf}} = (\preceq, (l_a)_{a\in\Sigma})$, and let $\mathbf{S} = \langle \Sigma^*, \Omega_{\text{sf}} \rangle$. One can easily show that the definable subsets of Σ^* in $\mathbf{S}$ are exactly the star-free ones. Furthermore, we will show that the definable n-ary relations of this model are exactly those definable by regular prefix automata (cf. [4]) whose underlying string automata are counter-free.

Just as there is a significant difference between the complexity-theoretic behavior of regular languages and star-free languages (the latter are in AC^0 whereas the former are not), we find that the model $\mathbf{S}$ is much more tractable, in terms of its model-theory and its complexity than $\mathbf{S}_{\text{len}}$. In particular, we show that $\mathbf{S}$ has quantifier-elimination in a natural relational extension, while $\mathbf{S}_{\text{len}}$ does not.

It would be tempting to think of $\mathbf{S}$ and $\mathbf{S}_{\text{len}}$ as canonical extensions of the notions of regularity and star-free to n-ary relations. However, we will show that in fact there are *many* choices for Ω that share the same one-dimensional definable sets (either star-free or regular). Furthermore, algebras of definable sets may be identical in terms of the string languages they define, but differ considerably in the n-ary string relations in the definable algebra. We thus say that an algebra of definable sets based on $\langle \Sigma^*, \Omega \rangle$, with $\Omega \subseteq \Omega_{\text{reg}}$ is a *regular algebra of definable sets* if the subsets of Σ^* in it (i.e the one-dimensional definable sets of $\langle \Sigma^*, \Omega \rangle$) are exactly the regular sets. We likewise say that the algebra based on definable sets for $\langle \Sigma^*, \Omega \rangle$ is a *star-free algebra of definable sets* if the subsets of Σ^* in the algebra are exactly the star-free sets.

We then study new examples of regular and star-free definable algebras. We give an example of a star-free algebra with considerably more expressive power than the basic star-free algebra $\mathbf{S}$. This model, which we denote by $\mathbf{S}_{\text{left}}$ (as it allows one to add characters on the *left* of a string), shares most of the desirable properties of $\mathbf{S}$: in particular, it has quantifier-elimination in a natural language, and membership test in this algebra has low complexity.

More surprisingly, perhaps, we give examples of regular algebras (which we denote $\mathbf{S}_{\text{reg}}$ and $\mathbf{S}_{\text{reg,left}}$) that are strictly contained in $\mathbf{S}_{\text{len}} = \langle \Sigma^*, \Omega_{\text{reg}} \rangle$. Although the one-dimensional sets in these algebras are still the regular sets, the algebra as a whole shares many of the attractive properties of the star-free languages. In particular, we give quantifier-elimination results for these algebras. In contrast to this, we present a result giving a partial answer to open question 0 in [55], which asks whether $\mathbf{S}_{\text{len}}$ itself has quantifier-elimination in a reasonable signature. We show that it does not have quantifier-elimination in any relational signature of bounded arity but does have quantifier-elimination in a signature containing binary functions.

We now turn to the second part of the paper, studying the string query languages formed from

each of these models. What are some properties one would desire of a string query language? One problem faced in any work combining string pattern-matching queries with relational calculus is that pattern-matching expressions may return an infinite number of strings. This is the standard issue of *safety*. Previous proposals for combining relational algebra with string-matching primitives tackle this problem by identifying safe fragments of their languages, using a number of syntactic restrictions — see, e.g., [39, 42, 38, 40, 59] — but they cannot capture the safe fragment of the language syntactically. A second issue with any string query language is its expressive power. Many query languages designed in the prior literature turn out to be Turing complete, a feature that in turn makes many sorts of analysis and optimization impossible. Indeed, as noted in [40], adding just concatenation to the relational calculus already yields a query language which is Turing complete. This immediately implies that there is no effective syntax for the corresponding safe fragment [64].

In contrast to the above, we would like our languages to fulfill the following criteria:

1. Query evaluation is efficient;
2. There is effective syntax capturing safe queries;
3. There is an algebra equivalent to the language.

Hence, we consider each of our query languages with respect to these criteria. As mentioned above, we consider relational calculus, RC, over each model defined in the first part, beginning with the weakest model, $\mathbf{S}$. The query language obtained by adding database relations to $\mathbf{S}$ captures basic SQL with simple `LIKE` pattern-matching and lexicographic ordering. We show that the safe fragment of this model can be effectively captured in a natural way, and prove complexity bounds for queries in this language that match the known bounds for ordinary relational calculus. RC($\mathbf{S}$) however, is unable to express certain natural queries, e.g., `SELECT` $a \cdot x$ `FROM` R, where a is a fixed character. We contrast this to the query language RC($\mathbf{S}_{\text{len}}$) formed over the richest model. This extension has much greater expressiveness: it enables additional operations such as trimming/adding symbols on both left and right of a string, and the `SIMILAR` pattern-matching for checking membership in a regular language [41]. We show that this language also satisfies criteria 2 and 3 above, but in RC($\mathbf{S}_{\text{len}}$) one can express NP-complete and coNP-complete problems.

This leads us to the consideration of the three intermediate languages, RC($\mathbf{S}_{\text{left}}$) ,RC($\mathbf{S}_{\text{reg}}$), and RC($\mathbf{S}_{\text{reg,left}}$). We find that each of these languages satisfies all three of the required criteria, while considerably extending the expressive power of RC($\mathbf{S}$).

Related Work: One motivation of our approach was the study of *automatic structures* [48, 14], which are a subclass of recursive structures [43], and were introduced as a generalization of automatic groups [30]. In an automatic structure $M = \langle \Sigma^*, \Omega \rangle$, every predicate in Ω is definable by a finite automaton. More precisely, an n-ary predicate P is given by a letter-to-letter n-automaton [29, 34]. These structures were also studied in [45] in connection with decidability questions for first-order theories.

It is known [19, 14] that a structure is automatic iff it can be interpreted in the structure $\mathbf{S}_{\text{len}}$; hence $\mathbf{S}_{\text{len}}$ is in some sense the universal automatic structure. The first part of this paper can be seen as a study of subclasses of automatic structures definable within $\mathbf{S}_{\text{len}}$ that are significantly more restrictive, and that might have stronger model-theoretic or computational properties than a rich structure like $\mathbf{S}_{\text{len}}$.

The structure $\mathbf{S}_{\text{left}}$, without the prefix relation, is useful for modeling queues and it first appeared in the verification context [16], where an algorithm for deciding existential sentences was given. That algorithm was extended to the full theory in [60], but still without the prefix relation.

On the database side, several approaches toward unifying string algebras with relational algebra have been developed in the prior literature. Most of them are based on the concatenation operator, or other operations that make logics undecidable in general. [36] studied the consequences of adding pattern-matching features to SQL. Papers [39, 42, 38] proposed an extension of the relational calculus with alignment logics and studied their complexity and expressive power. Without restrictions, they can define an arbitrary r.e. set [39]. Another approach was proposed in [17, 18], which considered Datalog extended with appropriate transducers for string operations, and proved a number of completeness results. In [24] arbitrary regions (substrings) can be queried; this, when coupled with relational calculus, gives the power of string concatenation. Closer to our approach, [40, 59] study the relational calculus/algebra extended with an operation for concatenating strings. [25] studies first-order logic over term algebras and extends expressive bounds and complexity results from relational calculus to this setting. But SQL-style string pattern-matching cannot be expressed in the language of [25] – indeed in this language one cannot even query for strings beginning with a fixed symbol.

The general approach to studying databases over interpreted domains is closely related to the field of constraint databases [50]. Most theory of constraint databases was done over continuous domains, typically various structures over the reals. In contrast, our results could be viewed as the theory of (finite) constraint databases over discrete domains, in particular, strings.

Organization: The paper is organized as follows. The next section gives the notation that will be used in the paper. Then we deal with definability for models on strings, in particular, quantifier elimination, bounded VC dimension and expressive power. The last part contains database applications in terms of expressiveness, data complexity and safety of the corresponding query languages. Earlier presentation of this work appeared in two conference proceedings: [13, 12].

2 Notation

Throughout the paper, Σ denotes a finite alphabet, and Σ^* the set of all finite strings over Σ. We consider a number of operations and predicates on Σ^*:

- $x \cdot y$ – concatenation of two strings x and y.
- $x \preceq y$ – x is a prefix of y.
- $l_a(x)$, $a \in \Sigma$, is $x \cdot a$ (adds *l*ast character).
- $f_a(x)$, $a \in \Sigma$, is $a \cdot x$ (adds *f*irst character).
- $|x|$ is the length of string x.
- $x \sqcap y$ is the longest common prefix of the strings x and y.
- $x - y$ – the string z such that $y \cdot z = x$, if it exists, and ϵ otherwise.

- $x{+}y$, which is an alternative notation for the concatenation $y{\cdot}x$. Note that always $(x{+}y){-}y = y$.
- $\mathrm{el}(x, y)$ is true iff $|x| = |y|$.

We write $w[i, j]$ to refer to the substring of a string w starting from position i and ending at position j. Here, the first position of a string has number 1, e.g., it holds that $w = w[1, |\, w\,|]$. We write $w[i]$ for $w[i, i]$.

We write $x \lessdot y$ to express that y extends x by exactly one symbol. Let $\mathit{prefix}(C)$ stand for the *prefix-closure* of C: $\{s \mid s \preceq s', s' \in C\}$. By $\downarrow(C)$ we denote $\{s \mid |s| \leq |s'|, s' \in C\}$.

Given a set S of strings , we let $\mathrm{Tree}(S)$ be the tree (i.e. the partially-ordered structure) generated by closing $S \cup \{\epsilon\}$ under $\sqcap$. In other words, $\mathrm{Tree}(S)$ is the poset $\langle \{x \sqcap y \mid x, y \in S \cup \{\epsilon\}\}, \prec\rangle$. (Note that for any set of strings $s_1, \ldots, s_k$, there are two indices $i, j \leq k$ such that $s_1 \sqcap \ldots \sqcap s_k = s_i \sqcap s_j$.)

If S is a set of strings and $w \in \Sigma^*$, let $\mathrm{Meet}(w, S)$ be the longest string among $\{w \sqcap u \mid u \in S\}$, let $\mathrm{Meet}_-(w, S)$ be the element of $\mathrm{Tree}(S)$ which is the longest prefix of $\mathrm{Meet}(w, S)$, and let $\mathrm{Meet}_+(w, S)$ be the smallest element of $\mathrm{Tree}(S)$ for which $\mathrm{Meet}(w, S)$ is a prefix. Note that $\mathrm{Meet}_+(w, S)$ is well-defined (as are $\mathrm{Meet}(w, S)$ and $\mathrm{Meet}_-(w, S)$), since $\mathrm{Meet}(w, S)$ is either a string from $\mathrm{Tree}(S)$ or it has a unique smallest extension in $\mathrm{Tree}(S)$.

A *complete tree-order description* of a vector $\vec{w}$ of variables is the atomic diagram of $\mathrm{Tree}(\vec{w})$ in the language of $\epsilon, \preceq, \sqcap$. In other words, it is a specification of all the $\preceq$ relations that hold and do not hold in $\mathrm{Tree}(\vec{w})$.

For example, let $\vec{w} = (a, aba, abbb)$. Then $aba \sqcap abbb = ab$, and $\mathrm{Tree}(\vec{w})$ is $\{\epsilon, a, ab, aba, abbb\}$. The complete tree-order description of $\vec{w}$ consists of all the $\preceq$ relations that hold among the elements of $\{\epsilon, a, ab, aba, abbb\}$, as well as all the $\sqcap$-relations, e.g., $aba \sqcap abbb = ab$, $a \sqcap aba = a$, $ab \sqcap \epsilon = \epsilon$, etc.

We shall consider several structures on Σ^*. The basic one is the structure $\mathbf{S} = \langle \Sigma^*, \preceq, (l_a)_{a\in\Sigma}\rangle$. We could equivalently use unary predicates L_a, where $L_a(x)$ is true for strings x having a as last symbol. Note that in the presence of $\preceq$, l_a and L_a are interdefinable, and we thus shall use both of them.

We further consider a number of extensions of $\mathbf{S}$. In one of them characters can be added on the *left* as well as on the right. This structure is denoted by $\mathbf{S}_{\mathrm{left}} \stackrel{\mathrm{def}}{=} \langle \Sigma^*, \preceq, (l_a)_{a\in\Sigma}, (f_a)_{a\in\Sigma}\rangle$.

Another extension, denoted by $\mathbf{S}_{\mathrm{len}}$, adds *length* comparisons via the el predicate (note that using $\preceq$ and el one can express various relationships between lengths of strings, e.g. $|x|\{=, \neq, <, >\}|y|$, $|x| = |y| + k$ for a constant k, etc.). To summarize, we mainly deal with the following structures:

- $\mathbf{S} = \langle \Sigma^*, \preceq, (l_a)_{a\in\Sigma}\rangle$;
- $\mathbf{S}_{\mathrm{left}} = \langle \Sigma^*, \preceq, (l_a)_{a\in\Sigma}, (f_a)_{a\in\Sigma}\rangle$;
- $\mathbf{S}_{\mathrm{len}} = \langle \Sigma^*, \preceq, (l_a)_{a\in\Sigma}, \mathrm{el}\rangle$.

Once we consider regular algebras, we introduce two more structures; however, operations in them will be motivated by quantifier-elimination results for $\mathbf{S}$ and $\mathbf{S}_{\mathrm{left}}$ and thus those structures will be defined later.

There is a very close connection between $\mathbf{S}_{\text{len}}$ and an extension of Presburger arithmetic. Assume that $\Sigma = \{0, 1\}$. Let $\text{val}(n)$, for $n \in \mathbb{N}$, be n in binary, considered as a string in Σ^*. Let $V_2(n)$ be the largest power of 2 that divides n. Then $P \subseteq \mathbb{N}^k$ is definable in $\langle \mathbb{N}, +, V_2 \rangle$ iff $\{(\text{val}(n_1), \ldots, \text{val}(n_k)) \mid (n_1, \ldots, n_k) \in P\}$ is definable in $\mathbf{S}_{\text{len}}$ [20, 19].

Definability over $\mathbf{S}, \mathbf{S}_{\text{left}}, \mathbf{S}_{\text{len}}$. We give a few simple examples of definability over these structures.

Matching with `LIKE` can be expressed over $\mathbf{S}$, since definable subsets in $\mathbf{S}$ are precisely star-free languages. For example, the condition x `LIKE a_b%a_` — saying that the first symbol of x is a, the third is b, and the last but one is a again — can be expressed by a formula $\varphi(x)$:

$$\exists u, v, w \left(\begin{array}{ll} & u \prec v \prec w \prec x \\ \wedge & L_a(u) \wedge L_b(v) \wedge L_a(w) \\ \wedge & \psi_1(u) \wedge \psi_3(v) \wedge \psi_{-1}(w) \end{array} \right),$$

where $\psi_1(u), \psi_3(v), \psi_{-1}(w)$ say that u, v, w are prefixes extending up to the first, third, and penultimate positions in the string x.

Another important operation expressible over $\mathbf{S}$ is the *lexicographic* ordering $\leq_{\text{lex}}$. Assume that $\Sigma = \{a_1, \ldots, a_n\}$ and an ordering $a_1 < \ldots < a_n$ is given. The lexicographic ordering $x \leq_{\text{lex}} y$ is then expressed by:

$$x \preceq y \vee \exists z \ (z \prec x \wedge z \prec y \wedge \bigvee_{i<j} ((l_{a_i}(z) \preceq x) \wedge (l_{a_j}(z) \preceq y))) \ .$$

The graph of the function f_a, $\{(x, y) \mid y = f_a(x)\}$, is definable in $\mathbf{S}_{\text{len}}$ by

$$\begin{array}{rl} & |y| = |x| + 1 \wedge (\exists w \prec y \ |w| = 1 \wedge L_a(w)) \\ \wedge & \forall z \prec x \exists v \prec y \ (|v| = |z| + 1 \wedge \bigwedge_{b \in \Sigma} L_b(z) \leftrightarrow L_b(v)), \end{array}$$

where $|v| = |u| + 1$ is defined by $\exists w (w \lessdot u \wedge \text{el}(w, v))$, and $w \lessdot u \equiv w \prec u \wedge \neg \exists t \ (w \prec t \wedge t \prec u)$.

Strings as structures We shall use classical results on definability of strings represented as finite first-order structures. If $\Sigma = \{a_1, \ldots, a_n\}$, then a string $s \in \Sigma^*$ can be represented as a structure M_s in the signature $(P_{a_1}, \ldots, P_{a_n}, <)$. If $|s| = k$, then the universe of M_s is $\{1, \ldots, k\}$, $<$ is interpreted as the usual ordering, and P_{a_j} is the set $\{i \mid 1 \leq i \leq k, \text{and the } i\text{th position of } s \text{ is } a_j\}$.

If Φ is a sentence of some logic, it defines a language $L(\Phi) = \{s \in \Sigma^* \mid M_s \models \Phi\}$. When the logic is MSO, monadic second-order logic, the languages that arise this way are precisely the regular languages [20]. When the logic is FO, first-order, then the languages that arise are precisely the star-free languages (that is, those that can be obtained from $\emptyset$ and $\{a_i\}, i \leq n$ by using the operations of union, complement, and concatenation) [54].

Databases and query languages A database schema SC is a collection of relation names $R_1, \ldots, R_l$, R_i being of arity $p_i > 0$. In an instance of SC over a set U, each R_i is interpreted as a finite subset of U^{p_i}. The *active domain* of a database D, $adom(D)$, is the set of elements from U that appear in D.

The general setting for query languages is that of a finite database and an infinite underlying structure $\mathcal{M} = \langle U, \Omega \rangle$, where Ω is a set of operations (functions and predicates) on U. As our basic language we consider relational calculus, or first-order logic, over the schema SC and $\mathcal{M}$, denoted by $\mathrm{RC}(SC, \mathcal{M})$. We often omit SC when it is understood, or irrelevant. Here we will focus exclusively on the string datatype, hence we will always have $U = \Sigma^*$. For example, if $\mathcal{M} = \langle \Sigma^*, \prec, (L_a)_{a \in \Sigma} \rangle$, the query

$$\exists x \; R(x) \wedge L_0(x) \wedge \exists y (y \prec x \wedge L_1(y) \wedge (\neg \exists z \; y \prec z \prec x))$$

tests if there is a string in the relation R which ends with 10. Indeed, it asks if the last symbol of x is 0, and if there exists a prefix y, which is the largest proper prefix of x (as there is no z with $y \prec z \prec x$) such that the last symbol of y is 1.

Given a query $\varphi(x_1, \ldots, x_n)$ in $\mathrm{RC}(SC, \mathcal{M})$ and $\vec{a} \in U^n$, we write $D \models \varphi(\vec{a})$ when $\varphi(\vec{a})$ is true in $(D, \mathcal{M})$. We write $\varphi(D)$ for the output of φ on D, that is, $\{\vec{a} \in U^n \mid D \models \varphi(\vec{a})\}$. We say that φ is *safe on* D if $\varphi(D)$ is finite, and that φ is *safe* if it is safe on every D. The safety problem is to determine whether a query is safe, and it is known to be undecidable even for the pure relational calculus [1]. The *state-safety* problem is to decide, for a given φ and D, if φ is safe on D.

We say that safe queries in $\mathrm{RC}(\mathcal{M})$ *have effective syntax* if there exists a recursively enumerable set A, of safe queries in $\mathrm{RC}(\mathcal{M})$ such that, for every SC, every safe $\mathrm{RC}(SC, \mathcal{M})$ query is equivalent to one in A.

Effective syntax is a first step towards an algebraic language expressing all safe queries. Indeed if such a language exists, safe queries must have effective syntax.

That effective syntax exists for safe queries in the pure relational calculus is a classical relational theory result [1]. Other results – both positive or negative – have been proved recently [11, 64].

Collapse results These establish very strong expressivity bounds for relational calculi. To formulate them, we need an important restriction of queries: to quantification over the active domain. We use quantifiers $\exists x \in adom$ and $\forall x \in adom$, whose meaning is as follows: $D \models \exists x \in adom \; \varphi(x, \cdot)$ if $D \models \varphi(a, \cdot)$ for some $a \in adom(D)$ (as opposed to for some $a \in U$ in the case of the usual $\exists x$ quantifier), and similarly for the universal quantifier. These restricted quantifiers are definable in relational calculus, but it is often helpful to have them available separately.

A relational calculus formula is called an *active-domain* formula if all quantifiers in it are of the form $\forall x \in adom, \exists x \in adom$. We say that $\mathrm{RC}(\mathcal{M})$ admits *natural-active collapse* [10] if every $\mathrm{RC}(\mathcal{M})$ formula is equivalent to an active-domain formula. We say that $\mathrm{RC}(\mathcal{M})$ admits *restricted quantifier collapse* if every $\mathrm{RC}(\mathcal{M})$ formula is equivalent to one in which SC-relations appear only under the scope of quantifiers $\exists x \in adom$ and $\forall x \in adom$. Note that if $\mathcal{M}$ admits quantifier-elimination, these two notions coincide.

A query is *generic* if it commutes with permutations on the domain. The active-generic collapse [10] states that if an $\mathrm{RC}(\mathcal{M})$ formula with quantification of the form $\exists x \in adom$ and $\forall x \in adom$ expresses a generic query Q, then Q must be expressible using only a linear order on the active domain, and no other predicates and functions from $\mathcal{M}$.

Model theory background Let Ω be a finite or countably infinite first-order signature, and M a model over Ω. By $\mathrm{FO}(M)$ we denote the set of all first-order formulae in the language of Ω. The

(complete) *theory* of M, $\mathrm{Th}(M)$, is the set of all sentences in $\mathrm{FO}(M)$ true in M. Two models M and M' over Ω are *elementary equivalent* if $\mathrm{Th}(M) = \mathrm{Th}(M')$.

We say that M admits *quantifier elimination (QE)* if for every formula $\varphi(\vec{x})$ in $\mathrm{FO}(M)$ there is a quantifier-free formula $\varphi'(\vec{x})$ such that $\forall \vec{x}\ \varphi(\vec{x}) \leftrightarrow \varphi'(\vec{x})$ is true in M. In every case where we show quantifier-elimination for a model in this paper, the conversion to a quantifier-free formula can be made effective, although in several cases (e.g. Theorem 3.12) we will not give the details of the effective versions.

For a tuple $\vec{a}$ and a model M over Ω, we let $tp_M(\vec{a})$ be the *type* of $\vec{a}$ in M (the set of all formulae of $\mathrm{FO}(M)$ satisfied by $\vec{a}$), and $atp_M(\vec{a})$ be the atomic type in M (the set of all quantifier-free formulae of $\mathrm{FO}(M)$ satisfied by $\vec{a}$). If A is a subset of M, $tp_M(\vec{a}/A)$ is the type of $\vec{a}$ over A in M (the set of all FO-formulae over $\Omega \cup A$ satisfied by $\vec{a}$).

An *ω-saturated model* M over Ω is a model such that each consistent type (a type is consistent if it has a witness in at least one model of Ω) over a finite set A in $\mathrm{FO}(M)$ is satisfied in M. It is known [21] that every model M over Ω has an elementary equivalent ω-saturated model M^*.

Many proofs use Ehrenfeucht-Fraïssé games [28, 33, 27]. For two structures $\mathcal{M}_1$ and $\mathcal{M}_2$ of the same vocabulary, we write $\mathcal{M}_1 \equiv_k \mathcal{M}_2$ if the duplicator has a winning strategy in the k-round game on $\mathcal{M}_1$ and $\mathcal{M}_2$ (that is, if $\mathcal{M}_1$ and $\mathcal{M}_2$ agree on all sentences of quantifier rank up to k). We also assume familiarity with Monadic Second Order Logic (MSO) [27]. Some proofs will use MSO games [27]; we write $\mathcal{M}_1 \equiv_{\mathrm{MSO}_k} \mathcal{M}_2$ if the duplicator has a winning strategy in the k-round MSO game, which similarly means the two structures can not be distinguished by MSO-sentences of quantifier depth k.

Isolation, VC-dimension, and collapse We review several model-theoretic concepts that prove useful in establishing bounds on the expressive power of query languages.

Let T be a theory over Ω and M be a model of T. A subset A of M is said to be *pseudo-finite* if $(M, A) \models F(T, P)$, where P is a unary predicate, and $F(T, P)$ is the set of all formulae of $\mathrm{FO}(\Omega \cup \{P\})$ satisfied by all finite sets of elements in any model of T.

If p is a type over A in M, a subset q of p *isolates* p if p is the only type over A in M containing q. A complete theory T over Ω is said to have the *strong isolation property* if for any model M of T and any pseudo-finite set A and any element a in M, there is a finite subset A_0 of A such that $tp_M(a/A_0)$ isolates $tp_M(a/A)$. We say that it has the *isolation property* if a countable A_0 exists as above.

Isolation is an interesting property in the database context because it implies the restricted quantifier collapse [8, 32]. Here we also use it to provide bounds on the VC-dimension of definable families.

For a family $\mathcal{C}$ of subsets of a set U, and a set $F \subseteq U$, we say that $\mathcal{C}$ *shatters* F if $\{F \cap C \mid C \in \mathcal{C}\}$ is the powerset of F. The *VC-dimension* of $\mathcal{C}$ is the maximum cardinality of a finite set shattered by $\mathcal{C}$ (or ∞, if arbitrarily large finite sets are shattered by $\mathcal{C}$). This concept is fundamental to learning theory, as finite VC-dimension of a hypothesis space is equivalent to learnability (PAC-learnability) [5, 15].

Now consider a structure $M = \langle \Sigma^*, \Omega \rangle$, and a $\mathrm{FO}(M)$ formula $\varphi(\vec{x}, \vec{y})$. For each $\vec{a}$, let $\varphi(\vec{a}, M) =$

$\{\vec{b} \mid M \models \varphi(\vec{a}, \vec{b})\}$. The family of sets $\varphi(\vec{a}, M)$, where $\vec{a}$ ranges over all tuples over M, is called a *definable family*. We say that M *has finite VC-dimension* if every definable family has finite VC-dimension. In particular, this implies learnability of FO-definable families over M.

We shall see more connections between isolation, VC dimension, and collapse results later in the paper.

Complexity classes Some complexity results in this paper refer to parallel complexity classes AC^0, TC^0, and NC^1. AC^0 is constant parallel time; more precisely, the class of languages accepted by polynomial-size constant-depth unbounded fan-in circuits. TC^0 additionally has majority gates of unbounded fan-in. In NC^1, there are no majority gates, the depth is allowed to be logarithmic, but fan-in is bounded. It is known that $\mathrm{AC}^0 \subset \mathrm{TC}^0 \subseteq \mathrm{NC}^1$ (parity separates TC^0 from AC^0). We consider uniform versions of these classes [7]; uniform AC^0 over finite structures can be characterized via definability in FO(BIT, <): first-order logic with linear order and the BIT predicate ($\mathrm{BIT}(i, j)$ is true iff the jth bit in the binary representation of i is one.) To capture uniform TC^0 it suffices to add counting quantifiers to FO(BIT, <) [7].

PH is the polynomial hierarchy, which contains, e.g., NP and coNP and is itself included in PSPACE [57].

As usual, for *data complexity*, one fixes a query Q and considers the complexity of $\{enc(D)\#enc(t) \mid t \in Q(D)\}$, where enc is an encoding of databases and tuples over some fixed alphabet, typically $\{0, 1\}$ [1]. Normally in pure relational calculus the encoding is such that the active domain is considered to be $\{1, \ldots, k\}$, and each number i is represented in binary. When we deal with *interpreted* elements stored in a database, such an encoding is not appropriate, as one needs to take into account operations on those interpreted elements. In particular, in the case of strings over a finite alphabet, we consider the encoding of a string to be itself (in the case of an alphabet different from $\{0, 1\}$ we may have to code letters in $\{0, 1\}$ first).

3 Model theory of strings

In this section we study logical definability over $\mathbf{S}_{\mathrm{len}}, \mathbf{S}, \mathbf{S}_{\mathrm{left}}$ and two other structures, defining regular algebras over Σ^*. We are particularly interested in quantifier-elimination results, and in some model-theoretic properties (isolation, VC dimension) that will later give us results about the expressive power of the relational calculi based on these structures. We start with the strongest regular algebra $\mathbf{S}_{\mathrm{len}}$, then move to the star-free algebra $\mathbf{S}$, and to a more expressive star-free algebra $\mathbf{S}_{\mathrm{left}}$. The quantifier-elimination proof for the latter is technically the most involved result in this section. We then show how to expand $\mathbf{S}$ and $\mathbf{S}_{\mathrm{left}}$ to regular algebras, without losing their nice properties.

3.1 A regular algebra based on $\mathbf{S}_{\mathrm{len}}$

In this subsection we will focus on the structure $\mathbf{S}_{\mathrm{len}}$. We will assume here that the alphabet Σ contains at least two letters. For a 1-letter alphabet , it is easy to see that $\mathbf{S}_{\mathrm{len}}$ reduces to $\mathbf{S}$, which will be dealt with in the next subsection.

3.1.1 Automata and Definability

A *letter-by-letter* automaton is a usual DFA whose alphabet is $(\Sigma \cup \{\#\})^n$, $\# \notin \Sigma$. An n-tuple of strings $s_1, \ldots, s_n$ can be viewed as a word of length $\max_i |s_i|$ over the alphabet $\Sigma \cup \{\#\}$, where the jth letter is the tuple $(s_1^j, \ldots, s_n^j)$; here s_k^j is the jth letter of s_k, if $|s_k| \leq j$, and $\#$ otherwise. We say that a predicate $P \subseteq (\Sigma^*)^n$ is definable by a letter-to-letter n-automaton A if $(s_1, \ldots, s_n) \in P$ iff A accepts $s_1, \ldots, s_n$.

As mentioned in the introduction, $\mathbf{S}_{\text{len}} = \langle \Sigma^*, \preceq, (l_a)_{a \in \Sigma}, \text{el} \rangle$ is the canonical automatic structure, and relations definable in $\mathbf{S}_{\text{len}}$ are precisely the *regular relations*, that is, k-ary definable relations are precisely those given by letter-to-letter k-automata [14, 19]. In particular, this gives a normal form for $\mathbf{S}_{\text{len}}$-formulae. We introduce a new type of *length-bounded quantifiers* of the form $\exists |x| \leq |y|$ and $\forall |x| \leq |y|$. A formula $\exists |x| \leq |y| \varphi$ is meant as an abbreviation for $\exists x((|x| \leq |y|) \wedge \varphi)$.

Since every finite automaton can be simulated by a length-bounded FO($\mathbf{S}_{\text{len}}$) formula, we conclude that each FO($\mathbf{S}_{\text{len}}$) formula is equivalent to a length-bounded FO($\mathbf{S}_{\text{len}}$) formula. Note that this result can also be shown directly by an Ehrenfeucht-Fraïssé game argument.

3.1.2 Quantifier Elimination

The universal property of $\mathbf{S}_{\text{len}}$ mentioned above indicates that $\mathbf{S}_{\text{len}}$ may be "too rich" in relations for many applications. We present evidence for this by addressing the open question of [22, 55] whether $\mathbf{S}_{\text{len}}$ has quantifier elimination in a reasonable signature. One first needs to define what "reasonable" means here. Clearly, every structure has quantifier elimination in a sufficiently large expansion of the signature: add symbols for all definable predicates, for example. One can thus take reasonable to mean a finite expansion, but this is not satisfactory: for example, Presburger arithmetic has quantifier elimination in an infinite signature $(+, <, 0, 1, (\text{mod } k)_{k>1})$ [31]. Note however that in this example, the maximum arity of the predicates and functions is 2. In fact, it appears to be a common phenomenon that when one proves quantifier elimination in an infinite signature, there is an upper bound on the arity of functions and predicates in it.

We thus view this condition as necessary for a signature to be "reasonable". In general, a reasonable signature might contain relation symbols as well as function symbols. Nevertheless, we can rule out the possibility of a signature with function symbols of arity at most 1 for which $\mathbf{S}_{\text{len}}$ has quantifier elimination. This is in contrast to the weaker structures that we consider, all of which have quantifier elimination in a relational signature of bounded arity. Let $\mathbf{S}_{\text{len}}^{(n,m)}$ be the expansion of $\mathbf{S}_{\text{len}}$ with all definable predicates of arity at most n, and definable functions of arity at most m. We show the following:

3.1 Theorem *(a) For any $n \geq 0$, and $m = 0, 1$, $\mathbf{S}_{\text{len}}^{(n,m)}$ does not have QE.*

(b) $\mathbf{S}_{\text{len}}^{(1,2)}$, the expansion of $\mathbf{S}_{\text{len}}$ with all unary predicates and binary functions, has QE.

Proof. (a). We assume $\Sigma = \{0, 1\}$ and fix n. Let $m = 0$. The definable property which can not be expressed by a quantifier-free formula is defined as follows. It holds for a tuple $x_1, \ldots, x_{n+1}$ of strings, if there is a position i such that the ith symbol in all x_js is 0.

This is clearly definable in $\mathbf{S}_{\text{len}}$ by $\varphi(x_1, \ldots, x_{n+1})$:

$$\exists y_1, \ldots, y_{n+1} \bigwedge_j y_j \preceq x_j \ \wedge \bigwedge_j L_0(y_j) \ \wedge \bigwedge_{j,k} \text{el}(y_j, y_k) \ .$$

We now assume that φ is a Boolean combination of formulae depending on n variables each. Let these formulae be named as α^i_j, $i \in \{1, \ldots, n+1\}$, $j \in \{1, \ldots, l_i\}$, where α^i_j does *not* have x_i as free variable.

By [14], each α^i_j is given by a letter-to-letter n-automaton A^i_j over Σ^n. Let m be the maximum number of states of the A^i_j.

Now let $p_1 < p_2 < \ldots < p_{n+1}$ be primes with $p_1 > m+1$. Let $\pi_i = \prod_{j \neq i} p_j$, and let $P = \prod_j p_j$ $(= \pi_i \cdot p_i$, for each $i)$.

We now define ω-words w^j, $j = 1, \ldots, n+1$, by

$$w^j[k] \ = \ \begin{cases} 0 & k = 0 (\text{mod } p_j), \\ 1 & \text{otherwise,} \end{cases}$$

where, as for finite strings, $w^j[k]$ denotes the kth position in w^j.

Now fix $i \leq n+1$ and $s \leq l_i$, and consider a run of A^i_s on $(w^j,\ j \neq i)$ (that is, the kth input symbol is $(w^1[k], \ldots, w^{i-1}[k], w^{i+1}[k], \ldots, w^{n+1}[k])$). At every position that is equal to 0 modulo π_i (and only at those positions), the input symbol is $\vec{0} = (0, \ldots, 0)$. Moreover, for any $l \geq 0$ and any $c_1, c_2 > 0$, the input symbols are the same at positions $l + c_1 \cdot \pi_i$ and $l + c_2 \cdot \pi_i$.

We now consider positions equal to 0 modulo π_i; since A^i_s has at most m states, we can find two numbers $d_1 < d_2 \leq m+1$ (depending on s) such that in positions $d_1 \cdot \pi_i$ and $d_2 \cdot \pi_i$ the automaton A^i_s is in the same state q, reading $\vec{0}$. Let $d = (d_2 - d_1) \cdot \pi_1$. Thus, at every position $d_1 \cdot \pi_i + k \cdot d$, the automaton is in the state q, reading $\vec{0}$.

Then for every $l \geq 0$ and every $k \geq 0$, we have that A^i_s is in the same state in positions $d_1 \cdot \pi_i + l$ and $d_1 \cdot \pi_i + l + k \cdot d$, and reads the same symbol in those states. Furthermore, notice that $d_2 \cdot \pi_i \leq (m+1) \cdot \pi_i < p_1 \cdot \pi_i \leq p_i \cdot \pi_i = P$.

Summing up, for each A^i_s, we have two constants, a^i_s $(= d_1 \cdot \pi_i)$ and b^i_s $(= d)$, such that $a^i_s < P$ and the state of A^i_s is the same in positions $a^i_s + l$ and $a^i_s + l + k \cdot b^i_s$, for $l, k \geq 0$.

Now let $C = \max_{i,s} a^i_s$ and $C' = C + P \cdot \prod_{i,s} b^i_s$. We have $C' > P > C$, and all automata A^i_s are in the same state in positions C and C'. In particular, if $w^j[1,k]$ denotes the finite word that consists of the first k positions of w^j, we have that every α^i_j agrees on

$$(w^1[1,C], \ldots, w^{i-1}[1,C], w^{i+1}[1,C], \ldots, w^{n+1}[1,C])$$

and

$$(w^1[1,C'], \ldots, w^{i-1}[1,C'], w^{i+1}[1,C'], \ldots, w^{n+1}[1,C']).$$

The assumption that φ is a Boolean combination of α^i_js now gives us that φ agrees on $(w^1[1,C], \ldots, w^{n+1}[1,C])$ and $(w^1[1,C'] \ldots, w^{n+1}[1,C'])$, which is impossible, since $\varphi(w^1[1,C], \ldots, w^{n+1}[1,C])$ is false ($C < P$ and there is no position with all zeros in it) and $\varphi(w^1[1,C'], \ldots, w^{n+1}[1,C'])$ is true ($C' > P$, and in position P all symbols are 0).

For the case of $m = 1$, it suffices to notice that for any $n > 1$, any quantifier-free formula $\alpha(x_1, \ldots, x_n)$ in $\mathbf{S}_{\text{len}}^{(n,1)}$ is equivalent to a quantifier-free formula in $\mathbf{S}_{\text{len}}^{(n,0)}$. For instance $R(f(x), f(y))$ where R is a definable $\mathbf{S}_{\text{len}}^{(2,0)}$ relation, is equivalent to $R_{f,g}(x, y)$, where $R_{f,g}$ is the $\mathbf{S}_{\text{len}}^{(2,0)}$ relation defined by $R(f(x), f(y))$.

Proof of (b).
Let us assume that Σ contains at least the symbols 0 and 1 and let $\mathbf{S}_{\text{len}}^{+}$ be the expansion of $\mathbf{S}_{\text{len}}$ by the following definable functions and predicates:

- the binary functions $f_\wedge, f_\vee$ which are the bitwise AND and OR of two 0-1 strings u and v, respectively (and ϵ for non-0-1-inputs). When u and v do not have the same length we add sufficiently many 0s to the right of the shorter string. Thus the length of the result is $\max(|u|, |v|)$. E.g., $f_\wedge(101, 11) = 100$;
- the unary function $f_\neg$ which is the bitwise NOT of a 0-1 string;
- for each $\sigma \in \Sigma$, a unary function Fil_σ, where $\text{Fil}_\sigma(w)$ has a 1 at position i iff $w[i] = \sigma$ and a 0 otherwise;
- for each j, k, $j < k$, a unary function $\text{Pat}_{j,k}$ where $\text{Pat}_{j,k}(w)$ has the same length as w and has a 1 at position i iff $i \equiv j (\text{mod} k)$ and a 0 otherwise;
- unary functions LShift, RShift, where RShift(w) is obtained from w by deleting the last (rightmost) symbol and LShift(w) is obtained from w by deleting the first (leftmost) symbol;
- for each j, m, $j < m$, the unary predicate $P_{m,j}$ which will be defined below.

Let R be an n-ary relation over Σ, definable in $\mathbf{S}_{\text{len}}$. Our goal is to find a quantifier-free $\mathbf{S}_{\text{len}}^{+}$-formula φ such that, for each n-tuple $\vec{w}$ of strings, $\mathbf{S}_{\text{len}}^{+} \models \varphi(\vec{w})$ iff $\vec{w} \in R$.

We know from [14, 19] that the relations definable in $\mathbf{S}_{\text{len}}$ are precisely the *regular relations*, that is, precisely those given by letter-to-letter n-automata [14, 19].

Let A be such an automaton for R over the alphabet $(\Sigma \cup \{\#\})^n$ with state set $Q_m = \{q_0, \ldots, q_{m-1}\}$, initial state q_0, transition function δ and set F of accepting states.

An *m-state behavior function* is any function $f : Q_m \to Q_m$. An m-state behavior function can be encoded into a *binary behavior string* $b(f)$ of length $M := m^2$ as follows. For $j, j' < m$, position $jm + j' + 1$ of $b(f)$ is 1 iff $f(q_j) = q_{j'}$.

Let $P_{m,j}$, $j < m$, be the unary predicate which holds for all strings $u = b_1 \cdots b_l$, where each b_i encodes an m-state behavior function f^i and $f^l(\cdots(f^1(q_0))\cdots) = q_j$. As the blocks b_i are of constant length these predicates are regular.

The idea of the proof is to map each block of the input of length m^2 to the string which describes the behavior of A on this block. Whether A accepts the input can then be expressed by means of the predicates $P_{m,j}$.

For a given n-tuple $\vec{w}$, let l be minimal such that $lM \geq |\vec{w}|$ where $|\vec{w}| = \max(|w_1|, \ldots, |w_n|)$ and let $f_{\vec{w}}^i = \delta^*(\cdot, \vec{w}[(i-1)M + 1, iM])$, for $i < l$ and $f_{\vec{w}}^l = \delta^*(\cdot, \vec{w}[(l-1)M + 1, |\vec{w}|])$. Then the state of A after reading $\vec{w}$, starting from the initial state q_0, is j if and only if $b(f_{\vec{w}}^1) \cdots b(f_{\vec{w}}^l) \in P_{m,j}$.

Hence, it is sufficient to find an $\mathbf{S}^+_{\text{len}}$-term θ such that $\theta(\vec{w}) = b(f^1_{\vec{w}}) \cdots b(f^l_{\vec{w}})$. The construction of θ is described in two steps.

First, let $f_{\max}(\vec{w})$ be defined as $\bigvee_{\sigma\in\Sigma} \bigvee_{i=1}^{n} \mathrm{Fil}_\sigma(w_i)$. Here, as in the following the Boolean operators are abbreviations for the respective terms using $f_\vee$, $f_\wedge$, $f_\neg$. Note that $f_{\max}(\vec{w})$ defines a string of length $\max\{|w_i| \mid i \leq n\}$ consisting only of ones. Further let $\widehat{\mathrm{Fil}}_{\sigma,i}(\vec{w})$ be the term $f_{\max}(\vec{w}) \wedge \mathrm{Fil}_\sigma(w_i)$ and let $\widehat{\mathrm{Fil}}_{\#,i}(\vec{w})$ be $f_{\max}(\vec{w}) \wedge \neg(\bigvee_{\sigma\in\Sigma} \mathrm{Fil}_\sigma(w_i))$. Hence, for each symbol $\tau \in \Sigma \cup \{\#\}$, $\widehat{\mathrm{Fil}}_{\tau,i}(\vec{w})$ has a 1 at position j, if the automaton A reads a τ as the j-th symbol of w_i.

Now we are ready to finish the description of θ. For simplicity, we describe θ for the case where $|f_{\max}(\vec{w})|$ is a multiple of M. The general case is slightly more complicated. $\theta(\vec{w})$ has to carry a 1 at a position $(j_0 - 1)M + jm + j' + 1$, for $j, j' < m$, $j_0 > 0$, iff the tuple $\vec{w}[(j_0 - 1)M + 1, j_0 M]$, consisting of n strings of length M is in the set $T(j, j') := \{\vec{s} \mid \delta^*(j, \vec{s}) = j'\}$. Therefore θ can be expressed as

$$\bigvee_{j,j'} \bigvee_{\vec{s}\in T(j,j')} \Big[\mathrm{Pat}_{l,M}(f_{\max}(\vec{w})) \wedge \bigwedge_{i=1}^{l} \bigwedge_{k=1}^{n} \mathrm{RShift}^{(l-i)}(\widehat{\mathrm{Fil}}_{s_k[i],k}(\vec{w})) \wedge \bigwedge_{i=l+1}^{M} \bigwedge_{k=1}^{n} \mathrm{LShift}^{(i-l)}(\widehat{\mathrm{Fil}}_{s_k[i],k}(\vec{w}))\Big], \tag{1}$$

where l is a shorthand for $jm + j' + 1$ and $f^{(i)}$ denotes the i-fold application of f.

The formula says the following: Assume $0 < l \leq M$ and $\vec{s} \in T(j, j')$ fixed, a block $\vec{w}[(j_0 - 1)M + 1, j_0 M]$ of size M is viewed centered in its lth position and thus has $l - 1$ characters on its left and $M - l$ on its right. The last part of the formula checks for the blocks of size M centered in l that equal $\vec{s}$. The test is made separately for the left and right part (this corresponds to the variable i) and for each element of $\vec{s}$ (this corresponds to the variable k). All the results of the tests are shifted to the right for the left part of the block and to the left for the right part in order to align them on the centered position l. Thus the big bitwise $\bigwedge$ is true iff all the previous tests were true and thus iff the block of size M centered in l equals $\vec{s}$.

The first part of the formula filters the blocks we are interested in by keeping only the one centered in $jm + j' + 1$ modulo M. The second bitwise $\bigvee$ will check for all possibilities for $\vec{s} \in T(j, j')$ thus the $jm + j' + 1$ modulo M positions will be equal to 1 iff the corresponding block is a string of $T(j, j')$ as desired. The first bitwise $\bigvee$ ensures that we cover all positions. □

3.1.3 VC-Dimension

Our next result shows another model-theoretic and learning-theoretic shortcoming of $\mathbf{S}_{\text{len}}$: namely, a single formula $\varphi(x, y)$ can define a widely varying collection of relations as we let the parameter x vary. We formalize this through the notion of VC-dimension.

3.2 Proposition *There are definable families in* $\mathbf{S}_{\text{len}}$ *that have infinite VC-dimension.*

Proof. Let $\Sigma = \{0, 1\}$, and let $\varphi(x, y)$ be $\exists z\ (z \prec x \wedge \mathrm{el}(z, y) \wedge L_1(z))$. Let $\mathcal{C}$ be the corresponding definable family: $S \in \mathcal{C}$ iff $S = \varphi(s, \mathbf{S}_{\text{len}})$ for some string s. Let $A_n = \{0^i \mid i < n\}$. Then A_n is shattered by $\mathcal{C}$: given any subset X of A_n, let s_X be a string of length n where the ith character

is 1 iff $0^i \in X$. Then $\varphi(s_X, \mathbf{S}_{\text{len}}) \cap A_n = X$. Since n was arbitrary, this shows that $\mathcal{C}$ has infinite VC-dimension. □

3.2 A star-free algebra based on S

We now turn to the most obvious analog of $\mathbf{S}_{\text{len}}$ for the star-free sets. This is the model $\mathbf{S} = \langle \Sigma^*, \preceq, (l_a)_{a\in\Sigma}\rangle$, which is the most basic model among those studied in the paper. We show that it has remarkably nice behavior: it admits effective QE in a rather small extension to the signature. This immediately tells us that the definable subsets of Σ^* are precisely the star-free languages. We then characterize the n-dimensional definable relations in $\mathbf{S}$ by their closure properties, and by an automaton model.

Note that $\mathbf{S}$ is very close to strings considered as *term algebras*, that is, to $\langle \Sigma, \epsilon, (l_a)_{a\in\Sigma}\rangle$. It is well-known that the theory of arbitrary term algebras is decidable and admits QE [53, 44]. However, adding the prefix relation is not necessarily a trivial addition: for arbitrary term algebras with prefix (subterm), only the *existential* theory is decidable, but the full theory is undecidable [68] (similar results hold for other orderings on terms [23]). The undecidability result of [68] requires at least one binary term constructor; our results indicate that in the simpler case of strings one recovers QE with the prefix relation.

3.2.1 A Normal Form for S

We start with a result that gives a normal form for formulae of FO($\mathbf{S}$).

For that, we need the following predicates, introduced in [52]. For each $L \subseteq \Sigma^*$, let P_L be the set of pairs (x, y) of strings such that $x \preceq y$ and $y - x \in L$. The following lemma is obvious, since it is well-known that star-free sets are first-order definable on string models [54].

3.3 Lemma *For each star free language L, there is a formula $\varphi_L(x, y)$ in* FO($\mathbf{S}$) *which defines P_L.*

We now give a normal form result for FO($\mathbf{S}$).

3.4 Proposition *Every formula $\psi(\vec{x})$ in* FO($\mathbf{S}$) *can be effectively transformed into an equivalent formula which is a disjunction of formulae of the form*

$$\gamma(\vec{x}) \wedge \delta(\vec{x}),$$

where $\gamma(\vec{x})$ is a complete tree-order description over $\vec{x}$ and $\delta(\vec{x})$ is a conjunction of formulae of the form $\varphi_L(t(\vec{x}), t'(\vec{x}))$, where L is star-free, each of $t(\vec{x})$ and $t'(\vec{x})$ is either ϵ or a term of the form $x_i \sqcap x_j$, and $\gamma(\vec{x})$ implies that $t'(\vec{x})$ is an immediate successor of $t(\vec{x})$ in the tree-order.

Proof. The proof is by induction on the structure of ψ. The base case of the induction is handled by noting that the atomic formulae are binary, and the basic formulae $x \prec y$ and and $y = x \cdot a$ are simple cases of $\varphi_L(x, y)$.

Note that for any conjunction $\chi(\vec{x})$ of formulae of the form $t_1(\vec{x})\{\prec, =\}t_2(\vec{x})$ and their negations (where t_1, t_2 are $\sqcap, \epsilon$-terms), there are finitely many complete tree order descriptions $\gamma_i, i \in I$ over $\vec{x}$

which are consistent with χ, and furthermore, all such γ_i's can be effectively found. Thus, any conjunction of two formulae in the normal form, $\xi_1(\vec{x}) \wedge \xi_2(\vec{x})$, can be put in the form $\bigvee_{i\in I} \gamma_i(\vec{x}) \wedge \chi(\vec{x})$, where $\chi(\vec{x})$ is a conjunction of formulae $\varphi_L(t(\vec{x}), t'(\vec{x}))$. This is almost in the normal form, but γ_i may not imply that $t'(\vec{x})$ is an immediate successor of $t(\vec{x})$ in the tree-order. If that is the case, choose some term $t''(\vec{x})$ such that $t(\vec{x}) \prec t''(\vec{x}) \prec t'(\vec{x})$. By a decomposition argument similar to the one used in the proof of Theorem 4.4 in [67], there exists a finite sequence of pairs of star-free languages (L'_j, L''_j) such that $\varphi_L(t(\vec{x}), t'(\vec{x}))$ is equivalent to $\bigvee_j (\varphi_{L'_j}(t(\vec{x}), t''(\vec{x})) \wedge \varphi_{L''_j}(t''(\vec{x}), t'(\vec{x})))$. We can now propagate disjunction and repeat the process until for all formulae of the form $\varphi_L(t(\vec{x}), t'(\vec{x}))$, γ_i implies that $t'(\vec{x})$ is an immediate successor of $t(\vec{x})$. This shows that any Boolean combination of formulae in the normal form can be put in the normal form itself.

Thus, the only nontrivial case is $\psi = \exists x\ \rho(x, \vec{y})$. By induction, we can assume that ρ is in the required form. So we have

$$\psi = \exists x \bigvee_i (\gamma_i(x, \vec{y}) \wedge \bigwedge_j \delta_{ij}(x, \vec{y})),$$

where the γ_i are tree-order descriptions, and the $\delta_{ij}(x, \vec{y}))$ are of the form $\varphi_L(t(x, \vec{y}), t'(x, \vec{y}))$. Thus, it suffices to show how to eliminate x from $\beta(\vec{y}) = \exists x\ \gamma(x, \vec{y}) \wedge \bigwedge_j \varphi_{L_j}(t_j(x, \vec{y}), t'_j(x, \vec{y}))$ where γ is a complete tree-order description, all L_js are star-free, and each t_j, t'_j is a $\epsilon, \sqcap$-term, such that γ implies that t'_j is an immediate successor of t_j in the tree-order. We can further assume without loss of generality that for every pair of terms t_j, t'_j, there is at most one formula of the form $\varphi_{L_j}(t_j, t'_j)$ in the conjunction (if not, one can take the intersection of all the languages in such formulae for these two terms, which will still be star-free). Furthermore, assume γ sets one of the y_l to ϵ (if not, add an extra variable and set it to ϵ in γ). Let $\gamma'(\vec{y})$ be the restriction of γ to $\vec{y}$ (that is, complete tree-order description of $\mathrm{Tree}(\vec{y})$ implied by γ).

We now consider four cases, depending on the relationship between x and $\mathrm{Tree}(\vec{y})$ which is implied by $\gamma(x, \vec{y})$. First, assume that $\gamma(x, \vec{y})$ implies that x is a node in $\mathrm{Tree}(\vec{y})$, that is, ϵ or $y_i \sqcap y_j$ for some i, j. In this case every term of the form $x \sqcap y_k$ can be rewritten as a term that only uses $\vec{y}$ variables, and every formula of the form $\varphi_{L_j}(t_j(x, \vec{y}), t'_j(x, \vec{y}))$ is thus equivalent to a disjunction of formulas $\varphi_{L_j}(\tau_j(\vec{y}), \tau'_j(\vec{y}))$, where τ_j, τ'_j are the result of eliminating x from t_j, t'_j. Thus, β is equivalent to a disjunction of formulas of the form $\gamma'(\vec{y}) \wedge \bigwedge_j \varphi_{L_j}(\tau_j(\vec{y}), \tau'_j(\vec{y}))$.

In the second case, $\gamma(x, \vec{y})$ implies that x is not a prefix of any y_k from $\vec{y}$, and that the meet of x and $\vec{y}$ is a node $y_i \sqcap y_j$ in $\mathrm{Tree}(\vec{y})$. In this case we may have a formula of the form $\varphi_L(y_i \sqcap y_j, x)$ as a conjunct in β. The case is handled just as the previous one, except that we need to deal with the formula $\varphi_L(y_i \sqcap y_j, x)$ (which is the only formula in this case that mentions x). The existence of x satisfying it is guaranteed iff there exists a string in L with a first symbol a such that $(y_i \sqcap y_j) \cdot a$ is not a prefix of any string in $\vec{y}$. Hence we can replace $\varphi_L(y_i \sqcap y_j, x)$ by

$$\bigvee_a \bigwedge_k \neg\varphi_{a\Sigma^*}(y_i \sqcap y_j, y_k),$$

where the conjunction is over all k for which y_k is an immediate successor of $y_i \sqcap y_j$ in the tree-order and the disjunction is over all symbols a for which $L \cap a\Sigma^* \neq \emptyset$.

For the remaining two cases, we need the fact that star-free languages are closed under concatenation. Hence, for star-free languages L' and L'' there exists a star-free language L such that the

following is true: for any two strings $s_0 \prec s_1$, it is the case that there is a string s with $s_0 \prec s \prec s_1$, $s - s_0 \in L'$ and $s_1 - s \in L''$ iff $s_1 - s_0 \in L$.

The proof is straightforward from the fact that star-free languages are precisely those first-order definable in string models [54].

Next, we consider the case when γ implies that x is in the prefix closure of $\vec{y}$, but not a node of Tree($\vec{y}$). That is, we have two nodes $s_0 = y_i \sqcap y_j, s_1 = y_k \sqcap y_l$ of Tree($\vec{y}$) such that there are no other nodes of Tree($\vec{y}$) between them, and $s_0 \prec x \prec s_1$. Notice that any $\epsilon, \sqcap$-term t in $x, \vec{y}$ that involves x can be rewritten as an equivalent term τ in variables $\vec{y}$ or by x. Thus, there are at most two formulae of the form φ_{L_j} where terms mention x: these are $\varphi_{L'}(s_0, x)$ and $\varphi_{L''}(x, s_1)$ for some star-free L', L''. Hence, $\beta(\vec{y})$ is equivalent to

$$\gamma'(\vec{y}) \wedge \bigwedge_m \varphi_{L_m}(\tau_m(\vec{y}), \tau'_m(\vec{y})) \wedge \exists x \ ((s_0 \prec x \prec s_1) \wedge \varphi_{L'}(s_0, x) \wedge \varphi_{L''}(x, s_0)),$$

where the big conjunction is over formulae φ_{L_j} and terms do not mention x. By the claim, there is a star-free language L such that $\exists x \ ((s_0 \prec x \prec s_1) \wedge \varphi_{L'}(s_0, x) \wedge \varphi_{L''}(x, s_0))$ is equivalent to $s_1 - s_0 \in L$, that is, $\varphi_L(y_i \sqcap y_j, y_k \sqcap y_l)$, which shows that $\beta(\vec{y})$ can be put in the required form.

The last case is when γ specifies that x is not in the prefix closure of $\vec{y}$, and the meet of x and Tree($\vec{y}$) is a string s between two nodes of Tree($\vec{y}$). That is, for two consecutive nodes $s_0 = y_i \sqcap y_j, s_1 = y_k \sqcap y_l$ of Tree($\vec{y}$) we have $s_0 \prec x \sqcap s_1 \prec s_1$. In particular, $x \sqcap s_1 = x \sqcap y_k = x \sqcap y_l$. We thus have formulae $\varphi_{L_1}(s_0, x \sqcap y_k), \varphi_{L_2}(x \sqcap y_k, y_l \sqcap y_k)$ and $\varphi_{L'}(x \sqcap y_k, x)$ as conjuncts of β, for some star-free languages L_1, L_2, L'. We may assume that other subformulae of the form φ_L do not mention x. Let $\chi(\vec{y})$ be the conjunction of all those other subformulae. Then $\beta(\vec{y})$ is equivalent to

$$\bigvee_{a \in \Sigma} \exists z \gamma'(\vec{y}) \wedge (s_0 \prec z \prec s_1) \wedge \chi(\vec{y}) \wedge \varphi_{L_1}(s_0, z) \wedge \varphi_{L_2 \cap (a\Sigma^*)}(z, s_1) \wedge \exists x (z \prec x \wedge \varphi_{L' - a\Sigma^*}(z, x))$$

(z plays the role of $x \sqcap s_1$, and the disjunction ensures that the first letters of $s_1 - z$ and $x - z$ are different). Let $\Sigma' = \{a \in \Sigma \mid L' - a\Sigma^* \neq \emptyset\}$. Then we obtain that $\beta(\vec{y})$ is equivalent to

$$\bigvee_{a \in \Sigma'} \gamma'(\vec{y}) \wedge \exists z (s_0 \prec z \prec s_1) \wedge \chi(\vec{y}) \wedge \varphi_{L_1}(s_0, z) \wedge \varphi_{L_2 \cap (a\Sigma^*)}(z, s_1),$$

from which z can be eliminated just as in the previous case. This concludes the proof. □

We now give an illustration of the normal form. Suppose we have a formula $\psi(x, y) = \exists z \ (z \prec x \wedge z \prec y \wedge L_a(z))$. In other words, there is a proper prefix of $x \sqcap y$ whose last letter is a. Let L be the language that consists of strings that have such a prefix. It is a star-free languages, since it is definable by an FO formula over string models: $\exists i \exists j \ (i < j \wedge P_a(i))$.

To produce the normal form for ψ, we consider four different possibilities for x and y: $x = y$, $x \prec y$, $y \prec x$, and $x \not\prec y, y \not\prec x, x \neq y$, and for each we state that the meet of x and y, in the corresponding tree, belongs to L. That is, the formula is:

$$\begin{array}{lll} & \big((\epsilon \prec x \wedge x = y) & \wedge \ \varphi_L(\epsilon, x)\big) \\ \vee & \big((\epsilon \prec x \wedge x \prec y) & \wedge \ \varphi_L(\epsilon, x)\big) \\ \vee & \big((\epsilon \prec y \wedge y \prec x) & \wedge \ \varphi_L(\epsilon, y)\big) \\ \vee & \big((\epsilon \prec x \sqcap y \wedge \neg(x \prec y) \wedge \neg(y \prec x) \wedge \neg(x = y)) & \wedge \ \varphi_L(\epsilon, x \sqcap y)\big) \ . \end{array}$$

3.2.2 Quantifier Elimination

Let $\mathbf{S}^+$ be the expansion of $\mathbf{S}$ to the signature that contains ϵ, $\sqcap$ and a binary predicate P_L for each star-free language L. Note that $\mathbf{S}^+$ is a *definable* expansion of $\mathbf{S}$, as all additional functions and predicates are definable. From the normal form we now immediately obtain:

3.5 Theorem $\mathbf{S}^+$ *admits quantifier elimination.*

Remark. As mentioned above, there is no need to nest the $\sqcap$-operator. Therefore, $\mathbf{S}^+$ can be turned into a relational signature that admits quantifier elimination as follows. For each star-free L, let P'_L be the set of tuples (s_1, s_2, s_3, s_4) of strings for which $P_L(\sqcap(s_1, s_2), \sqcap(s_3, s_4))$. Note, that $\sqcap(s_1, s_2) \preceq \sqcap(s_3, s_4)$ can be expressed as $P_{\Sigma^*}(\sqcap(s_1, s_2), \sqcap(s_3, s_4))$. It is straightforward to check that this signature admits quantifier elimination. In the same way, the quantifier elimination results in the remainder of the paper can be turned into quantifier-elimination results in a relational signature.

Note also that $\mathbf{S}^+$ could be considered as an expansion of $\mathbf{S}$ with either functions l_a or predicates L_a in the signature. In the latter case, predicates L_a are not needed as $L_a(x)$ iff $P_{\Sigma^* a}(\epsilon, x)$.

Another corollary of the normal form is that in the language of $\mathbf{S}$, it suffices to use only bounded quantification. That is, we introduce *bounded quantifiers* of the form $\exists x \preceq y$ and $\forall x \preceq y$ (where $\exists x \preceq y\ \varphi$ means $\exists x\ x \preceq y \wedge \varphi$), and let $\mathrm{FO}_b(\mathbf{S})$ be the restriction of $\mathrm{FO}(\mathbf{S})$ to formulae $\varphi(y_1, \ldots, y_k)$ in which all quantifiers are of the form $Qx \preceq y_i$. From the normal form and the fact that each φ_L can be defined with bounded quantifiers, we obtain:

3.6 Corollary $\mathrm{FO}_b(\mathbf{S}) = \mathrm{FO}(\mathbf{S})$.

Finally, we characterize $\mathbf{S}$-definable subsets of Σ^* and $(\Sigma^*)^k$. Given a subset $R \subseteq (\Sigma^*)^k$ and a permutation π on $\{1, \ldots, k\}$, by $\pi(R)$ we mean the set $\{(s_{\pi(1)}, \ldots, s_{\pi(k)}) \mid (s_1, \ldots, s_k) \in R\}$.

3.7 Corollary

a) *A language $L \subseteq \Sigma^*$ is definable in $\mathbf{S}$ iff it is star-free.*

b) *The class of relations definable over $\mathrm{FO}(\mathbf{S})$ is the minimal class containing the empty set, $\{\epsilon\}$, $\{a\}$, for $a \in \Sigma$, $\preceq$, $\sqcap$, and closed under Boolean operations, Cartesian product, permutation, and the operation $*$ defined by $L_1 * L_2 = \{(s_1, s_1 \cdot s_2) \mid s_1 \in L_1, s_2 \in L_2\}$ for $L_1, L_2 \subseteq \Sigma^*$.*

Proof. a) $\mathbf{S}^+$ formulae in one free variable are Boolean combinations of $P_L(\epsilon, x)$, for L star-free, and thus they define only star-free languages.

b) For one direction notice that ϵ, $\{a\}$, $\prec$, $\sqcap$ are definable in $\mathrm{FO}(\mathbf{S})$, and that $\mathrm{FO}(\mathbf{S})$ is closed under Boolean operations, permutation and Cartesian product. The closure under $*$ is an easy consequence of Lemma 3.3 as $L_1 * L_2$ corresponds to $\{(x, y) \mid \varphi_{L_1}(\epsilon, x) \wedge \varphi_{L_2}(x, y)\}$. The other direction follows from the normal form. □

Note that the projection operation is not needed in the closure result above.

3.2.3 Automata

We now give an automaton model characterizing definability in FO(**S**). This automaton model corresponds exactly to the counter-free variant of *regular prefix automaton* as defined in [4].

Let us recall the definition of regular prefix automata. Let A be a finite non-deterministic automaton on strings with state set Q, transition relation δ and initial state q_0. We construct from A an automaton $\hat{A} = (\Sigma, Q, q_0, F, \delta)$ accepting n-tuples $\vec{w} = (w_1, \cdots, w_n)$ of strings in the following way. F is a subset of Q^n which denotes the accepting states of $\hat{A}$. Let $\mathit{prefix}(\vec{w})$ be the set of all prefixes of all w_i. A *run* of $\hat{A}$ over $\vec{w}$ is a mapping h from $\mathit{prefix}(\vec{w})$ to Q which assigns to every node $\alpha \in \mathit{prefix}(\vec{w})$ a state $q \in Q$ such that $h(\epsilon) = q_0$ and, $\beta = l_a(\alpha)$ implies $h(\beta) \in \delta(h(\alpha), a)$. The run is accepting if $(h(w_1), \cdots, h(w_n)) \in F$. The n-tuple $\vec{w}$ is accepted by $\hat{A}$ if there is an accepting run of $\hat{A}$ over $\vec{w}$. See [4] for more details.

For each finite non-deterministic automaton A a corresponding automaton $\hat{A}$ is called a *regular prefix automaton* (RPA). The subset of $(\Sigma^*)^n$, $n \in \mathbb{N}$, it defines is called a *regular prefix relation* (RPR).

We say that $\hat{A}$ is counter-free (CF-PA) if A is counter-free. The following shows that the relations definable in FO(**S**) are exactly those recognizable by a CF-PA.

3.8 Proposition *A relation is definable in* FO(**S**) *if and only if it is definable by a counter-free prefix automaton.*

Proof. One direction follows from Corollary 3.7 as it is easy to verify that counter-free prefix automata can recognize the empty set, $\{\epsilon\}$, $\{a\}$ $a \in \Sigma$, $\{(u, v) \mid u \preceq v\}$, $\{(u, v, w) \mid u \sqcap v = w\}$, and are closed under Boolean operations, Cartesian product, permutation, and $*$.

For the opposite direction let $\hat{A}$ be a CF-PA accepting the relation R of arity n. We show that R can be defined by an FO(**S**) formula φ. Let Q be the set of states of A. If q_1, q_2 are two states in Q, let $L(q_1, q_2)$ be the set of strings w such that A can get from state q_1 to state q_2 by reading w. Because A is counter-free $L(q_1, q_2)$ is a star-free language.

The formula φ is a disjunction over formulae $\gamma(\vec{x}) \wedge \psi_\gamma(\vec{x})$, where γ cycles through all complete tree-order descriptions. Each formula $\psi_\gamma(\vec{x})$ is a disjunction over all possible assignments of states to the (at most $2n$) strings of $\mathrm{Tree}(\vec{x})$. For each such assignment it checks that the vector of states at $\vec{x}$ is accepting and that the states are consistent, i.e., that, for each pair (y, z) of successive elements of $\mathrm{Tree}(\vec{x})$, the path from y to z fulfills $P_L(q_1, q_2)$ where q_1 and q_2 are the states at y and z in the assignment under consideration, respectively. □

3.2.4 VC-dimension and Isolation

We defined the notions of isolation and VC dimension in Section 2; these notions are very important for the database part of the paper, as they provide strong bounds on the expressiveness of various relational calculi. The notion of finite VC-dimension, coming originally from statistics and machine learning [5], is of independent interest, as it states that families definable over some structures on strings could be learned effectively.

We have seen that $\mathbf{S}_{\text{len}}$ has infinite VC-dimension. It turns out that all other structures we consider here, have finite VC-dimension. To prove this, we have to introduce some new machinery, which is presented next. After that, we show that $\mathbf{S}$ has finite VC-dimension.

3.9 Lemma *Let M be a model with the isolation property. Then its definable families have finite VC-dimension.*

Proof. We give two proofs of this result, one is complexity-theoretic and one is model-theoretic. We start with the complexity-theoretic proof. Assume that M does not have finite VC dimension. By [51] it has the *independence property*, and by [63], there is a single formula $\varphi(\vec{x},\vec{y})$ (in fact, $\varphi(\vec{x},y)$) that has the independence property: that is, for every n, there is a set $F_n \subseteq M$ of size n such that for every $X \subseteq F_n$, there is $\vec{x}_X$ such that for any $y_0 \in F_n$, $\varphi(\vec{x}_X, y_0)$ iff $y_0 \in X$.

Next consider an expansion of M with one unary predicate U, and one binary predicate E. Let Φ be

$$\begin{array}{rl} & \forall v,w\Big(E(v,w) \to (U(v) \wedge U(w))\Big) \\ \wedge & \neg\exists \vec{s}_1,\vec{s}_2 \left(\begin{array}{rl} & \forall v U(v) \leftrightarrow (\varphi(\vec{s}_1,v) \vee \varphi(\vec{s}_2,v)) \\ \wedge & \forall v,w\ (U(v) \wedge U(w) \wedge \varphi(\vec{s}_1,v) \wedge \varphi(\vec{s}_2,w)) \to \neg E(v,w) \end{array} \right). \end{array}$$

The first conjunct says that E is a graph whose nodes are in the set U. The second says that, assuming $U \subseteq F_n$, there cannot be two subsets of U such that there are no E-edges between them. Thus, if U is a finite subset of F_n, Φ says that E is connected.

The isolation property [8, 32] implies that Φ can be expressed by a sentence Ψ of the form $Qz_1 \in U \ldots Qz_l \in U\alpha(\vec{z})$ over all *finite* U, where α is a Boolean combination of E,U-atomic formulae, and formulae $\gamma(\vec{z})$ in the language of M.

Next, for each n, fix a 1-to-1 mapping $\pi : \{1,\ldots,n\} \to F_n$ and for each γ appearing in Ψ, define $P^n_\gamma(\vec{z})$ on $\{1,\ldots,n\}$ to contain all the tuples $\vec{n}$ such that $\gamma(\pi(\vec{n}))$ is true. Let then Ψ_n be the sentence in the language of E and all P^n_γ of the form $Qz_1 \ldots Qz_l\alpha'$ where α' is obtained from α by replacing each $U(\cdot)$ by true, and each $\gamma(\vec{z})$ by $P^n_\gamma(\vec{z})$. It then follows that for a graph E on $\{1,\ldots,n\}$, $E \models \Psi_n$ iff E is connected. However, this implies that connectivity is in non-uniform AC^0, which is false [26]. This concludes the proof.

Second proof. We now give another, model-theoretic proof. For a formula $\varphi(\vec{x},\vec{y})$ and set $A \subseteq M$, a *φ-type over A* is a maximal consistent (w.r.t. $\mathrm{Th}(M)$) set of formulae of the form $\varphi(\vec{x},\vec{a})$ with $\vec{a}$ a tuple over A. For $\vec{c}$ in M and A as above, we can then talk about the *φ-type of $\vec{c}$ over A*, denoted $tp_\varphi(\vec{c}/A)$.

Let $\varphi(\vec{x},\vec{y})$ be a formula over M. We next show that there are integers n and K such that for any finite set A, there are at most $K|A|^n$ φ-types over A.

To prove this we first claim that for each φ there is a formula $\gamma_\varphi(\vec{x},\vec{z})$ and an integer n such that for every finite set A, and any vector $\vec{s}$, there is an n-element subset X of A such that $tp_\varphi(\vec{s}/A)$ is isolated by $tp_{\gamma_\varphi}(\vec{s}/X)$.

Indeed, assume that for some φ there was no such n and γ. Then for each γ and each n there exists a finite set A^n_γ and a vector $\vec{s}^n_\gamma$ such that for any finite subset X of A^n_γ of size $< n$, $tp_\varphi(\vec{s}^n_\gamma/A^n_\gamma)$ is not isolated by $tp_\gamma(\vec{s}^n_\gamma/X)$. Then, by compactness, we get a pseudo-finite set W_γ (the ultraproduct of the $(A^n_\gamma)_{n\in\mathbb{N}}$) and a vector $\vec{s}_\gamma$ (the ultraproduct of the $(\vec{s}^n_\gamma)_{n\in\mathbb{N}}$) in a model of $\mathrm{Th}(M)$ such that for

any finite set X of W_γ, $tp_\varphi(\vec{s}_\gamma/W_\gamma)$ is not isolated by $tp(\vec{s}_\gamma/X)$. Then, by compactness again, we get another model of $\mathrm{Th}(M)$ with a pseudo-finite set W and $\vec{s}$, such that for any countable subset X of W, $tp(\vec{s}/W)$ is not isolated by $tp(\vec{s}/X)$, which contradicts isolation.

Now let K be $2^{n^{|\vec{z}|}}$. It is easy to see that n and K work. There are at most $|A|^n$ subsets X from A of size n. For each fixed set X of size n, there are at most $n^{|\vec{z}|}$ formulae of the form $\gamma(\vec{x},\vec{e})$ with $\vec{e} \in X$, and hence there are at most K γ-types over X. Since the φ-type of a vector $\vec{c}$ from M is determined by the choice of the set X whose γ-type isolates it and the γ-type of $\vec{c}$ over X, it follows that there are at most $K|A|^n$ types.

Now let $\mathcal{C}$ be the family definable by $\varphi(\vec{x},\vec{y})$. If a finite set A is shattered by members of $\mathcal{C}$, then the number of φ-types over A is $2^{|A|}$. Hence, arbitrarily large finite sets cannot be shattered by $\mathcal{C}$. □

Next, we show the following.

3.10 Proposition $\mathrm{Th}(\mathbf{S})$ *has the strong isolation property.*

Proof. Let M be a model of $\mathrm{Th}(\mathbf{S})$, W be a pseudo-finite set of elements of M, and $a \in M$. We exhibit a finite subset W_0 of W such that $tp_M(a/W_0)$ isolates $tp_M(a/W)$.

Note that for each finite set X, the elements $\mathrm{Meet}(a,X), \mathrm{Meet}_-(a,X)$ and $\mathrm{Meet}_+(a,X)$ can be described by means of formulae of $\mathrm{FO}(\mathbf{S})$: $\mathrm{Meet}(a,X)$ is the largest prefix of a which is in the prefix closure of X, and $\mathrm{Meet}_-(a,X)$, $\mathrm{Meet}_+(a,X)$ are the nodes of $\mathrm{Tree}(X)$ (meets of two elements of X) which are closest to $\mathrm{Meet}(a,X)$. Hence, such elements exist for W, since W is pseudo-finite. Let $w_1, w_2, w_3, w_4 \in W$ be such that $w_1 \sqcap w_2 = \mathrm{Meet}_-(a,W)$ and $w_3 \sqcap w_4 = \mathrm{Meet}_+(a,W)$. Take $W_0 = \{w_1, w_2, w_3, w_4\}$.

We know that any formulae of $\mathrm{FO}(\mathbf{S})$ can be put in the normal form described in Proposition 3.4. Thus a type of a over W is entirely defined by the tree structure of $a \cup W$ and the paths between definable nodes of that tree. If we fix W, we conclude that the paths between $\mathrm{Meet}(a,W)$, $\mathrm{Meet}_-(a,W)$, $\mathrm{Meet}_+(a,W)$ and a completely define $tp_M(a/W)$. Because $tp_M(a/W_0)$ already describes all the paths between $\mathrm{Meet}(a,W)$, $\mathrm{Meet}_-(a,W)$, $\mathrm{Meet}_+(a,W)$ and a, the result follows. □

Combining Proposition 3.10 and Lemma 3.9, we conclude that the model $\mathbf{S}$, unlike $\mathbf{S}_{\mathrm{len}}$, has learnable definable families.

3.11 Corollary *Every definable family in* $\mathbf{S}$ *has finite VC-dimension.*

3.3 A star-free algebra based on $\mathbf{S}_{\mathrm{left}}$

We now study an example of a star-free algebra, in which the n-ary relations in the algebra are more complex than those definable over $\mathbf{S}$. Recall that $\mathbf{S}_{\mathrm{left}} = \langle \Sigma^*, \preceq, (l_a)_{a\in\Sigma}, (f_a)_{a\in\Sigma}\rangle$; that is, in this structure one can add characters on the right as well as on the left.

Without the prefix relation, this structure was studied in [16, 60], as a model of queues. A quantifier-elimination result was proved in [60], by extending quantifier-elimination for term algebras (in fact [60] showed that term algebras with queues admit QE). However, as in the case of $\mathbf{S}$, which differs

from strings as terms algebras in that it has the prefix relation, the prefix relation complicates things considerably.

We start with the easy observation that $\mathrm{FO}(\mathbf{S}_{\mathrm{left}})$ expresses more relations that $\mathrm{FO}(\mathbf{S})$. Indeed, the graph of f_a, $F_a = \{(x, a \cdot x) \mid x \in \Sigma^*\}$ is not expressible in $\mathrm{FO}(\mathbf{S})$, which can be shown by a simple game argument. More precisely, given a number k of rounds, let $n = 2^k + 1$ and consider the game on the tuples $(0^n, 10^n)$ and $(0^{n+1}, 10^n)$. By Corollary 3.6 it is sufficient to play on the prefixes of the participating strings. The duplicator has a trivial winning strategy on the strings 10^n and a well-known winning strategy on 0^n versus 0^{n+1}.

3.3.1 Quantifier Elimination

Let $\mathbf{S}^+_{\mathrm{left}}$ be the extension of $\mathbf{S}_{\mathrm{left}}$ with the same (definable) functions and predicates we added to $\mathbf{S}^+$ (that is, a constant ϵ for the empty string, the binary function $\sqcap$ for the longest common prefix, the predicate $P_L(x, y)$ for each star-free language L), and the unary function $x \mapsto x - a$, for each $a \in \Sigma$ (which is also definable).

3.12 Theorem $\mathbf{S}^+_{\mathrm{left}}$ *admits quantifier elimination.*

In the rest of the section, we prove Theorem 3.12. Let $\Omega_{\mathbf{S}^+}$ and $\Omega_{\mathbf{S}^+_{\mathrm{left}}}$ be the first-order signature of $\mathbf{S}^+$ and $\mathbf{S}^+_{\mathrm{left}}$, respectively. Let M be an ω-saturated model over $\Omega_{\mathbf{S}^+_{\mathrm{left}}}$ elementary equivalent to $\mathbf{S}^+_{\mathrm{left}}$. It suffices to prove quantifier elimination in M. Note that M can have both finite and infinite strings.

We next need the following standard result:

Claim 1 *If there exists a formula which does not admit quantifier elimination in M, then there exist two tuples of elements in M which have the same atomic type but not the same type.*

Proof of Claim 1. Let $\varphi(\vec{x}) \in \mathrm{FO}(\mathbf{S}^+_{\mathrm{left}})$, and let Q enumerate all quantifier free formulae over $\Omega_{\mathbf{S}^+_{\mathrm{left}}}$ realizable in M. Let $\Gamma_\varphi(\vec{x}_1, \vec{x}_2)$ be the type asserting $\bigwedge_{\psi \in Q}(\psi(\vec{x}_1) \leftrightarrow \psi(\vec{x}_2)) \ \wedge \ \neg(\varphi(\vec{x}_1) \leftrightarrow \varphi(\vec{x}_2))$.

We show that if φ is not equivalent to a quantifier-free formula then Γ_φ is satisfied in M. Towards a contradiction assume Γ_φ is not satisfied in M. Since M is ω-saturated, by compactness it follows that there is a finite set $J \subseteq Q$ such that

$$\forall \vec{x}_1 \ \forall \vec{x}_2 \ [(\bigwedge_{i \in J} \psi_i(\vec{x}_1) \leftrightarrow \psi_i(\vec{x}_2)) \rightarrow (\varphi(\vec{x}_1) \leftrightarrow \varphi(\vec{x}_2))]$$

holds in M. For $K \subseteq J$ let χ_K be $\bigwedge_{i \in K} \psi_i \wedge \bigwedge_{i \in J - K} \neg\psi_i$.

Let G be $\{I \subseteq J \mid M \models \forall \vec{x} \ \chi_I(\vec{x}) \rightarrow \varphi(\vec{x})\}$ and $\rho = \bigvee_{I \in G} \chi_I$. To get a contradiction we show that ρ is equivalent to φ in M. Let $\vec{c}$ be a tuple of M with $M \models \varphi(\vec{c})$. Let $L = \{i \in J \mid M \models \psi_i(\vec{c})\}$. If a tuple $\vec{d}$ from M satisfies χ_L then for each $i \in J$, $M \models \psi_i(\vec{c}) \leftrightarrow \psi_i(\vec{d})$. By the choice of J we can conclude that $M \models \varphi(\vec{c}) \leftrightarrow \varphi(\vec{d})$, hence $M \models \varphi(\vec{d})$. Therefore $L \in G$ and $M \models \rho(\vec{c})$. On the other hand, by the definition of G and ρ it follows immediately that $M \models \rho(\vec{c})$ implies $M \models \varphi(\vec{c})$. Hence, φ and ρ are equivalent in M, the desired contradiction. The claim is proved. □

Thus, to prove QE, we must show that every two tuples of elements of M that have the same atomic type, have the same type.

Define a *nice term* of $\Omega_{\mathbf{S}^+_{\text{left}}}$ as a term of the form $t(x) = x - a + b$ (meaning $(x - a) + b$), where a and b are finite strings.

We define two relations $\equiv$ and $\equiv_1$ on tuples (of the same length) of strings as follows.

- $\vec{c} \equiv \vec{d}$ for n-tuples $\vec{c}$ and $\vec{d}$ iff for all sequences $i_1, \ldots, i_k$ from $\{1, \ldots, n\}$ and all sequences $t_1, \ldots, t_k$ of nice terms:
$$atp_{\mathbf{S}^+}(t_1(c_{i_1}), \ldots, t_k(c_{i_k})) = atp_{\mathbf{S}^+}(t_1(d_{i_1}), \ldots, t_k(d_{i_k})) \ .$$
- $(c', \vec{c}) \equiv_1 (d', \vec{d})$ for n-tuples $\vec{c}$, $\vec{d}$ and strings c', d' iff for all sequences $i_1, \ldots, i_k$ from $\{1, \ldots, n\}$ and all sequences $t_1, \ldots, t_k$ of nice terms:
$$atp_{\mathbf{S}^+}(c', t_1(c_{i_1}), \ldots, t_k(c_{i_k})) = atp_{\mathbf{S}^+}(d', t_1(d_{i_1}), \ldots, t_k(d_{i_k})) \ .$$

Of course, $(c', \vec{c}) \equiv (d', \vec{d})$ implies $(c', \vec{c}) \equiv_1 (d', \vec{d})$, as the identity is a nice term. We will show that these two relations coincide.

We will show in Lemma 3.14 a stronger result than what is needed by Claim 1 in order to prove Theorem 3.12. Indeed we will show that $\equiv$ has the back-and-forth property. In order to simplify the strategy for the $\equiv$ game we first show in Lemma 3.13 that it is enough to have a strategy for the $\equiv_1$ game. Lemma 3.13 is proved by rewriting rules on the atomic formulas that get rid of nice terms containing c'.

3.13 Lemma *If $(c', \vec{c}) \equiv_1 (d', \vec{d})$, then also $(c', \vec{c}) \equiv (d', \vec{d})$.*

Once the equivalence of $\equiv_1$ and $\equiv$ is established, we will show that they have the back-and-forth property, from which quantifier-elimination will follow.

Proof of Lemma 3.13. We start with a few observations. It is easy to see that for every atomic formula of $\mathrm{FO}(\mathbf{S}^+_{\text{left}})$, there is an equivalent $\mathrm{FO}(\mathbf{S}^+_{\text{left}})$ formula in which every term is a meet of two nice terms (addition and subtraction of $t_1 \sqcap t_2$ can be pushed back into t_1 and t_2, while multiple meets can be eliminated by adding disjunctions of tree-ordering formulae considering all possible cases). Notice also that atomic formulae of the form $t \preceq t'$ where t and t' are terms are equivalent to $P_{\Sigma^*}(t, t')$, and $t \prec t'$ is equivalent to $P_{\Sigma^+}(t, t')$. Thus, we can assume that no symbols $\preceq$ and $\prec$ occur.

We call a nice term $t(x) = x - a + b$ *empty* if $a = b = \epsilon$.

The proof of Lemma 3.13 is done by rewriting atomic formulas in order to get rid of nice terms from one of the variables. We will proceed by a case analysis based on the rewriting rules presented in the next 4 claims.

The first claim shows how to replace a single nice terms from a distinguished variable s'. The proof is straightforward.

Claim 2 *1. Let s, s' be in M and let a, b be finite strings and let L be star-free. Then $P_L(s, s' - a + b)$ is true in M iff one of the following conditions holds.*

- $s \preceq b$, *and* $s' - a + (b - s) \in L$
- $a \preceq s'$, $b \preceq s$ *and* $P_L(s - b + a, s')$.
  Notice that in the first case above s *is finite, and thus the condition over* s' *is expressible in* FO($\mathbf{S}$)*).*

2. *Let* s, s' *be in* M *and let* a, b *be finite strings and let* L *be star-free. Then* $P_L(s - a + b, s')$ *is true in* M *iff one of the following conditions holds.*

   - $a \not\preceq s$, $b \preceq s'$ *and* $s' - b \in L$;
   - $a \preceq s$ *and* $P_L(s, s' - b + a)$.

The next claim shows how to get rid of terms of the form $t(s) \sqcap s$ from distinguished variable s.

Claim 3 *Let* s *be an element of* M, t *a nice term over* $\Omega_{\mathbf{S}^+_{\text{left}}}$. *Let* $s' = t(s) \sqcap s$. *There is a quantifier-free* FO($\mathbf{S}^+$) *formula* $\varphi_{s,t}(x, y)$ *such that* $\varphi_{s,t}(s, s')$ *and* $\forall x\ \varphi_{s,t}(x, y) \to y = x \sqcap t(x)$ *hold in* M.

Proof of Claim 3. Let a, b finite strings such that $t(x) = x - a + b$. If $s' = s \sqcap (s - a + b)$ is finite, then $\varphi_{s,t}(x, y)$ is $s' = (x \sqcap (x - a + b)) \wedge y = s'$. Here, $s' = x \sqcap (x - a + b)$ can be expressed in FO($\mathbf{S}$) by $(s' \preceq x) \wedge (s' - b + a \preceq x) \wedge \bigwedge_{\alpha \in \Sigma} \neg(s' \cdot \alpha \preceq x \wedge s' \cdot \alpha - b + a \preceq x)$. If s' is infinite, then let $n = |a|$ and $m = |b|$. We have $b \preceq s$, $a \preceq s$, and $s[n + i] = s[m + i]$ for $i \in \mathbb{N}$. For given n, m it is possible to define an FO($\mathbf{S}$) formula $\psi(x, y)$ which is true if and only if y is maximal such that $y \preceq x$, $|y| > m$, and $x(n + i) = x(m + i)$, where $i = |y| - m$. Then we let $\varphi_{s,t}(x, y)$ be $a \preceq x \wedge b \preceq x \wedge \psi(x, y)$. It is easy to verify that $\varphi_{s,t}(s, s')$ holds and that $\varphi_{s,t}(x, y)$ implies $y = x \sqcap t(x)$. Finally, by quantifier-elimination in FO($\mathbf{S}^+$) $\varphi_{s,t}$ can be made quantifier-free. □

The following is the analog of the preceding claim for terms of the form $t(s \sqcap s')$.

Claim 4 *Let* t, t' *be nice terms and* L *star-free. Assume that there are strings* s, s', s'' *such that* $P_L(t(s \sqcap s'), t'(s \sqcap s''))$ *holds. Then there is an* FO($\mathbf{S}$) *formula* $\rho(x, y, z)$ *such that* $\rho(s, s', s'')$ *holds and such that, for all* r, r', r'' *in* M, $\rho(r, r', r'')$ *implies* $P_L(t(r \sqcap r'), t'(r \sqcap r''))$.

Proof of Claim 4. Let $t(x) = x - a + b$ and $t'(x) = x - a' + b'$. First of all, if $P_L(t(s \sqcap s'), t'(s \sqcap s''))$ holds then from Claim 2 we have either:

1. $t(s \sqcap s') \preceq b'$ and $(s \sqcap s'') - a' + (b' - t(s \sqcap s')) \in L$, or
2. $a' \preceq s \sqcap s''$, $b' \preceq t(s \sqcap s')$, and $P_L(t(s \sqcap s') - b' + a', s \sqcap s'')$.

Consider the first case. Notice that it implies that $t(s \sqcap s')$ is a finite string. Hence, the second condition says that $s \sqcap s'' \in L'$, for the star-free set L' of strings z with $z - a' + (b' - t(s \sqcap s')) \in L$. The first condition holds iff (a) a is not a prefix of $s \sqcap s'$ and $b \preceq b'$ or (b) $s \sqcap s'$ is finite in which case $t(s \sqcap s') \preceq b'$ can be easily expressed in FO($\mathbf{S}$).

Consider now the second case. The conditions $a' \preceq s \sqcap s''$ and $b' \preceq t(s \sqcap s')$ can be easily expressed in FO($\mathbf{S}$). It remains to express $P_L(t(s \sqcap s') - b' + a', s \sqcap s'')$. As before, we can assume that the first term is nice, i.e., we only have to show how $P_L(t(s \sqcap s'), s \sqcap s'')$, where $t(x) = x - a + b$, can be expressed.

We distinguish two subcases.

If $t(s \sqcap s')$ is finite then the corresponding FO($\mathbf{S}$) formula is obtained similarly to the previous case.

Assume now $t(s \sqcap s')$ infinite. In this case, as $s \sqcap s'$ is a prefix of s (and therefore $s \sqcap s' \preceq s \sqcap s''$ or $s \sqcap s'' \preceq s \sqcap s'$ holds), it is sufficient to express that the suffix of $s \sqcap s''$ relative to its prefix of length $|s \sqcap s'| - |b| + |a|$ is in L. This can clearly be expressed in FO($\mathbf{S}$). □

Let $\varphi(x, y)$ be an FO($\mathbf{S}$) formula. If in M, there is at most one s' for each s such that $\varphi(s, s')$ holds, then we call φ *functional*, as φ defines a partial function f_φ on M by $f_\varphi(s) = s'$ if $\varphi(s, s')$ holds. Note that $\varphi_{s,t}$ of Claim 3 is functional. We call a term of the form $f_\varphi(x)$ where φ is functional a $\Omega_{\mathbf{S}}$-*function term*, if for each s in M, $f_\varphi(s) \preceq s$. Let $\Omega_{\mathbf{S}^{++}_{\text{left}}}$ be the signature obtained from $\Omega_{\mathbf{S}^{+}_{\text{left}}}$ by adding all $\Omega_{\mathbf{S}}$-function terms.

The next claim shows that in attempting to eliminate terms with "$-$" from distinguished variable y, it suffices to deal with terms of a particularly simple form.

Claim 5 *Let s be an element of M. For every atomic* FO($\mathbf{S}^{+}_{\text{left}}$) *formula $\varphi(y, \vec{x})$ there is a quantifier-free* FO($\mathbf{S}^{++}_{\text{left}}$) *formula $\varphi'(y, \vec{x})$, such that for all $\vec{r}$ from M, $\varphi(s, \vec{r})$ holds if and only if $\varphi'(s, \vec{r})$ holds. We can also ensure that y appears in φ' only in terms of the form $t(y \sqcap t'(x_i))$, where t and t' are nice terms, and in $\Omega_{\mathbf{S}}$-function terms $t(f_\varphi(y))$. Furthermore, we can arrange that $\Omega_{\mathbf{S}}$-function terms in y are the only $\Omega_{\mathbf{S}}$-function terms in φ'.*

Proof of Claim 5. As mentioned before, we can assume w.l.o.g. that φ only contains terms of the form $t_1(v_1) \sqcap t_2(v_2)$, where t_1, t_2 are nice and v_1, v_2 are from $y, \vec{x}$. We first show, that every atomic formula $\psi(t(y) \sqcap t'(x_i), t''(y, \vec{x}))$ can be replaced by an equivalent formula

$$\psi'(y, \vec{x}) = \bigvee_j \psi_j(y, x_i) \wedge \psi(t_j(y \sqcap t'_j(x_i)), t''(y, \vec{x})),$$

where the ψ_j are quantifier-free FO($\mathbf{S}$) formulae and the t_j, t'_j are nice terms.

Let $t(y)$ be $y - a + b$ and $t'(x_i)$ be $x_i - a' + b'$. To prove the above statement we consider three cases.

Case 1 $b \preceq b'$. Then $y - a + b \sqcap x_i - a' + b'$ is b if $a \not\preceq y$ and $(y \sqcap (x_i - a' + (b' - b) + a)) - a + b$, otherwise.

Case 2 $b' \prec b$. Then $y - a + b \sqcap x_i - a' + b'$ is $(y - a + (b - b')) \sqcap x_i - a') + b'$. There are two subcases. Either $(b - b') \preceq (x_i - a')$ and then $y - a + b \sqcap x_i - a' + b'$ is $((y - a) \sqcap (x_i - a' - (b - b'))) + b$ and we proceed as in case 1. Otherwise $b \not\preceq x_i - a' + b'$ and therefore $y - a + b \sqcap x_i - a' + b'$ is $(b \sqcap (x_i - a' + b'))$.

Case 3 b and b' are incomparable. Then $y - a + b \sqcap x_i - a' + b'$ is just $b \sqcap b'$.

Next, we consider formulae of the form $\psi(t(y) \sqcap t'(y), t''(y, \vec{x}))$. In a completely analogous way, we can replace ψ by a formula ψ' of the form $\psi'(y, \vec{x}) = \bigvee_j \psi_j(y, x_i) \wedge \psi(t_j(y \sqcap t'_j(y)), t''(y, \vec{x}))$. By Claim 3, for each j, there is a functional FO($\mathbf{S}$) formula $\theta_j(y, x)$ such that $\theta_j(s, s \sqcap t'_j(s))$ holds and such that, for all r, r' in M, $\theta_j(r, r')$ holds only if $r' = r \sqcap t'_j(r)$.

Hence, each subformula $\psi(t_j(y \sqcap t'_j(y)), t''(y, \vec{x}))$ can be replaced by $\psi(t_j(f_{\theta_j}(y)), t''(y, \vec{x}))$.

The same reasoning can of course be used to transform formulae $\psi(t''(y,\vec{x}), t(y) \sqcap t'(x_i))$ and $\psi(t''(y,\vec{x}), t(y) \sqcap t'(y))$. □

Now we return to the proof of Lemma 3.13. Assume $(c',\vec{c}) \equiv_1 (d',\vec{d})$. Recall that by Theorem 3.5, if two strings satisfy exactly the same atomic formulae of $\Omega_{\mathbf{S}^+}$, then they agree on all FO($\mathbf{S}^+$) formulae.

By Claim 5 it is enough to prove that if $(c',\vec{c}) \equiv_1 (d',\vec{d})$ then $(c',\vec{c})$ and $(d',\vec{d})$ agree on all atomic $\Omega_{\mathbf{S}^+_{\text{left}}}$ formulae that have one or two terms of the form $t(y \sqcap t'(x_i))$ or $t(f_\psi(y))$, where t, t' are nice terms.

Let $\varphi(y,\vec{x})$ be an atomic $\mathbf{S}^+_{\text{left}}$ formula with two terms, where at least one of the terms is of the form $t(y \sqcap t'(x_i))$ or $t(f_\psi(y))$. Assume that $\varphi(y,\vec{x})$ holds for $(c',\vec{c})$ (the case where $\varphi(y,\vec{x})$ holds for $(d',\vec{d})$ is completely analogous). Let $t(z) = z - a + b$. We distinguish the following cases.

Case 1. One term of φ is $t(y \sqcap t'(x_i))$ or $t(f_\psi(y))$ and the other does not contain y. Hence φ is of one of the following forms:

- $P_L(t(y \sqcap t'(x_i)), t''(\vec{x}))$
- $P_L(t''(\vec{x}), t(y \sqcap t'(x_i)))$
- $P_L(t(f_\psi(y)), t''(\vec{x}))$
- $P_L(t''(\vec{x}), t(f_\psi(y)))$

It follows from Claim 2 that in all these subcases one can get rid of the t term, e.g., by adding $-b + a$ to the other term. It is important here that, for a nice term t_1, $t_1(x) \in L$ is an FO($\mathbf{S}$) expressible property. Then the claim follows from the assumption $(c',\vec{c}) \equiv_1 (d',\vec{d})$.

Case 2. φ is of the form $P_L(t_1(y \sqcap t_2(x_i)), t_3(y \sqcap t_4(x_j)))$. By Claim 4 there is an FO($\mathbf{S}$) formula θ such that $\theta(c', t_2(c_i), t_4(c_j))$ holds in M and $\theta(r', t_2(r_i), t_4(r_j))$ implies $P_L(t_1(r' \sqcap t_2(r_i)), t_3(r' \sqcap t_4(r_j)))$, for all $(r',\vec{r})$ in M. By our assumption $(c',\vec{c}) \equiv_1 (d',\vec{d})$ it follows that $P_L(t_1(y \sqcap t_2(x_i)), t_3(y \sqcap t_4(x_j)))$ holds also for $(d',\vec{d})$.

Case 3. φ is of the form $P_L(t(y \sqcap t'(x_i)), t''(f_\psi(y)))$ or of the form $P_L(t''(f_\psi(y)) \sqcap t(y, t'(x_i)))$. Again by Claim 2 we can assume that t'' is empty. Recall that by definition of $\Omega_{\mathbf{S}}$-function terms $f_\psi(y) \preceq y$ and therefore $f_\psi(y) \sqcap y = f_\psi(y)$. Hence, by applying Claim 4 (where we take one term as empty and $s = y$) we get a FO($\mathbf{S}$) formula $\theta(y, t'(x_i))$ such that θ holds for $(c',\vec{c})$ and, whenever $\theta(y, t'(x_i))$ holds for $(d',\vec{d})$, then also φ holds for $(d',\vec{d})$. Again the claim follows from our assumption that $(c',\vec{c}) \equiv_1 (d',\vec{d})$.

Case 4. Both terms of φ are of the form $t(f_\psi(y))$. In this case, we also get an equivalent FO($\mathbf{S}$) formula by first applying Claim 2 to get rid of one symbol t and then applying Claim 4.

This concludes the proof of Lemma 3.13. □

Now we come back to the proof of Theorem 3.12. We actually prove the following which is stronger than what is needed for quantifier-elimination.

3.14 Lemma *$\equiv$ has the back-and-forth property in M.*

As mentioned at the beginning of the proof of the theorem, the statement of the theorem follows from the lemma, as each type of the form $atp_{\mathbf{S}^+}(t_1(c_{i_1}), \ldots, t_k(c_{i_k}))$ is also an atomic type of $\mathbf{S}^+_{\text{left}}$.

Let $\vec{c}$ and $\vec{d}$ such that $\vec{c} \equiv \vec{d}$. Our goal is to show, that for each c', there is d' such that $(c', \vec{c}) \equiv (d', \vec{d})$. By Lemma 3.13 it is enough to find d' such that $(c', \vec{c}) \equiv_1 (d', \vec{d})$.

By compactness, it suffices to show that for all finite sequences $t_1, \ldots, t_k$ of terms and all sequences $i_1, \ldots, i_k$ there is a d' such that

$$atp_{\mathbf{S}^+}(c', t_1(c_{i_1}), \ldots, t_k(c_{i_k})) = atp_{\mathbf{S}^+}(d', t_1(d_{i_1}), \ldots, t_k(d_{i_k})).$$

Let therefore such sequences and c' be fixed. Let T be $\text{Tree}(\{t_j(c_{i_j}) \mid j \leq k\})$. Let T' be the corresponding tree for $\vec{d}$. Let $w = \text{Meet}(c', T)$, $N = \text{Meet}_+(c', T)$ and $P = \text{Meet}_-(c', T)$.

Note that both of these last two strings are given by meets of terms in $+$ and $-$ over $\vec{c}$. Let N' be the image of N in the other model (i.e. the corresponding term in $\vec{d}$), and P' be the image of P. Notice that the inductive hypothesis $\vec{c} \equiv \vec{d}$ guarantees that the ordering relation between meets of these terms in T is preserved when we look at the image terms over $\vec{d}$ and T'. The inductive hypothesis also tells us that (N, P) and (N', P') are equivalent as string models (that is, models in the usual string signature plus an extra predicate for the shorter string); this is because these terms satisfy all the same atomic formulae of $\mathbf{S}^+$, which include all P_Ls.

Now let w' be between N' and P' such that the pairs (N, w) and (N', w'), and (w, P) and (w', P'), are elementary equivalent as string models. Such a string w' exists because quantifier elimination over $\mathbf{S}^+$ (Theorem 3.5) implies that (M, N, P) and (M, N', P') are elementary equivalent in the language of $\mathbf{S}$, and hence for any w there is w' such that the equivalence extends to (M, N, P, w) and (M, N, P, w'). It is clear that such a w' suffices.

Now, let $d' = w' \cdot (c' - w)$. We obviously have that (w, c') and (w', d') are elementary equivalent as string models. We can now check that d' is what we want. We have to show that $\text{Meet}(d', T')$, $\text{Meet}_-(d', T')$ and $\text{Meet}_+(d', T')$ are w', P' and N' respectively, and that for every star-free language L we have: $P_L(w', d')$ iff $P_L(w, c')$, $P_L(P', w')$ iff $P_L(P, w)$, and $P_L(w', N')$ iff $P_L(w, N)$. All of these easily follow from the definition of d'.

This finishes the proof of Lemma 3.14 and thus of Theorem 3.12. □

From the previous theorem we get the following corollaries. First, the back-and-forth property of $\equiv_1$ gives us the following normal form for $\text{FO}(\mathbf{S}^+_{\text{left}})$ formulae.

3.15 Corollary *For every* $\text{FO}(\mathbf{S}_{\text{left}})$ *formula* $\rho(x, \vec{y})$ *there is an* $\text{FO}(\mathbf{S})$ *formula* $\rho'(x, \vec{z})$ *and a finite set of nice* $\mathbf{S}^+_{\text{left}}$ *terms* $\vec{t}$ *such that*

$$\forall x \vec{y}\ (\rho(x, \vec{y}) \leftrightarrow \rho'(x, \vec{t}(\vec{y})))$$

holds in $\mathbf{S}_{\text{left}}$.

Then Corollary 3.15 for the empty tuple $\vec{y}$ and Corollary 3.7 imply:

3.16 Corollary *Subsets of* Σ^* *definable over* $\mathbf{S}_{\text{left}}$ *are precisely the star-free languages.*

For formulae in the language of $\mathbf{S}_{\text{left}}$ (as opposed to $\mathbf{S}^+_{\text{left}}$), we can show that bounded quantification suffices, although the notion of bounded quantification is slightly different here from that used in the previous section. Let $N_p(s)$ be the prefix-closure of $\{s - s_1 + s_2 \mid |s_1|, |s_2| \leq p\}$. Clearly $N_p(s)$ is definable from s over $\mathbf{S}_{\text{left}}$. We then define $\text{FO}_*(\mathbf{S}_{\text{left}})$ as the class of $\text{FO}(\mathbf{S}_{\text{left}})$ formulae $\varphi(\vec{x})$ in which all quantification is of the form $\exists z \in N_p(x_i)$ and $\forall z \in N_p(x_i)$, where x_i is a free variable of φ and $p \geq 0$ arbitrary.

3.17 Corollary $\text{FO}_*(\mathbf{S}_{\text{left}}) = \text{FO}(\mathbf{S}_{\text{left}})$.

Isolation and VC-dimension We now show that the results about isolation and VC-dimension extend from $\mathbf{S}$ to $\mathbf{S}_{\text{left}}$.

3.18 Proposition $\text{Th}(\mathbf{S}_{\text{left}})$ *has the isolation property.*

Proof. Let M be a model of $\text{Th}(\mathbf{S}_{\text{left}})$, W be a pseudo-finite set of elements of M, and $a \in M$. Let $p = tp_M(a/W)$. We exhibit a countable subset W_0 of W such that $tp_M(a/W_0)$ isolates $tp_M(a/W)$.

Let $\vec{e}, \vec{f}$ be finite tuples of finite strings, and let $W(\vec{e}, \vec{f}) = \{w - e + f \mid w \in W, e \in \vec{e}, f \in \vec{f}\}$. Let $w_1(\vec{e}, \vec{f}), w_2(\vec{e}, \vec{f}), w_3(\vec{e}, \vec{f}), w_4(\vec{e}, \vec{f})$ be elements of W such that for some e_1, e_2 in $\vec{e}$ and some $f_1, f_2 \in \vec{f}$,

$$(w_1(\vec{e}, \vec{f}) - e_1 + f_1) \sqcap (w_2(\vec{e}, \vec{f}) - e_2 + f_2) = \text{Meet}_-(a, W(\vec{e}, \vec{f}))$$

and likewise for some e_3, e_4, f_3, f_4 in $\vec{e}, \vec{f}$

$$(w_3(\vec{e}, \vec{f}) - e_3 + f_3) \sqcap (w_4(\vec{e}, \vec{f}) - e_4 + f_4) = \text{Meet}_+(a, W(\vec{e}, \vec{f})).$$

Take

$$W_0 = \bigcup \{w_1(\vec{e}, \vec{f}), w_2(\vec{e}, \vec{f}), w_3(\vec{e}, \vec{f}), w_4(\vec{e}, \vec{f})\},$$

where the union is taken over all finite tuples of finite strings. Clearly W_0 is countable. We claim that $tp_M(a/W_0)$ isolates $tp_M(a/W)$.

Suppose we have a' with $tp_M(a'/W_0) = tp_M(a/W_0)$. Note that by construction of W_0 and definition of $tp_M(a/W_0)$ this implies that a' has the same Meet_- and Meet_+ over each $W(\vec{e}, \vec{f})$ that a does. This also implies that the type of $a' - \text{Meet}(a', W(\vec{e}, \vec{f}))$ is the same as for a, and similarly for the type of $\text{Meet}_+(a', W(\vec{e}, \vec{f})) - \text{Meet}(a', W(\vec{e}, \vec{f}))$ and the type of $\text{Meet}(a', W(\vec{e}, \vec{f})) - \text{Meet}_-(a', W(\vec{e}, \vec{f}))$.

We want to show that $tp_M(a'/W) = tp_M(a/W)$. By quantifier elimination (Theorem 3.12) over $\mathbf{S}_{\text{left}}$, it suffices to show that they have the same atomic types over $\mathbf{S}^+_{\text{left}}$.

From the remark above that a and a' have the same meets and the same paths between those meets and $\text{Meet}_+, \text{Meet}_-$ and themselves it follows that whenever an atom of the form $P_L(t_1 \sqcap t_2, t_3 \sqcap t_4)$ holds for a, where the t_i are either a or nice terms over $\vec{w}$ and where $t_1 \sqcap t_2$ is a direct predecessor of $t_3 \sqcap t_4$ in the tree defined by W, then it also holds for a'. By the normal form for $\mathbf{S}^+$ queries (Proposition 3.4) we can conclude $atp_{\mathbf{S}^+}(a, \vec{w} - \vec{e} + \vec{f}) = atp_{\mathbf{S}^+}(a', \vec{w} - \vec{e} + \vec{f})$, for all finite $\vec{e}, \vec{f}$. Hence, by Claim 3.13 we get that $tp_M(a'/W) = tp_M(a/W)$ have the same atomic types over $\mathbf{S}^+_{\text{left}}$, as required. □

By Lemma 3.9, we obtain the following.

3.19 Corollary *Every definable family in* $\mathbf{S}_{\text{left}}$ *has finite VC-dimension.*

3.4 A regular algebra extending S

The previous sections presented star-free algebras with attractive properties. We now give an example of a regular algebra that has significantly *less* expressive power than the rich structure $\mathbf{S}_{\text{len}}$, and which shares some of the nice properties (isolation, finite VC, QE) of the star-free algebras in the previous sections.

This algebra can be obtained by considering two possible ways of extending FO($\mathbf{S}$): the first is by adding the predicates P_L for all *regular* languages L; that is, predicates $P_L(x, y)$ which hold for $x \preceq y$ such that $y - x \in L$, where L is a regular language. The second extension is by using monadic-second order logic instead of only first-order logic. It turns out that these extensions define exactly the same algebra. We show this, and also show that the resulting regular algebra shares the QE and VC-dimension properties of the star-free algebras defined previously.

Let $\mathbf{S}_{\text{reg}} = \langle \Sigma^*, \preceq, (l_a)_{a\in\Sigma}, (P_L)_{L \text{ regular}} \rangle$. Since it defines arbitrary regular languages in Σ^*, it is a proper extension of $\mathbf{S}$. Every FO($\mathbf{S}_{\text{reg}}$)-definable set is definable over $\mathbf{S}_{\text{len}}$, because the predicates P_L are definable in $\mathbf{S}_{\text{len}}$ (the easiest way to see this is by using the characterization of $\mathbf{S}_{\text{len}}$ definable properties via letter-to-letter automata). Thus, we have:

3.20 Proposition *Subsets of Σ^* definable over $\mathbf{S}_{\text{reg}}$ are precisely the regular languages.*

Let $\mathbf{S}^+_{\text{reg}}$ be the extension of $\mathbf{S}_{\text{reg}}$ with ϵ and $\sqcap$. Most of the results about $\mathbf{S}$ and $\mathbf{S}^+$ from Section 3.2 can be straightforwardly lifted to $\mathbf{S}_{\text{reg}}$ and $\mathbf{S}^+_{\text{reg}}$. For example, the normal form Proposition 3.4 holds for $\mathbf{S}_{\text{reg}}$ if one replaces "star-free" with "regular": the proof given in Section 3.2 applies verbatim. In fact, similar normal form arguments, in a slightly different form, were given in [52, 66]. We now obtain:

3.21 Theorem (see [52]) $\mathbf{S}^+_{\text{reg}}$ *admits quantifier elimination.*

The normal form result also shows that neither the functions f_a nor the predicate el are definable in $\mathbf{S}_{\text{reg}}$ (the latter can also be seen from the fact that $\mathbf{S}_{\text{reg}}$ has QE in a relational signature of bounded arity, and $\mathbf{S}_{\text{len}}$ does not; for inexpressibility of f_a it suffices to apply the normal form results to pairs of strings of the form $(1 \cdot 0^k, 0^k)$: since $1 \cdot 0^k \sqcap 0^k = \epsilon$, it is impossible to check if two sequences of zeros have the same length). One can also show, as in the case of $\mathbf{S}$, that bounded quantification over prefixes is sufficient.

Furthermore, there is a close connection between FO-definability over $\mathbf{S}_{\text{reg}}$ and MSO-definability over $\mathbf{S}$. It was shown in [52] that

$$\text{MSO}(\mathbf{S}) \;=\; \text{FO}(\mathbf{S}_{\text{reg}}).$$

This result was used in [52] to show that S2S and WS2S define the same relations over the infinite binary tree. Here S2S refers to the monadic second-order theory of the infinite binary tree, and WS2S to the weak monadic theory (that is, monadic second-order quantification is restricted to finite sets). Note that it follows from [58] that sets, rather than arbitrary relations, definable in S2S and WS2S, are the same.

From the result of [52] it thus follows that the subsets of Σ^* definable in MSO over $\mathbf{S}_{\text{reg}}$ are precisely the regular languages.

3.4.1 Automata model, isolation, and VC dimension

It was proved in [4] that Regular Prefix Relations (RPR) (those definable by Regular Prefix Automata (RPA), introduced in Section 3.2) are exactly those definable in MSO($\mathbf{S}$). Thus, the results of [4] and [52] give a characterization of FO($\mathbf{S}_{\text{reg}}$).

3.22 Corollary *The relations definable in* FO($\mathbf{S}_{\text{reg}}$) *are exactly the RPR relations. Thus each relation definable in* FO($\mathbf{S}_{\text{reg}}$) *is recognizable by a RPA.*

The proof of the isolation property for $\mathbf{S}$ (Proposition 3.10) is unaffected by the change from star-free P_L to regular P_L. Thus, we obtain:

3.23 Corollary Th($\mathbf{S}_{\text{reg}}$) *has the isolation property, and definable families of* $\mathbf{S}_{\text{reg}}$ *have finite VC-dimension.*

3.5 A regular algebra extending $\mathbf{S}_{\text{left}}$

We now give a final example of a regular algebra. Let $\mathbf{S}_{\text{reg,left}}$ be the common expansion of $\mathbf{S}_{\text{left}}$ and $\mathbf{S}_{\text{reg}}$, that is, $\langle \Sigma^*, \preceq, (l_a)_{a\in\Sigma}, (f_a)_{a\in\Sigma}, (P_L)_{L \text{ regular}}\rangle$. Since $\mathbf{S}_{\text{reg}}$ cannot express the functions f_a, and $\mathbf{S}_{\text{left}}$ cannot define arbitrary regular sets, we see that $\mathbf{S}_{\text{reg,left}}$ is a proper expansion of $\mathbf{S}_{\text{reg}}$ and $\mathbf{S}_{\text{left}}$. Furthermore, all $\mathbf{S}_{\text{reg,left}}$-definable sets are $\mathbf{S}_{\text{len}}$-definable; the finiteness of VC dimension for $\mathbf{S}_{\text{reg,left}}$, shown below, implies that this containment is proper, too.

Let $\mathbf{S}^+_{\text{reg,left}}$ be the common expansion of $\mathbf{S}^+_{\text{left}}$ and $\mathbf{S}_{\text{reg}}$, that is, the expansion of $\mathbf{S}_{\text{reg,left}}$ with ϵ and $\sqcap$. The techniques of the previous sections can be used to show the following:

3.24 Theorem $\mathbf{S}^+_{\text{reg,left}}$ *has quantifier-elimination. Furthermore,* Th($\mathbf{S}_{\text{reg,left}}$) *has the isolation property, and definable families in* $\mathbf{S}_{\text{reg,left}}$ *have finite VC-dimension.*

Proof. We sketch the proof of QE. This is done by simply mimicking the proof of Theorem 3.12, but with the role of $\mathbf{S}$ played now by $\mathbf{S}_{\text{reg}}$. Once again, we work in a saturated model M, and define the equivalence relations $\equiv$ and $\equiv_1$ as in the proof of Theorem 3.12, but the atomic type is with respect to $\mathbf{S}^+_{\text{reg}}$. We then show that $\equiv_1$ and $\equiv$ are the same. This is done by proving the following modification of Claims 2, 3, 4, and 5, by substituting uniformly $\mathbf{S}^+_{\text{reg,left}}$ for $\mathbf{S}^+_{\text{left}}$, and $\mathbf{S}_{\text{reg}}$ for $\mathbf{S}$. The property of star-free languages used in each these claims is just that if L is star-free, and a and b are strings, then the set of x such that $x - a + b \in L$ is also star-free. This clearly holds with regular substituted uniformly for star-free.

We then show that $\equiv$ has the back-and-forth property in M, which implies QE. The proof is the same as before, but instead of elementary equivalence of string models in first-order logic, we consider their elementary equivalence in monadic second-order logic. □

Similarly to $\mathbf{S}_{\text{left}}$, we derive from the proof of Theorem 3.24 the following normal form for $\mathbf{S}_{\text{reg,left}}$ formulae:

3.25 Corollary *For every* FO($\mathbf{S}_{\text{reg,left}}$) *formula* $\rho(x, \vec{y})$ *there is an* FO($\mathbf{S}_{\text{reg}}$) *formula* $\rho'(x, \vec{z})$ *and a finite set of nice* $\mathbf{S}^+_{\text{left}}$ *terms* $\vec{t}$ *such that*

$$\forall x \vec{y}\ \rho(x, \vec{y}) \leftrightarrow \rho'(x, \vec{t}(\vec{y}))$$

holds in $\mathbf{S}_{\mathrm{reg,left}}$.

As we have seen earlier that $\mathrm{MSO}(\mathbf{S}) = \mathrm{FO}(\mathbf{S}_{\mathrm{reg}})$, one might ask if a similar result holds when insertion on the left is allowed; that is, whether $\mathrm{MSO}(\mathbf{S}_{\mathrm{left}}) = \mathrm{FO}(\mathbf{S}_{\mathrm{reg,left}})$. Since the MSO-theory of $\mathbf{S}_{\mathrm{left}}$ is undecidable [67], there is certainly no effective translation. And in fact one can easily see that the two are different. Since the function $g : x \mapsto 0 \cdot x \cdot 1$ is FO-definable in $\mathbf{S}_{\mathrm{left}}$, one can easily see that even weak $\mathrm{MSO}(\mathbf{S}_{\mathrm{left}})$, where set quantification is restricted to finite sets, defines $\{0^n 1^n \mid n \geq 0\}$, a non-regular set.

We conclude this section with a remark showing that arithmetic properties definable in structures $\mathbf{S}, \mathbf{S}_{\mathrm{left}}, \mathbf{S}_{\mathrm{reg}}, \mathbf{S}_{\mathrm{reg,left}}$ are weaker than those definable in $\mathbf{S}_{\mathrm{len}}$. As we mentioned earlier, under the binary encoding, $\mathbf{S}_{\mathrm{len}}$ gives us an extension of Presburger arithmetic; namely, it defines $+$ and V_2, where $V_2(x)$ is the largest power of 2 that divides x. But even $\mathbf{S}_{\mathrm{reg,left}}$ is much weaker:

3.26 Proposition *Neither successor, nor order, nor addition, are definable in* $\mathbf{S}_{\mathrm{reg,left}}$ *(and hence in* $\mathbf{S}, \mathbf{S}_{\mathrm{reg}}, \mathbf{S}_{\mathrm{left}}$*).*

Proof. Since order is definable from addition, and successor from order, it suffices to show that successor is not definable. Let $x_k = 10^k, y_k = 1^k$; that is, under the binary encoding, x is the successor of y. We show that $\{(x_k, y_k) \mid k > 0\}$ is not definable in $\mathbf{S}_{\mathrm{reg,left}}$.

Assume it were; by Corollary 3.25 we get a set of nice terms $t_i(y) = y - a_i + b_i$ and a formula $\alpha(x, \vec{z})$ over $\mathbf{S}_{\mathrm{reg}}$ such that $\alpha(x, \vec{t}(y))$ is true iff for some k, $x = x_k$ and $y = y_k$. For sufficiently large k, $\vec{t}(y_k)$ consists of strings of the form $c_i \cdot 1^{k-p_i}$ where c_i and p_i depend on $\vec{t}$ only. As $c_i \cdot 1^{k-p_i} \cdot 1^{p_i}$ is $c_i \cdot y_k$, there is a formula $\beta(x, z_1, \ldots, z_l)$ of $\mathbf{S}_{\mathrm{reg}}$ (where l is the length of $\vec{t}$) such that $\beta(x, \vec{z})$ is true iff for some big enough k, $x = x_k$ and $z_i = c_i \cdot y_k$.

We now show that for sufficiently large k, depending on β, if $\beta(x_k, c_1 y_k, \ldots, c_l y_k)$ is true, then for some $m > k$, $\beta(x_m, c_1 y_k, \ldots, c_l y_k)$ is true. Clearly this will suffice. For this we use the normal form for $\mathbf{S}_{\mathrm{reg}}$ which is analogous to Proposition 3.4 except that L in P_L could be regular. Note that for sufficiently large k_0, and any $k, m \geq k_0$, $\mathrm{Tree}(x_k, \vec{c} \cdot y_k)$ is isomorphic (as a tree) to $\mathrm{Tree}(x_m, \vec{c} \cdot y_k)$. In particular, the predecessor of x_k (and x_m) in such a tree is its meet with one of $c_i \cdot y_k$, say $c_1 \cdot y_k$. Such a meet is 1 if $c_1 = \epsilon$, or a prefix of c_1 if $c_1 \neq \epsilon$. Thus, $x_k - (x_k \sqcap c_1 y_k)$ is either x_k or a string 0^p for $p \geq k - |c_1|$, with p depending only on c_1. (The same is true when one replaces k by m).

Let P_L be the formula describing the segment $(x \sqcap c_1 y, x)$ in the normal form for β (we may assume w.l.o.g. that there is only one such formula; if there are several, one can combine them into one by taking the intersection of the languages). Pick $k_1, k_2 > k_0$ such that $x_{k_1} - (c_1 y_{k_1} \sqcap x_{k_1})$ is in L iff $x_{k_2} - (c_1 y_{k_1} \sqcap x_{k_2})$ is. It follows from the description of those meets given above that such k_1, k_2 always exist. Now it is immediate from the normal form result that $\beta(x_{k_1}, c_1 y_{k_1}, \ldots, c_l y_{k_1})$ iff $\beta(x_{k_2}, c_1 y_{k_1}, \ldots, c_l y_{k_1})$, which finishes the proof. □

Figure 1 and Table 1 summarize the results of this section.

4 String query languages

The goal of this section is to study relational calculi based on the five structures considered in the previous section. Note, however, that most of the previous research on string query languages used

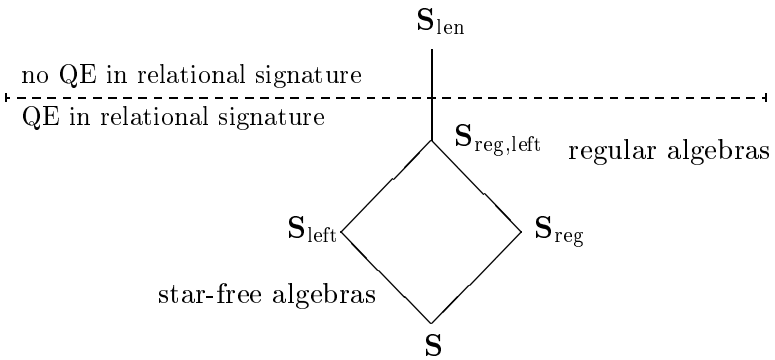

Figure 1: *Relationships between* $\mathbf{S}, \mathbf{S}_{\text{left}}, \mathbf{S}_{\text{reg}}, \mathbf{S}_{\text{reg,left}}$, *and* $\mathbf{S}_{\text{len}}$.

Structure	Signature	Expansion with quantifier-elimination	Expansion name
$\mathbf{S}_{\text{len}}$	$\prec, (l_a)_{a\in\Sigma}, \text{el}$	all unary relations & binary functions	$\mathbf{S}_{\text{len}}^{(1,2)}$
$\mathbf{S}$	$\prec, (l_a)_{a\in\Sigma}$	$\prec, (l_a)_{a\in\Sigma}, \epsilon, \sqcap,$ $(P_L)_{L \text{ star-free}}$	$\mathbf{S}^+$
$\mathbf{S}_{\text{left}}$	$\prec, (l_a)_{a\in\Sigma}, (f_a)_{a\in\Sigma}$	$\prec, (l_a)_{a\in\Sigma}, (f_a)_{a\in\Sigma}, \epsilon,$ $(x-a)_{a\in\Sigma}, \sqcap, (P_L)_{L \text{ star-free}}$	$\mathbf{S}_{\text{left}}^+$
$\mathbf{S}_{\text{reg}}$	$\prec, (l_a)_{a\in\Sigma}, (P_L)_{L \text{ regular}}$	$\prec, (l_a)_{a\in\Sigma}, \epsilon, \sqcap, (P_L)_{L \text{ regular}}$	$\mathbf{S}_{\text{reg}}^+$
$\mathbf{S}_{\text{reg,left}}$	$\prec, (l_a)_{a\in\Sigma}, (f_a)_{a\in\Sigma}, (P_L)_{L \text{ regular}}$	$\prec, (l_a)_{a\in\Sigma}, (f_a)_{a\in\Sigma}, \epsilon,$ $(x-a)_{a\in\Sigma}, \sqcap, (P_L)_{L \text{ regular}}$	$\mathbf{S}_{\text{reg,left}}^+$

Definition of P_L: $\quad (x, y) \in P_L$ iff $x \prec y$ and $y - x \in L$.

Table 1: Summary of quantifier-elimination results

concatenation as the main string operation. We give a few simple results indicating that our main goals of getting a low complexity language with an adequate notion of relational algebra cannot be achieved if we include concatenation as a primitive. After that, we explain how operations used in $\mathbf{S}, \mathbf{S}_{\text{left}}, \mathbf{S}_{\text{reg}}, \mathbf{S}_{\text{reg,left}}, \mathbf{S}_{\text{len}}$ are related to SQL string operations, and present properties of relational calculi based on these structures. Most of these are based on model-theoretic properties of the five structures established in Section 3.

4.1 Problematic concatenation

Most earlier papers considered relational calculus with concatenation RC_{concat}, that is, $\text{RC}(SC, \langle \Sigma^*, \Omega \rangle)$ where Ω has the operation of concatenation, and constant symbols for each $a \in \Sigma$. This language is extremely attractive in terms of compositionality: given queries Q and Q' returning sets of strings, one can substitute Q and Q' within regular-expressions to form new `LIKE` queries. However, as noticed in [40], for $\Sigma = \{0, 1, \sharp\}$, RC_{concat} expresses all computable queries on databases containing strings from $\{0, 1\}^*$ (see [61] for a proof). In fact, it is easy to show a somewhat stronger result which only requires two letters in Σ.

4.1 Proposition *Let Σ contain at least two letters. Then RC_{concat} expresses all computable queries on databases over Σ^*.*

Proof. We first show that all computable predicates on $\{0, 1\}^*$ are expressible. We follow the lines of [61], Chapter III, Theorem 12.4, which uses an extra symbol $\sharp$ to encode a Turing machine computation in RC_{concat}. Let M be a Turing machine. Let $Q = \{q_2, \cdots, q_m\}$ be the set of states of M, q_2 being the initial state. At step i of the execution of M over an input x, the configuration of M can be represented by a string $u_i \sharp^{\alpha_i} v_i$, $u_i, v_i \in \{0, 1\}^*$, where u_i is the tape content left of the head, v_i is the content of the current position and the positions right of the current position, and q_{α_i} is the current state. Let $\varphi_M(x)$ be the formula of RC_{concat} which states the existence of a string $w \in \{0, 1, \sharp\}^*$ which will represent the computation of M on x. This is done as follows:

1. $w = \sharp^{\alpha_0} v_0 \sharp u_1 \sharp^{\alpha_1} v_1 \sharp \cdots \sharp u_n \sharp^{\alpha_n} v_n$, for some n, where $u_i, v_i \in \{0, 1\}^*$.
2. $v_0 = x$, $\alpha_0 = 2$.
3. if $u_i \sharp^{\alpha} v_i \sharp u_{i+1} \sharp^{\beta} v_{i+1}$ is a substring of w then $u_{i+1} \sharp^{\beta} v_{i+1}$ represents the configuration after executing M, for one step, from the configuration represented by $u_i \sharp^{\alpha} v_i$.
4. q_{α_n} is an accepting state of M.

All the points enumerated above can be checked in RC_{concat} [61]. It is also easy to see that the existence of such a string w is equivalent to the acceptance of x by M.

In order to remove the extra symbol $\sharp$, the formula $\varphi_M(x)$ also states the existence of a string $x_\sharp$ of the form 10^k1, such that none of the strings u_i, v_i contains 0^k as a substring. As the computation is finite, such a string always exists and it can easily be distinguished from the u_i and v_i. The formula then states the existence of a string w' of the form $x_\sharp^{\alpha_0} v_0 x_\sharp u_1 x_\sharp^{\alpha_1} v_1 x_\sharp \cdots x_\sharp u_n x_\sharp^{\alpha_n} v_n$ and condition 3 is changed analogously.

Since all computable predicates on $\{0,1\}^*$ are expressible, there is a one-to-one mapping $f : \mathbb{N} \to \{0,1\}^*$ such that the image of addition and multiplication under f is expressible in FO over $\langle \Sigma^*, \cdot, 0, 1\rangle$. It is known (see [50], Chapter 3), that relational calculus over $\langle \mathbb{N}, +, \cdot\rangle$ expresses all computable queries over finite databases (simply by coding finite databases with numbers). Hence, the same coding will apply to RC_{concat}, showing that it expresses all computable queries. □

In databases, we are accustomed to relational calculus having limited expressiveness; then the queries can be analyzed and often good optimizations can be discovered. This is certainly not the case here; moreover, there is no hope of finding a syntax for safe queries.

4.2 Corollary *Let Σ contain at least two letters. Then there is no effective syntax for safe queries in RC_{concat}. Furthermore, the state-safety problem is undecidable for RC_{concat}.*

Proof. This follows from [64]. Indeed from Proposition 4.1, RC_{concat} is Turing-complete and thus the structure of [64] in which there is no safe syntax for safe queries, and in which state-safety is undecidable is definable. □

Note that when Σ has one symbol, $\langle \Sigma^*, \cdot\rangle$ is essentially $\langle \mathbb{N}, +\rangle$, and there exists effective syntax for safe queries, and state-safety is decidable [64].

4.2 Basic string operations in SQL

When looking at existing SQL string operations, the most often-used operation is `LIKE` pattern-matching. It allows one to say, for example, that a given string is a prefix of another string and also that a string has a fixed string as a substring. `LIKE` patterns are built from alphabet letters, and characters `%` (which matches any string, including ϵ), and `_` (which matches a single letter). For example, the pattern `ab_c%` matches any string whose first letter is a, second is b, and fourth is c. Matching with `LIKE` can be expressed in first-order logic over $\mathbf{S}$: indeed, with `LIKE` one can only define star-free languages, which are FO-definable in $\mathbf{S}$.

Another important SQL string operation is the lexicographic ordering $\leq_{\mathrm{lex}}$, which, as we saw earlier, is also expressible in $\mathbf{S}$.

SQL also allows trimming/adding symbols on both left and right of a string. We know that trimming/adding symbols on the right (operation l_a and its inverse) is expressible over $\mathbf{S}$, but adding/trimming on the left (operation f_a and its inverse) is not. This motivated the study of the structure $\mathbf{S}_{\mathrm{left}}$; it corresponds to `LIKE` pattern matching, lexicographic ordering, and arbitrary trimming/adding operators of SQL.

The operator `LIKE` checks membership in a star-free language. The new SQL standard [41] introduces an arbitrary regular expression pattern-matching by a new operator called `SIMILAR`. Adding this operator corresponds to going from $\mathbf{S}$ to $\mathbf{S}_{\mathrm{reg}}$ or $\mathbf{S}_{\mathrm{left}}$ to $\mathbf{S}_{\mathrm{reg,left}}$: in both cases, the addition means that the one-dimensional definable families become regular instead of star-free.

Finally, SQL has a string-length operation called `LEN`. Since this does not return a string, we turn it into a pure string operation that compares lengths of strings: $\mathrm{el}(x, y)$ is true if $|x| = |y|$. Thus, $\mathbf{S}_{\mathrm{len}}$ corresponds to a set of SQL operations that includes `LIKE`, lexicographic ordering and length comparison. Furthermore, since $\mathbf{S}_{\mathrm{len}}$ subsumes $\mathbf{S}_{\mathrm{left}}, \mathbf{S}_{\mathrm{reg}}$ and $\mathbf{S}_{\mathrm{reg,left}}$, the operator `SIMILAR` and trimming/adding on the left are expressible over $\mathbf{S}_{\mathrm{len}}$.

4.3 Expressive power and complexity

In this section we study expressiveness and complexity of the five relational calculi. We obtain a number of collapse results using the isolation property shown in the first part of the paper, and establish complexity bounds, both in the cases with and without collapse.

4.3.1 Relational calculus over S

Our goal here is to get bounds on the expressiveness and data complexity for queries in RC($\mathbf{S}$). The main tool used is a collapse result, Theorem 4.3, in the spirit of those produced for constraint databases [10, 8]. Recall that relational calculus over a domain RC($\mathcal{M}$) admits *restricted quantifier collapse* if every RC($SC, \mathcal{M}$) formula $\varphi(\vec{x})$ is equivalent to a formula $\varphi'(\vec{x})$ in which SC-predicates occur only within the scope of active domain quantifiers $\exists x \in adom$ and $\forall x \in adom$. It admits the natural-active collapse if every formula is equivalent to one with only active-domain quantifiers.

We already mentioned that the isolation property implies restricted quantifier collapse [8, 32]. From the QE of $\mathbf{S}^+$ we also get

4.3 Theorem RC($\mathbf{S}$) *admits restricted quantifier collapse, and* RC($\mathbf{S}^+$) *admits the natural-active collapse.*

Another quantifier-restriction result is given in the following corollary. Extend RC($SC, \mathbf{S}$) with quantifiers of the form $\exists x \prec adom$ and $\forall x \prec adom$, whose meaning is as follows. Given a formula $\varphi(x, \vec{y})$, an interpretation $\vec{a}$ for $\vec{y}$, and a database D, $\exists x \prec adom\ \varphi(x, \vec{a})$ states that there exists a string c making $\varphi(c, \vec{a})$ true such that either $c \preceq a_i$ for a_i a component of $\vec{a}$, or $c \preceq b$ where b is in $adom(D)$. Since bounded quantification suffices for $\mathbf{S}$ formulae (Corollary 3.6), we obtain:

4.4 Corollary *Every* RC($SC, \mathbf{S}$) *formula is equivalent to a formula that only uses quantifiers* $\exists x \prec adom$ *and* $\forall x \prec adom$.

We note that a a straightforward corollary of Theorem 4.3 shows that the data complexity for RC($\mathbf{S}$) matches that of pure relational calculus.

4.5 Corollary *The data complexity of* RC($\mathbf{S}$) *is in* AC^0. *In particular, neither parity nor connectivity test is expressible in* RC($\mathbf{S}$).

Proof. By Corollary 4.4 we can assume that a given query $\varphi(\vec{x})$ is of the form $Q\vec{y} \in \mathrm{adom}\ \bigvee \bigwedge \alpha_i(\vec{x}, \vec{y})$ where each α_i is either an atomic or negated atomic SC-formula, or an $\mathbf{S}$ formula, in which all quantification is restricted to prefixes of $\vec{x}, \vec{y}$. The proof then follows the standard proof of AC^0 data complexity for the relational calculus (see, for example, [1]), and one only has to prove that each $\mathbf{S}$ formula can be evaluated in AC^0.

Suppose $\alpha(z_1, \ldots, z_k)$ is an $\mathbf{S}$ formula in which all quantification is restricted to prefixes of z_is. With $\vec{z}$, associate a structure $S_{\vec{z}}$ of the signature consisting of unary predicates Z_i, $(P_a)_{a \in \Sigma}$, $\#$ and a binary predicate $<$ as follows: the domain is $\{1, \ldots, M\}$, where $M = \sum_i |z_i| + (k-1)$, and the interpretation of $<$ is standard. The first $|z_1|$ elements belong to Z_1, followed by an element that belongs to $\#$, followed by $|z_2|$ elements that belong to Z_2 etc. The membership in P_a is determined

by the corresponding symbol in the z_is. To show that α can be evaluated in AC^0, it is enough to show that there is a FO(BIT, <) sentence β such that $\mathbf{S} \models \alpha(\vec{z})$ iff $S_{\vec{z}} \models \beta$. This is done by a straightforward induction on the structure of α, as one can encode the prefix relation over $S_{\vec{z}}$ using the definability of + and × in FO(BIT, <) (cf. [47]). □

Another corollary concerns the expressive power of generic queries. Recall that a query is *generic* if it commutes with permutations on the domain; in other words, it is independent of specific elements stored in a database.

Every query expressible in pure relational calculus is generic. Examples of other generic queries are parity test and graph connectivity test; these are well known to be inexpressible in relational calculus.

Combining Theorem 4.3 with the active generic collapse [10], we obtain:

4.6 Corollary *Every generic query expressible in* $\mathrm{RC}(\mathbf{S})$ *is already expressible in* $\mathrm{RC}(<)$*, relational calculus over ordered databases.*

With respect to time complexity Corollary 4.5 only gives a polynomial upper bound. We show next that for unary databases we get a much stricter complexity result. We call a database schema SC *unary* if it only contains unary relation names. We next show that queries over unary databases can be evaluated in linear time. This is because a unary database can be transformed into a tree, and a query can be transformed into a first-order sentence over the tree, which can then be evaluated by a tree automaton. More precisely, we have:

4.7 Proposition *Let SC be unary. Then every Boolean* $\mathrm{RC}(SC, \mathbf{S})$*-query can be evaluated in linear time in the size of the database.*

Proof. Let SC be unary. We define a representation of SC-databases by finite labeled trees as follows. Let $R_1, \ldots, R_m$ be the relation names of SC. Let, for simplicity, Σ be $\{0, 1\}$. Let $X = \{x_1, \ldots, x_k\}$ be a set of variables. For a finite database D over SC and a vector $\vec{a} = a_1, \ldots, a_k$ of strings from Σ^* the (finite) tree $t = t(D, \vec{a})$ is defined as follows.

- The set of vertices of t is $\mathit{prefix}(D, \vec{a})$.
- Each vertex v of t is labeled by a 0-1-vector $\vec{r}(v) = (r_1(v), \ldots, r_m(v))$, where $r_i(v) = 1$ if and only if $v \in R_i$.
- Each vertex v is labeled by a subset $X(v)$ of X, where $x_i \in X(v)$ if and only if $a_i = v$.

It should be pointed out that all leaves v of $t(D, \vec{a})$ carry a label $\vec{r}(v)$ with at least one non-zero entry or a label $X(v)$ which is not the empty set.

It is straightforward that, for each $\mathrm{RC}(SC, \mathbf{S})$-formula φ (with prefix quantification) there is a first-order formula φ' on labeled trees (represented as finite structures in the usual way) such that for each SC-database D and each vector $\vec{a} = a_1, \ldots, a_k$ of strings, $(D, \vec{a}) \models \varphi$ if and only if $t(D, \vec{a}) \models \varphi'$. In the case of Boolean queries, k equals 0. As it is well-known that even MSO-sentences can be evaluated in linear time on labeled trees (e.g., via the simulation of suitable tree automata, see [67]), we can conclude the desired complexity bound. □

From Lemma 4.20 below and the results of [56] it follows that safe unary RC($SC, \mathbf{S}$)-queries (i.e., with one free variable) can be evaluated in linear time in the size of the database. By combining this with the techniques of [62] it can be shown that, in general, k-ary queries can be evaluated in time $O(n^k)$ for databases of size n.

4.3.2 Relational calculus over $\mathbf{S}_{\text{len}}$

We have seen that query evaluation for relational calculus over $\mathbf{S}$ has low complexity. However, many useful queries of low complexity, such as the query that appends a fixed string on the left of a given column, are not expressible in $\mathbf{S}$. Hence we examine the addition of the equal length predicate, that is, relational calculus over $\mathbf{S}_{\text{len}}$. Throughout this section, we again assume that the alphabet has at least two symbols (as over the one-symbol alphabet, equal length is simply equality and thus does not give us any extra power).

To analyze the expressive power and complexity of $\mathbf{S}_{\text{len}}$, we again make use of a normal-form result for queries. In this case it is no longer sufficient to quantify over prefixes of strings in the active domain; however a different restricted quantification suffices.

We introduce quantifiers $\exists\ |x| \leq adom$ and $\forall\ |x| \leq adom$ to be interpreted as follows. Given a formula $\varphi(\vec{y})$, a database D and an interpretation $\vec{a}$ for $\vec{y}$, a subformula $\exists\ |x| \leq adom\ \alpha(x, \cdot)$ is satisfied if there exists a string c satisfying $\alpha(c, \cdot)$ such that the length of c does not exceed the length of the longest string in $adom(D)$ and $\vec{a}$. We call these *length-restricted quantifiers*. Note that they are just a notational convenience, as they can be expressed in RC($\mathbf{S}_{\text{len}}$). Moreover, they capture the expressiveness of RC($\mathbf{S}_{\text{len}}$):

4.8 Proposition *Every* RC($SC, \mathbf{S}_{\text{len}}$) *formula is equivalent to a formula that uses only length-restricted quantifiers.*

Proof. For an SC-database D and a tuple of strings $\vec{s}$, we use the notation $\downarrow(D, \vec{s})$ for $\{s' \mid \exists s \in adom(D) \cup \vec{s} : |s'| \leq |s|\}$, and $S[D, \vec{s}]$ for the structure with the universe $\downarrow(D, \vec{s})$ in the language of $\mathbf{S}_{\text{len}}$ plus the SC-relations, plus constants for the elements of $\vec{s}$. We write $\mathbf{S}_{\text{len}}(D, \vec{s})$ for the structure in the same language whose universe is Σ^*. Let m be the maximum arity of any relation name of SC.

We write $(D_1, \vec{s}_1) \equiv_k (D_2, \vec{s}_2)$ if the duplicator has a winning strategy in the k-round Ehrenfeucht-Fraïssé game on $\mathbf{S}_{\text{len}}(D_1, \vec{s}_1)$ and $\mathbf{S}_{\text{len}}(D_2, \vec{s}_2)$, and $(D_1, \vec{s}_1) \equiv^b_k (D_2, \vec{s}_2)$ if the duplicator has a winning strategy in the k-round Ehrenfeucht-Fraïssé game on $S[D_1, \vec{s}_1]$ and $S[D_2, \vec{s}_2]$. We claim that $\equiv^b_{k+m+1}$ refines $\equiv_k$. By the Ehrenfeucht-Fraïssé theorem (cf. [27, 47]), this implies the result, as both equivalence relations are of finite index, each class of $\equiv^b_{k+m+1}$ is definable with length-restricted quantifiers, and each RC($\mathbf{S}_{\text{len}}$) query of quantifier rank k is a union of $\equiv_k$-classes.

We now describe the winning strategy for the duplicator for k moves in the game on $\mathbf{S}_{\text{len}}(D_1, \vec{s}_1)$ and $\mathbf{S}_{\text{len}}(D_2, \vec{s}_2)$. Let l_j be the maximum length of a string in $S[D_j, \vec{s}_j], j = 1, 2$. In response to each move, say $a_i \in \mathbf{S}_{\text{len}}(D_1, \vec{s}_1)$ by the spoiler, the duplicator produces, in addition to his response $b_i \in \mathbf{S}_{\text{len}}(D_2, \vec{s}_2)$, two extra elements $a'_i \in S[D_1, \vec{s}_1]$ and $b'_i \in S[D_2, \vec{s}_2]$. This is done as follows. Suppose the rounds $1, \ldots, i-1$ have already been played, and the spoiler plays $a_i \in \mathbf{S}_{\text{len}}(D_1, \vec{s}_1)$.

There are two cases. If $a_i \in S[D_1, \vec{s}_1]$, then $a'_i = a_i$, and the duplicator looks at the position $(a'_1, \ldots, a'_{i-1}, a'_i), (b'_1, \ldots, b'_{i-1})$ in the game on $S[D_1, \vec{s}_1]$ and $S[D_2, \vec{s}_2]$, and selects $b'_i \in S[D_2, \vec{s}_2]$

according to his winning strategy. He then sets $b_i = b'_i$.

In the other case, we have $a_i \notin S[D_1, \vec{s}_1]$, that is, $|a_i| > l_1$. Let a'_i be the prefix of a_i of length l_1. As before, the duplicator now looks at the configuration $(a'_1, \ldots, a'_{i-1}, a'_i), (b'_1, \ldots, b'_{i-1})$ in the game on $S[D_1, \vec{s}_1]$ and $S[D_2, \vec{s}_2]$, and selects b'_i as the response to a'_i. Note that b'_i is of length l_2. Indeed, since the duplicator can play in the game on $S[D_1, \vec{s}_1]$ and $S[D_2, \vec{s}_2]$ for $k+m+1$ moves, for every move up to k his response to a string of length l_1 must be a string of length l_2, for otherwise with the next $m+1$ moves the spoiler would be able to choose an extension b_{i+1} of b_i and strings $b_{i+2}, \ldots, b_{i+m+1}$ such that b_{i+2} has the same length as b_{i+1} and is in $\vec{s}_2$ or D_2. The latter might be witnessed by the strings $b_{i+3}, \ldots, b_{i+m+1}$. The duplicator would have no suitable response in $S[D_1, \vec{s}_1]$. We now set $b_i = b'_i \cdot x$, where $x = a_i - a'_i$, that is, x is the relative suffix of a'_i in a_i. It follows immediately that this strategy ensures the win by the duplicator in the k-round game on $\mathbf{S}_{\mathrm{len}}(D_1, \vec{s}_1)$ and $\mathbf{S}_{\mathrm{len}}(D_2, \vec{s}_2)$. □

Prefix-restricted quantification does not suffice for $\mathrm{RC}(\mathbf{S}_{\mathrm{len}})$. Indeed, consider the following query Q on a unary relation U: $Q(U)$ is true iff U contains a single element, which is from 0^* and of even length. This is expressible in $\mathrm{RC}(\mathbf{S}_{\mathrm{len}})$ by

$$\exists! x\, U(x) \wedge \forall x(U(x) \to (x \in 0^*) \wedge \exists z \in (01)^* \mathrm{el}(z, x)),$$

where $\exists! x U(x)$ expresses that there is exactly one x with $U(x)$. Note that the predicates $x \in 0^*$ and $z \in (01)^*$ can be expressed even over $\mathbf{S}$: recall that $\mathbf{S}$ can define any star-free language and $\mathbf{S}_{\mathrm{len}}$ any regular language. However, this query Q is inexpressible with just prefix quantification: if it were, then over single-element databases contained in 0^*, the predicate el could be replaced by equality. Hence the set of strings from 0^* of even length would be definable over $\mathbf{S}$. But this language is not star-free, and this contradicts the fact that the languages definable over $\mathbf{S}$ are exactly the star-free languages (Corollary 3.7).

As with Theorem 4.3, from Proposition 4.8 we get us a rough upper bound on the complexity of $\mathrm{RC}(\mathbf{S}_{\mathrm{len}})$, which should be compared with Corollary 4.11 and Proposition 4.12 below:

4.9 Corollary *The data complexity of* $\mathrm{RC}(\mathbf{S}_{\mathrm{len}})$ *is in PH.*

Proof. To check if $D \models \varphi(\vec{a})$, it is enough to quantify over strings whose length does not exceed N, where N is the maximum length of a string in $adom(D) \cup \vec{a}$ (see Proposition 4.8). If φ has alternation depth k this can be done by a polynomial time alternating Turing machine with k alternations, hence in PH. □

The result below establishes two bounds. The first one is for complexity of generic queries in $\mathrm{RC}(\mathbf{S}_{\mathrm{len}})$. That is, the complexity of the language $\{enc(D)\#\mathrm{enc(t)} \mid D \models \varphi(t)\}$ for a generic φ. The other complexity bound is very useful for proving expressibility results. A *relational (Boolean) query* is a set of isomorphism types of SC-databases (w.r.t. the SC-relations only). A relational query is in AC^0 if it is in AC^0 under the usual relational encoding enc_0: elements of a k-element active domain are encoded by $1, \ldots, k$, in binary (cf. [1]). A relational query Q is expressible in $\mathrm{RC}(\mathbf{S}_{\mathrm{len}})$ if there is a $\mathrm{RC}(\mathbf{S}_{\mathrm{len}})$ sentence Φ such that the SC-isomorphism type of D is in Q iff $D \models \Phi$.

4.10 Theorem *The data complexity of generic queries in* $\mathrm{RC}(\mathbf{S}_{\mathrm{len}})$ *is in* TC^0. *Furthermore, any relational query that is expressible in* $\mathrm{RC}(\mathbf{S}_{\mathrm{len}})$ *is in* AC^0.

Proof. Without loss of generality, we consider Boolean queries and assume that $\Sigma = \{0, 1\}$. For a string $s \in \Sigma^*$, let $N(s)$ be the number which is $1 \cdot s$ in binary. Let $s <_N s'$ iff $N(s) < N(s')$. Note that for strings of length k, $N(s)$ ranges from 2^k to $2^{k+1} - 1$, and $|s| < |s'|$ implies $N(s) < N(s')$. We call a database *nice* if the set $\{N(s) \mid s \in adom(D)\}$ is of the form $\{1, \ldots, n\}$ for some $n \geq 1$. Note that the maximum length of a string in such a database is $l(n) = \lceil \log_2(n+1) \rceil - 1$.

Now we claim that every Boolean generic query Φ can be evaluated in AC^0 over nice databases. By Proposition 4.8, without loss of generality, all quantifiers in Φ are assumed to be length-restricted. With a nice database D, we associate a new database D' of the same schema with the universe $\{1, \ldots, n\} = \{N(s) \mid s \in adom(D)\}$, such that $(t_1, \ldots, t_k) \in R$ in D iff $(N(t_1), \ldots, N(t_k)) \in R$ in D'. We next show that Φ can be expressed in $\mathrm{FO}(\mathrm{BIT}, <)$ over structures of the form D', where D is nice. This will suffice to prove the claim, as the encodings of D and D' are identical, and $\mathrm{FO}(\mathrm{BIT}, <)$ captures uniform AC^0 [7]. Recall the definition of BIT from Section 2. We also recall that the usual arithmetic predicates ($+$ and $\times$, given as ternary predicates) are definable in $\mathrm{FO}(\mathrm{BIT}, <)$, and so are many other helpful predicates, for example, a predicate for the powers of 2 [47].

There are two main problems: first, quantification in Φ is restricted to the maximum length of a string (that is, over nice databases, quantifiers in Φ range not over $\{1, \ldots, n\}$ but rather $\{1, \ldots, 2^{l(n)+1} - 1\}$); second, we must show that the operations of $\mathbf{S}_{\mathrm{len}}$ can be expressed.

To deal with the first problem, we assume that Φ is in prenex form, and replace each quantifier $\exists s$ with two quantifiers $\exists i_s \exists i'_s$. Each string s of length not exceeding $l(n)$ can be represented uniquely by two numbers i_s, i'_s such that:

$$i_s = \begin{cases} N(s) & \text{if } N(s) \leq n, \\ n & \text{if } N(s) > n, \end{cases} \qquad i'_s = \begin{cases} 2^{l(n)} & \text{if } N(s) \leq n, \\ N(s) - n & \text{if } N(s) > n. \end{cases}$$

Note that $i_s, i'_s \leq n$, and for $N(s) > n$, $i'_s < 2^{l(n)}$, if $|s| \leq l(n)$. For each new pair of quantifiers $\exists i_s \exists i'_s$ we add a formula stating that i_s, i'_s satisfy the following conditions: either $i_s < n$ and $i'_s = 2^{l(n)}$, or $i_s = n$, and $i'_s < 2^{l(n)}$. This can be done in $\mathrm{FO}(\mathrm{BIT}, <)$, as the condition $x = 2^{l(n)}$ is expressible (it says that x is the largest power of 2 that does not exceed n, which is expressible with BIT).

Next, we must show how to translate the atomic and negated atomic subformulae of Φ. Each subformula of the form $R(s_1, \ldots, s_k)$, where $R \in SC$, is translated into $R(i_{s_1}, \ldots, i_{s_k}) \wedge \bigwedge_i i'_{s_i} = 2^{l(n)}$. Checking $L_0(s)$ is simply $\neg\mathrm{BIT}(i_s, 1)$, and $L_1(s)$ is $\mathrm{BIT}(i_s, 1)$. For $\mathrm{el}(s, u)$, one has to check that the largest power of 2 not exceeding $i_s + i'_s$ and $i_u + i'_u$ is the same. This happens iff either both i'_s, i'_u are less than $2^{l(n)}$ (in this case $|s| = |u| = l(n)$), or both equal $2^{l(n)}$ (in which case both s and u are in the active domain), and for each $p \leq \max(i_s, i_u)$ which is a power of 2, $p\theta i_s \leftrightarrow p\theta i_u$, where θ ranges over the comparisons $<$, $>$ and $=$. These conditions can be expressed in $\mathrm{FO}(\mathrm{BIT}, <)$.

We now consider the predicate $s \prec u$. There are four cases. If both $i'_s, i'_u < 2^{l(n)}$, this is false, as s, u are not in the active domain, and hence of the same length. Similarly if $i'_s < 2^{l(n)}$ and $i'_u = 2^{l(n)}$, then $s \prec u$ is false.

The third case is when $i'_s = i'_u = 2^{l(n)}$. In this case both s and u are in the active domain, and the formula below states that $s \prec u$:

$$\begin{aligned} \exists p, p' \quad & \mathrm{FirstBIT}(i_s, p) \wedge \mathrm{FirstBIT}(i_u, p') \wedge p < p' \wedge \\ & \forall q \leq p' \; \mathrm{BIT}(i_s, p - q) \leftrightarrow \mathrm{BIT}(i_u, p' - q), \end{aligned}$$

where FirstBIT(u, p) is the formula

$$\mathrm{BIT}(u,p) \wedge \forall q\ (p < q \leq l(n)) \rightarrow \neg \mathrm{BIT}(u,q)$$

expressing that u has length p.

The last case is when $i'_s = 2^{l(n)}$ and $i'_u < 2^{l(n)}$ (that is, s is in the active domain, u is not). We reduce it to the previous case as follows: $s \prec u$ iff $s = v$ or $s \prec v$, where v is the immediate predecessor (in the $\prec$ relation) of u. Note that for u of length $l(n)$, its predecessor is in the active domain, so if we can state this condition, then the previous case applies to test if $s \prec v$. To check that a number m is such that v with $N(v) = m$ is an immediate predecessor of u, we consider two subcases. In the first subcase, $n + i'_u$ is odd (this can be tested with BIT). In that case, one should test if $2m + 1 = n + i'_u$. Note that in FO(BIT, $<$) we can only quantify over numbers not exceeding n, so this test is done by

$$\exists k\ (k + m = n) \wedge (k + i'_u = m + 1).$$

In the subcase when $n + i'_u$ is even, one should test if $2m = n + i'_u$, which is done by $\exists k\ (k + m = n) \wedge (k + i'_u = m)$.

Thus, we have shown that every Boolean query can be evaluated in AC^0 over nice databases. Now let Q be a Boolean relational query Q, that is expressible in RC(C) by a query Ψ. There is a family of circuits $\mathcal{C}$ that computes Ψ on nice databases. Now, for a relational database, let $enc_0(D)$ be the standard encoding under which elements of the active domain of size k are coded as integers $1, \ldots, k$ in binary. Given an arbitrary relational database D, consider $enc_0(D)$ as the input to $\mathcal{C}$. Let D_0 be a (nice) database over strings obtained from D by replacing the ith element of the active domain with the string s such that $N(s) = i$. Then $enc(D_0) = enc_0(D)$, and thus when it is given to $\mathcal{C}$, $\mathcal{C}$ returns $\Psi(D_0)$. But by genericity, we have $Q(D) = Q(D_0) = \Psi(D_0)$, which implies that Q is in AC^0.

It remains to show that the data complexity of generic queries in RC($\mathbf{S}_{\mathrm{len}}$) is in TC^0. Let Ψ be a generic query definable in RC($\mathbf{S}_{\mathrm{len}}$). For each database D, let $nice(D)$ be a database obtained from D as follows: let $adom(D) = \{s_1, \ldots, s_k\}$, where $s_1 \leq_{\mathrm{lex}} \ldots \leq_{\mathrm{lex}} s_k$. Then in $nice(D)$, each s_i from D is replaced by a string s'_i with $N(s'_i) = i$. Note that this transformation can be carried out in TC^0, as $\leq_{\mathrm{lex}}$ is in AC^0 by Corollary 4.5, and counting the number of elements satisfying a formula can be done in TC^0 [7]. Furthermore, by genericity, $D \models \Psi$ iff $nice(D) \models \Psi$. The latter can be checked in AC^0, which gives us a TC^0 upper bound on the data complexity of generic queries. The theorem is proved. □

One cannot draw any definite conclusions from the first statement of Theorem 4.10, as TC^0 is not yet separated from NP (although widely believed to be properly contained in DLogSpace). However, the second statement, and known lower bounds for AC^0 [2, 35] give us:

4.11 Corollary *Parity test and connectivity test are not definable in* RC($\mathbf{S}_{\mathrm{len}}$).

We now prove lower bounds that show the complexity of $\mathbf{S}_{\mathrm{len}}$ queries, although within PH, may be prohibitively high. Let MSO(SC) be the class of queries over SC expressible in monadic second-order logic. This includes queries of high-complexity, namely for each level of the polynomial hierarchy, PH, complete queries [3], in particular, NP-complete and coNP-complete ones (3-colorability and its complement). Such queries cannot be expressed over arbitrary databases in RC($\mathbf{S}_{\mathrm{len}}$) (e.g., not over nice ones); however, they can be expressed under some additional assumptions.

We say that the *width* of the active domain of an SC database D (over Σ^*) is k if k is the maximal size of a subset of $adom(D)$ whose elements are pairwise incomparable by the prefix relation. It should be noted that every database D can be transformed into a database D' of width 1 which is isomorphic to D with respect to the SC-predicates.

4.12 Proposition *For every fixed k, all* MSO(SC)*-expressible queries can be expressed over databases of width at most k in* RC($SC, \mathbf{S}_{\text{len}}$).

Proof. Assume without loss of generality that $0, 1 \in \Sigma$. For a database D of width k, the set of $\preceq$-maximal elements $\{s_1, \ldots, s_l\}$ of $adom(D)$ has cardinality $l \leq k$, and thus $prefix(D)$ is the union of chains $prefix(s_1), \ldots, prefix(s_l)$, where $prefix(s) = \{s' \mid s' \preceq s\}$. The idea of the proof is this: a subset Z of $prefix(s)$ can be modeled by a string $s_Z \in \{0,1\}^*$ of the same length as s, such that $s' \preceq s$ is in Z iff the prefix of s_Z of the length $|s'|$ ends on a 1.

Now suppose an MSO(SC) query Q is given. We assume it is expressed by an MSO sentence Φ in which all quantified second-order variables are distinct. Let $m_1, \ldots, m_k$ be fresh first-order variables (to be interpreted as maximal elements of $adom(D)$). We then associate with each second-order quantifier $\exists Z$ new first-order variables $s_Z^1, \ldots, s_Z^k$, and define the following transformation $\varphi \mapsto \varphi^\circ$ of subformulae of Φ:

- Every atomic subformula other than $Z(x)$, where Z is a second-order variable, is unchanged.
- Every subformula $Z(x)$ is replaced by $(Z(x))^\circ$ defined as

$$\bigvee_{i=1}^{k} x \preceq m_i \wedge \exists y \preceq s_Z^i \ \mathrm{el}(y, x) \wedge L_1(y).$$

- $(\varphi_1 * \varphi_2)^\circ = \varphi_1^\circ * \varphi_2^\circ$, where $*$ is $\wedge$ or $\vee$, $(\neg\varphi)^\circ = \neg\varphi^\circ$, $(\exists u\varphi)^\circ = \exists u\varphi^\circ$, where u is a first-order variable.
- A subformula $\exists Z\varphi$ is replaced by $(\exists Z\varphi)^\circ$ defined as

$$\exists s_Z^1, \ldots, s_Z^k \ \bigwedge_{i=1}^{k} \mathrm{el}(s_Z^i, m_i) \wedge \varphi^\circ.$$

The result of this transformation is an open RC($\mathbf{S}_{\text{len}}$) query $\Phi^\circ(m_1, \ldots, m_k)$. We now define a Boolean RC($\mathbf{S}_{\text{len}}$) as

$$\exists m_1 \in adom \ldots \exists m_k \in adom \ \forall u \in adom \bigvee_i u \preceq m_i \wedge \Phi^\circ(m_1, \ldots, m_k),$$

stating that $m_1, \ldots, m_k$ list all (not necessarily distinct) maximal elements of $adom(D)$, and that $\Phi^\circ(m_1, \ldots, m_k)$ holds. For a database of width at most k, this means that $\Phi^\circ(m_1, \ldots, m_k)$ holds for the list of all maximal elements in $adom(D)$, which happens iff $D \models \Phi$. □

Thus, while not computationally complete as RC_{concat}, RC($\mathbf{S}_{\text{len}}$) can express some queries that normally would not be expected to be expressible in a first-order language.

Recall that we had a linear time bound for the evaluation of Boolean RC($\mathbf{S}$)-queries on unary databases. We show next, that this might not be the case for RC($\mathbf{S}_{\text{len}}$). Even worse, there might be even no fixed polynomial bound.

We consider ordered graphs as finite structures with a universe U of the form $\{1, \ldots, n\}$, the natural order relation $<$ on U and a binary relation E. Let SC be the database schema with one unary relation name R.

4.13 Lemma *For every first-order formula φ on ordered graphs there is a* RC($SC, \mathbf{S}_{\text{len}}$)*-formula φ' and an algorithm which computes for each graph G an SC-database D_G such that $G \models \varphi$ if and only if $D_G \models \varphi'$. Furthermore, the algorithm works in time $O(n^2 \log n)$ on graphs with n vertices and the maximum length of a string in D_G is $2\lceil \log_2 n \rceil + 1$ and, consequently, the size of D_G is $O(n^2 \log n)$.*

Proof. We give the proof for $\Sigma = \{0, 1\}$. Let an ordered graph G with n vertices be given and let $m := \lceil \log_2 n \rceil$. We define D_G as follows. Let $a_1, \ldots, a_n$ denote the lexicographically first n strings of length m. We define the set R as

$$\{a_1, \ldots, a_n\} \cup \{a_i \cdot 0 \cdot a_i \mid i \leq n\} \cup \{a_i \cdot 1 \cdot a_j \mid (j, i) \in E\}.$$

Intuitively, the strings $a_1, \ldots, a_n$ represent the vertices of G. There is an edge from vertex j to vertex i if and only if $a_i \cdot 1 \cdot a_j \in R$. The vertices $a_i \cdot 0 \cdot a_i$ are used to get a_i from $a_j \cdot 1 \cdot a_i$.

It is straightforward to check that D_G has the desired size and can be produced in time $O(n^2 \log n)$ assuming a suitable representation of G.

The formula φ' is obtained from φ as follows. First, all subformulas of the form $\exists x \psi(x)$ are replaced by $\exists x \in adom(\neg \exists y\ y \prec x \wedge R(y)) \wedge \psi(x)$. Intuitively, the quantification is restricted to minimal elements of the active domain of D_G, i.e., to $a_1, \ldots, a_n$. Note however that the next two steps will introduce new unrestricted quantifiers.

Next, atomic formulas $x < y$ are replaced by

$$\exists z, z_0, z_1\ l_0(z) = z_0 \wedge l_1(z) = z_1 \wedge z_0 \preceq x \wedge z_1 \preceq y$$

Finally, atomic formulas $E(x, y)$ are replaced by

$$\begin{aligned} \exists x_1, x_2, y_1, y_2 \quad & l_0(x) = x_1 \wedge l_1(y) = y_1 \wedge x_1 \preceq x_2 \wedge y_1 \preceq y_2 \wedge R(x_2) \wedge R(y_2) \wedge \\ & \forall x_3, y_3 (x_1 \prec x_3 \preceq x_2 \wedge y_1 \prec y_3 \preceq y_2 \wedge \mathrm{el}(x_3, y_3)) \rightarrow (L_0(x_3) \leftrightarrow L_0(y_3)) \end{aligned}$$

which states the existence of strings x_2 and y_2 of the form $x0x$ and $y1x'$ and such that, second line of the formula, $x = x'$. It is straightforward to check that $G \models \varphi$ if and only if $D_G \models \varphi'$. □

It follows from the lemma that a linear (or fixed polynomial) bound for the evaluation of Boolean RC($\mathbf{S}_{\text{len}}$)-queries on unary databases would imply a fixed polynomial bound for the data complexity of first-order sentences on ordered graphs. It would imply further a fixed polynomial bound for the evaluation of first-order sentences on BIT-structures (cf., [6]). This, in turn, would separate first-order logic from least fixed point logic on such structures and therefore imply the validity of the *ordered conjecture* [49] with various consequences in complexity theory (see [6] for a discussion).

We cannot conclude from this connection that linear time evaluation for RC($\mathbf{S}_{\text{len}}$) queries on unary databases is impossible. But we cannot expect a proof as simple as that of Proposition 4.7 for RC($\mathbf{S}$).

4.3.3 Relational calculi over $\mathbf{S}_{\text{left}}$, $\mathbf{S}_{\text{reg}}$ and $\mathbf{S}_{\text{reg,left}}$

These calculi behave similarly to RC($\mathbf{S}$), although some complexity bounds are slightly different. From the isolation property shown for all the structures and from QE results we conclude the following:

4.14 Theorem RC($\mathbf{S}_{\text{left}}$), RC($\mathbf{S}_{\text{reg}}$), *and* RC($\mathbf{S}_{\text{reg,left}}$) *admit the restricted quantifier collapse.*

Furthermore, RC($\mathbf{S}^+_{\text{left}}$), RC($\mathbf{S}^+_{\text{reg}}$), *and* RC($\mathbf{S}^+_{\text{reg,left}}$) *admit the natural-active collapse.*

4.15 Corollary RC($\mathbf{S}_{\text{left}}$) *queries have* AC^0 *data complexity, while* RC($\mathbf{S}_{\text{reg}}$) *and* RC($\mathbf{S}_{\text{reg,left}}$) *queries have* NC^1 *data complexity. Furthermore, every generic query expressible in* RC($\mathbf{S}_{\text{left}}$) *or* RC($\mathbf{S}_{\text{reg}}$) *is expressible in* RC($<$).

Proof. The proof of the AC^0 bound is the same as for Corollary 4.5 except that we need to show that each fixed $\mathbf{S}_{\text{left}}$ formula can be evaluated in AC^0. By the quantifier elimination result quoted in the proof of Theorem 4.14, it suffices to show that every fixed quantifier-free formula in $\mathbf{S}^+_{\text{left}}$ can be evaluated in AC^0. For that, we notice that every $\mathbf{S}^+_{\text{left}}$ term can be evaluated in AC^0 (since both $x - a$ and $a \cdot x$ operations are available), and the rest follows the proof for $\mathbf{S}$.

For $\mathbf{S}_{\text{reg}}$, we again use the collapse result and the proof that RC($\mathbf{S}$) queries with active-domain quantification can be evaluated in AC^0 (and hence NC^1). The only difference is in evaluating the P_L predicates, which can no longer be done in AC^0 as L may not be star-free. However, every regular language is in NC^1 [65], and thus P_L can be evaluated in NC^1 on its inputs, showing that the data complexity of RC($\mathbf{S}_{\text{reg}}$) is in NC^1. The proof for $\mathbf{S}_{\text{reg,left}}$ combines the proofs for $\mathbf{S}_{\text{left}}$ and $\mathbf{S}_{\text{reg}}$.

The last statement follows from the collapse result and [10]. □

Note the contrast of the above with Proposition 4.12, which implies that relational calculus over $\mathbf{S}_{\text{len}}$ contains problems complete for each level of the polynomial hierarchy. Theorem 4.14 is the key for obtaining low data complexity. It follows from the isolation property of the underlying structure, which fails for $\mathbf{S}_{\text{len}}$ as it does not have finite VC-dimension (recall Proposition 3.2).

4.4 Safe Queries

All the relational calculi we study here contain queries that sometimes produce infinite output. Thus one of our goals is to syntactically capture the safe queries in these languages, and to be able to analyze safety properties of a query – for example, given an arbitrary query and a database, to tell whether the output of the query on that database is finite. We saw that this cannot be done if the set of operations includes concatenation. In contrast, for our five structures, we can syntactically describe safe queries, give an algebra that captures these queries, and extend the major decidability results for query safety analysis that hold for pure relational calculus.

4.4.1 Effective syntax for safe queries: defining finiteness

The simplest way to show that safe queries in RC($\mathcal{M}$) have effective syntax is to show that one can test if a given query returns a finite result on a given database. To do so, it is enough to ensure that *finiteness is definable* in RC($\mathcal{M}$). Formally, finiteness is definable in RC($\mathcal{M}$) if there exists a sentence Φ^{safe} in the language of $\mathcal{M}$ and SC expanded with a single new unary predicate symbol U such that for any query $\varphi(x)$ and any database D, $(D, \varphi(D)) \models \Phi^{\text{safe}}$ iff $\varphi(D)$ is finite. For example, finiteness is easily definable in RC($\mathbf{S}_{\text{len}}$) by

$$\exists y \forall x (U(x) \to \exists z \prec y \ \text{el}(z, x)).$$

Once finiteness is definable, an enumeration of safe queries can easily be obtained. Given a query $\varphi(\vec{x})$, let $\psi_\varphi(x)$ be another relational calculus query that defines the active domain of the output of φ. Let $\Phi^{\text{safe}}_\varphi$ be the Boolean query obtained from Φ^{safe} be replacing $U(\cdot)$ by $\psi_\varphi(\cdot)$. Then $\varphi(\vec{x}) \wedge \Phi^{\text{safe}}_\varphi$ lists all safe queries.

For traditional relational calculus, and for its analogs over order constraints, linear constraints, and polynomial constraints, finiteness can easily be shown to be definable [11]. It is thus surprising that for RC($\mathbf{S}$) this approach does not work:

4.16 Proposition *Finiteness is not definable in* RC($\mathbf{S}$).

Proof. We prove the proposition for $\Sigma = \{0, 1\}$; it is straightforward to generalize this for any alphabet. We consider databases with one unary predicate U. We show by an Ehrenfeucht-Fraïssé game argument that, for each k, there are databases A_k and B_k such that U is a finite set in A_k and an infinite set in B_k but A_k and B_k can not be distinguished by a RC($\mathbf{S}$)-formula of quantifier rank k.

Let $k \geq 0$ be fixed.

Let T_i denote the set of strings of length at most i. Intuitively, T_i is the full binary tree of depth i (and formally it is the same as $\Sigma^{\leq i}$).

We use $\equiv_k$ to denote equivalence in the k-round Ehrenfeucht game on structures based on $\mathbf{S}$ and $\equiv^s_k$ to denote equivalence in the k-round Ehrenfeucht-Fraïssé game on strings.

We will use the following Claim.

Claim 1 *There exist N and $n > 0$ (depending on k) such that for each $i \geq N$ it holds that $(\mathbf{S}, T_i) \equiv_k (\mathbf{S}, T_{i+n})$. Without loss of generality we can choose N as a multiple of n.*

Proof of the Claim: For every k, $\equiv_k$ has finitely many equivalence class. Let N be this number. By the pigeon-hole principle there exists two integers i, j such that $i \leq N + 1$ and $j \leq N + 1$ and $T_i \equiv_k T_j$. We show that for any two integers u, v, $T_u \equiv T_v$ implies $T_{u+1} \equiv T_{v+1}$, the claim will then follow with $n = j - i$. To prove the latter notice that T_{u+1} is simply $|\Sigma|$ copies of T_u plus one node. Similarly T_{v+1} is simply $|\Sigma|$ copies of T_v plus one node. The FO$_k$ strategy on T_{u+1} and T_{v+1} mimics the strategy for T_u and T_v on each copy separately and the root is played as soon as the other root is played. □

Let $m = 2^k n$ and $M = 2^{3k} kn + N$. Let A_k be $(\mathbf{S}, T_M)$.

Next, we define an infinite set S such that A_k and $(\mathbf{S}, S)$ can not be distinguished by a formula of depth k. Let h be the string homomorphism which maps 0 to 0^m and 1 to 1^m. We call a string w *normal* if it is of the form $h((01)^i)$, for some $i \geq 0$. We call w *semi-normal* if it is $h(v)$ for some string v. The set S is defined as the set of all strings of the form uv, where u is a normal string and v is a string of length at most $N + 2m$. We set $B_k = (\mathbf{S}, S)$. Note that S is prefix-closed and that all maximal strings in B_k have a length which is a multiple of n.

For two strings u and w such that u is a prefix of w we write $A_k[u, w]$ for the substructure of A_k that consists of all strings v such that u is a prefix of v but w is not a strict prefix of v and analogously for B_k. Let Mod_n denote a sequence $Z_0, \ldots, Z_{n-1}$ of unary relations over (initial segments of) the natural numbers such that $Z_i(j)$ holds if and only if $(j \bmod n) = i$.

For later use we need the following lemma.

4.17 Lemma *(a) Let v, w be semi-normal strings and v', w' normal strings such that v is a prefix of w and v' is a prefix of w' and $|w| \leq M - N$. Let $u = w - v$ and $u' = w' - v'$. If $(u, Mod_n) \equiv_k^s (u', Mod_n)$ then $A_k[v, w] \equiv_k B_k[v', w']$.*

(b) $(h(0), Mod_n) \equiv_k^s (h(00), Mod_n)$ and $(h(01), Mod_n) \equiv_k^s (h(001), Mod_n)$.

(c) For each $i \geq 2^k + 1$ it holds that $(h((01)^{2^k+1}), Mod_n) \equiv_k^s (h((01)^i), Mod_n)$.

Proof of Lemma 4.17.

(a) Intuitively in the tree T_M, $[v, w]$ consists of the path from v to w and of trees branching off the strings on that path. By definition of A_k the tree branching off a string z of the path has depth $M - |z| - 1$ which is at least N and congruent to $N - |z - v| - 1$ modulo n, as M, N and $|v|$ are multiples of n. More precisely, we refer here to the tree that is rooted at the child of z which is not a prefix of w. Analogously, if z' is a string of the path from v' to w' in B_k there is a tree of depth $(2m+N) - |z' - y'| - 1$ branching off z', where y' is the longest normal string which is a prefix of z'. Hence, the depth of this tree is at least N and it is congruent to $N - |z' - v'| - 1$ modulo n. We can conclude from Claim 1 that the branching trees at z and z' are k-equivalent, whenever $|z - v|$ and $|z' - v'|$ are congruent modulo n.

By combining the winning strategy of the duplicator on (u, Mod_n) and (u', Mod_n) with the winning strategies on the off-branching trees we get (a).

(b) The first statement is shown by a standard game argument using the fact that $h(0)$ is the concatenation of 2^k strings of length n. Each of these substrings is identically labeled by Mod_n. In a k round game this can not be distinguished from the concatenation of $2 \cdot 2^k$ such strings. The second statement follows directly from the first one.

(c) This can also be shown by a standard argument.

□

Next, we have to show that $(\mathbf{S}, T_M) \equiv_k (\mathbf{S}, S)$.

Claim 2 *The duplicator can play the k round Ehrenfeucht-Fraïssé game in a way that guarantees that the following holds after l rounds of the game.*

Let $\vec{a} = a_1, \ldots, a_l$ denote the selected elements of A_k and let $\vec{b} = b_1, \ldots, b_l$ denote the corresponding elements in B_k.

There is a semi-normal string p_l and a normal string q_l (the pivot strings*) such that*

1. *None of the a_i has p_l as a prefix and none of the b_i has q_l as a prefix.*
2. *$(A_k - p_l*, \vec{a}) \equiv_k (B_k - q_l*, \vec{b})$.*
3. *$|p_l| \leq l2^{3k}n$.*

Here, $A_k - p_l*$ denotes the substructure of A_k in which all strings that have p_l as a strict prefix are omitted and in which p_l is a distinguished constant (and analogously for $B_k - q_l*$).

Proof of the claim. It should be noted that, as q_l is normal, $B_k - q_l*$ only contains a finite part of S. In the proof, it will always be the case that p_l is a prefix of p_{l+1} and q_l is a prefix of q_{l+1}.

Because of condition (1) we can conclude from (2) that there is a partial **S**-isomorphism from $\vec{a})$ to $\vec{b})$ at the end of the game. Hence the claim implies the statement of the theorem.

We prove the claim by induction on l. For $l = 0$ we choose $p_0 = q_0 = \epsilon$. This guarantees (1)-(3).

Now assume that, for some $l < k$, l rounds have been played and there are p_l and q_l such that (1)-(3) hold. We show that the duplicator can play in a way such that, for suitable choices of p_{l+1} and q_{l+1} (1)-(3) also holds for $l + 1$.

We distinguish 3 cases.

Case 1. The spoiler chooses a vertex in $A_k - p_l*$ or $B_k - q_l*$. Then we simply set $p_{l+1} = p_l$ and $q_{l+1} = q_l$ and (1)-(3) follow directly.

Case 2. The spoiler chooses a string a_{l+1} which has p_l as a prefix. Let $u = a_{l+1} - p_l$.

- If u is of the form $h(01) \cdot v$, for some v then we set $p_{l+1} = p_l \cdot h(001)$ and $q_{l+1} = q_l \cdot h(01)$.
- Otherwise we set $p_{l+1} = p_l \cdot h(01)$ and $q_{l+1} = q_l \cdot h(01)$.

In both subcases, p_{l+1} is not a prefix of a_{l+1}. As $|p_{l+1}| \leq |p_l| + 3m \leq M - N$ it follows from Lemma 4.17 (a) and (b) that in both subcases $A_k[p_l, p_{l+1}] \equiv_k B_k[q_l, q_{l+1}]$. Therefore the duplicator can choose a string b_{l+1} in $B_k[q_l, q_{l+1}]$ that guarantees a winning strategy on $A_k[p_l, p_{l+1}]$ and $B_k[q_l, q_{l+1}]$ for $k - 1$ more rounds. By combining this winning strategy with the winning strategy on $(A_k - p_l*, \vec{a})$ and $(B_k - q_l*, \vec{b})$ we obtain a $k - l - 1$ round winning strategy on $(A_k - p_{l+1}*, \vec{a}, a_{l+1})$ and $(B_k - q_{l+1}*, \vec{b}, b_{l+1})$. Hence, we can conclude (2). Furthermore, of course, (1) and (3) hold.

Case 3. The spoiler chooses a string b_{l+1} which has q_l as a prefix. Let i be maximal such that b_{l+1} can be written as $q_l \cdot h((01)^i) \cdot v$, for some string v. We choose $q_{l+1} = q_l \cdot h((01)^{i+1})$ and

$$p_{l+1} = \begin{cases} p_l \cdot h((01)^{i+1}) & \text{if } i \leq 2^k, \\ p_l \cdot h((01)^{2^k+1}) & \text{otherwise.} \end{cases}$$

The choice of q_{l+1} guarantees that it is not a prefix of b_{l+1}. From Lemma 4.17 (c) and (a) it follows that in both subcases $A_k[p_l, p_{l+1}] \equiv_k B_k[q_l, q_{l+1}]$. This implies the existence of an

appropriate a_{l+1} in $A_k[p_l, p_{l+1}]$ such that (2) holds again. By the choice of p_{l+1} and induction we also get (1) and (3).

□

This completes the proof of the proposition. □

4.4.2 Effective syntax for safe queries: range-restriction

While post-checking finiteness is a way to obtain effective syntax for safe queries, one often wishes to have a more explicit representation of safe queries. It turns out that we can get natural representations for safe queries in RC($\mathbf{S}$) and RC($\mathbf{S}_{\text{len}}$) and other calculi. The technique we use derives from work on safe languages with linear or polynomial constraints [11]: for each query Q, we effectively construct another *safe* query Q' that gives an upper bound on $Q(D)$, if it is finite. Such explicit constructions are used to prove the theorem below, as well as to provide relational algebra extensions.

We follow the idea of range-restriction as presented in [11]. A formula $\gamma(x, z)$ over $\mathcal{M}$ is called *algebraic* if for every b, the set $\{a \mid \mathcal{M} \models \gamma(a, b)\}$ is finite. An RC($\mathcal{M}$) query in *range-restricted form* is a pair $Q = (\gamma(x, y), \varphi(x_1, \ldots, x_n))$, where φ is an arbitrary query and γ is an algebraic formula over $\mathcal{M}$. The semantics is given by $\varphi(\vec{x}) \wedge \exists \vec{y} \in adom\ (\bigwedge_i \gamma(x_i, y_i))$. That is,

$$Q(D) \quad = \quad \gamma(adom(D))^n \cap \varphi(D),$$

where $\gamma(X) = \{a \mid \gamma(a, b) \text{ for some } b \in X\}$. Clearly, every query in range-restricted form is safe.

4.18 Theorem *Let $\mathcal{M}$ be $\mathbf{S}$, or $\mathbf{S}_{\text{left}}$, or $\mathbf{S}_{\text{reg}}$, or $\mathbf{S}_{\text{reg,left}}$, or $\mathbf{S}_{\text{len}}$. Then there is a recursive set Γ of algebraic formulae over $\mathcal{M}$ such that, given a query $\varphi(\vec{x})$ in* RC($\mathcal{M}$)*, there is $\gamma(x, y) \in \Gamma$ with the property that the range-restricted query $Q = (\gamma, \varphi)$ coincides with φ on all databases over which φ is safe.*

Proof. The proof is based on a number of lemmas, which show that if a query $\varphi(x)$ is satisfied by an element that is sufficiently far from $adom(D)$, then φ returns an infinite result on D. The definition of "sufficiently far" depends on the particular structure.

First, we need two observations. The first one is a generalized version of the pumping lemma for finite automata.

4.19 Lemma *For each sequence $L_1, \ldots, L_m$ of regular languages there is a number k such that for each string z, $|z| > k$, there are strings u, v, w, with $z = uvw$ and $|v| > 0$, such that for each string x, each $j \in \{1, \ldots, m\}$ and each $i > 0$,*

$$xuvw \in L_j \iff xuv^iw \in L_j.$$

Proof of Lemma 4.19. Let, for each $i \leq m$, A_i be a deterministic automaton for L_i with transition function δ_i. Without loss of generality we assume that all automata have the same set $\{1, \ldots, n\}$ of states with 1 as the initial state. Let $k := n^{nm}$ and z be a string with $|z| > k$. For each $j \leq m$, $\alpha \leq n$ and $l \leq |z|$, let $q_{j\alpha l}$ be defined as $\delta_j(\alpha, z[1, l])$, where $z[1, l]$ is the prefix of z of length l.

I.e., $q_{j\alpha l}$ is the state of A_j after reading the first l symbols of z starting from state α. As $|z| > k$ there must be $l_1 \neq l_2$ such that $q_{j\alpha l_1} = q_{j\alpha l_2}$, for all $j \leq m$ and $\alpha \leq n$. Let u, v, w be chosen such that $z = uvw$, u is the prefix of z of length l_1 and v is of length $l_2 - l_1$. We claim that for every $j \leq m$, every $i > 0$ and every string x, $xuvw \in L_j$ if and only if $xuv^iw \in L_j$. Indeed, let α be the state $\delta_j(1, x)$. Then, as $q_{j\alpha l_1} = q_{j\alpha l_2}$ we have $\delta_j(\alpha, u) = \delta_j(\alpha, uv) = \delta_j(\alpha, uv^i)$. Therefore $xuvw$ is accepted by A_j if and only if xuv^iw is accepted by A_j. □

Using this lemma, we show:

Claim. *Let $\mathcal{M} = \langle \Sigma^*, \Omega \rangle$ be such that all operations in Ω are definable in $\mathbf{S}_{\text{len}}$. Then, for every $r > 0$, there exists $k > 0$ such that for any string s with $|s| \geq k$, there are infinitely many strings s' satisfying $(\mathcal{M}, s) \equiv_r (\mathcal{M}, s')$.*

Proof of the claim. Indeed, let $\alpha_1(x), \ldots, \alpha_l(x)$ list formulae (of quantifier rank r) that define all the r-types of a single string over $\mathcal{M}$. Since each α_i is definable over $\mathbf{S}_{\text{len}}$, there is a DFA A_i which accepts a string s iff $\mathcal{M} \models \alpha_i(s)$ [14]. In particular, the set of strings s which make $\alpha_i(s)$ true is a regular language L_i. From Lemma 4.19 it follows, that there is a k such that, for each string s with $|s| > k$ there are infinitely many strings s' that are contained exactly in the same languages L_i as s, i.e., make the same formulas α_i true, which implies $(\mathcal{M}, s) \equiv_r (\mathcal{M}, s')$. This proves the claim. □

Given $C \subseteq \Sigma^*$ and $s \in \Sigma^*$, let $d(s, C)$ be $|s| - |\text{Meet}(s, C)|$, that is, the length of the relative suffix of $\text{Meet}(s, C)$ in s.

Given a database D, let $\mathit{prefix}(D) = \{s \mid s \preceq s', s' \in \mathit{adom}(D)\}$.

4.20 Lemma *Let $\varphi(x)$ be a $\text{RC}(\mathbf{S})$ query. Then there exists a number $k > 0$, such that the following holds. If $D \models \varphi(s)$ for some s with $d(s, \mathit{prefix}(D)) > k$ then there are infinitely many strings c such that $D \models \varphi(c)$. If φ only uses prefix-restricted quantification then k can be effectively computed.*

Proof of Lemma 4.20. By Corollary 4.4 we may assume without loss of generality that all quantification in φ is prefix-restricted. Let r be the quantifier rank of φ. We show that we can find k such that the following holds. Let D be a database, and s a string with $d(s, \mathit{prefix}(D)) > k$. For a string u, let $C_u = \mathit{prefix}(D) \cup \{s' \mid s' \preceq u\}$. Then there are infinitely many strings u such that the duplicator has a winning strategy for the r-round Ehrenfeucht game on C_s and C_u (with the partial isomorphism being with respect to the operations of $\mathbf{S}$, and with s mapped to u); moreover, in the winning strategy, the duplicator simply copies the spoiler's moves on $\mathit{prefix}(D)$. Note that this condition implies that in the final position all the SC-relations are preserved, and hence $D \models \varphi(s)$ iff $D \models \varphi(u)$, thus implying the lemma.

To prove the above condition, let $k > 0$ be given by the claim. Consider s with $d(s, \mathit{prefix}(D)) > k$, and let s' be the relative suffix of $\text{Meet}(s, \mathit{prefix}(D))$ in s. We have $|s'| > k$. We then have infinitely many strings u' such that $(\mathbf{S}, s') \equiv_r (\mathbf{S}, u')$. Take any such string u', and form a new string $u = (\text{Meet}(s, \mathit{prefix}(D))) \cdot u'$. It is clear that the required strategy exists for the duplicator on C_s and C_u.

To show that k can be found from φ, note first that the conversion into a query with prefix-bounded quantification is effective, and the claim is effective too, as any $\mathbf{S}_{\text{len}}$ formula can be effectively converted into an automaton. The lemma is proved. □

Next we define $\downarrow D = \{s \mid |s| \leq |s'|, s' \in adom(D)\}$.

4.21 Lemma *Let $\varphi(x)$ be a $\mathrm{RC}(\mathbf{S}_{\mathrm{len}})$ query. Then there exists a number $k > 0$ such that the following holds. If $D \models \varphi(s)$ for some s with $d(s, \downarrow D) > k$ then there are infinitely many strings c such that $D \models \varphi(c)$. If φ only uses length-restricted quantification then k can be effectively computed.*

Proof of Lemma 4.21. By Proposition 4.8 we may assume without loss of generality that in $\varphi(x)$ all quantification is length-restricted. Let r be the quantifier rank of φ. For any string s, let $\mathbf{S}^s_{\mathrm{len}}$ be the structure $(\downarrow s, \prec, (L_a)_{a\in\Sigma}, \mathrm{el}, s)$. By the Claim, we can find a number k such that for any string s of $|s| > k$, there exist infinitely many strings s' of $|s'| > k$ with $\mathbf{S}^s_{\mathrm{len}} \equiv_r \mathbf{S}^{s'}_{\mathrm{len}}$. Note that k can be found effectively for a given φ.

Now assume that for some D and s, $D \models \varphi(s)$ with $d(s, \downarrow D) > k$. Let m be the maximum length of a string in $adom(D)$, and s_0 the prefix of s of length m. Then $s = s_0 \cdot s_1$ for a string s_1 of $|s_1| > k$. We now show that there are infinitely many strings s' of length greater than $m + k$ such that the duplicator has a winning strategy in the r-round Ehrenfeucht game on $\mathbf{S}^s_{\mathrm{len}}$ and $\mathbf{S}^{s'}_{\mathrm{len}}$ such that the play is the identity function when restricted to strings of length not exceeding m. Clearly, this suffices to prove the lemma, since $|x| \leq m$ for all $x \in adom(D)$ and thus $(D, s) \equiv_r (D, s')$ and $D \models \varphi(s')$.

Consider any string s'_1 such that $\mathbf{S}^{s_1}_{\mathrm{len}} \equiv_r \mathbf{S}^{s'_1}_{\mathrm{len}}$ (we know that there are infinitely many of them), and let s' be $s_0 \cdot s'_1$. We prove that the duplicator wins the r-round game on $\mathbf{S}^s_{\mathrm{len}}$ and $\mathbf{S}^{s'}_{\mathrm{len}}$. The strategy is as follows. The duplicator maintains (for his memory) a separate game on $\mathbf{S}^{s_1}_{\mathrm{len}}$ and $\mathbf{S}^{s'_1}_{\mathrm{len}}$. If the spoiler plays a string of length not exceeding m, the duplicator's response is the same string. Assume that the spoiler plays x of $|x| > m$. Let $x = x_0 \cdot x_1$ with x_0 being the length m prefix of x. Assume that the spoiler plays it in $\mathbf{S}^s_{\mathrm{len}}$ (if the spoiler plays in $\mathbf{S}^{s'}_{\mathrm{len}}$, the proof is identical). The duplicator then looks at the current position of the auxiliary game on $\mathbf{S}^{s_1}_{\mathrm{len}}$ and $\mathbf{S}^{s'_1}_{\mathrm{len}}$ (which is empty until the spoiler makes the first move of length $> m$), and extends it by one move: spoiler's move is x_1 on $\mathbf{S}^{s_1}_{\mathrm{len}}$, and the response is a string x'_1 in $\mathbf{S}^{s'_1}_{\mathrm{len}}$ according to the winning strategy $\mathbf{S}^{s_1}_{\mathrm{len}} \equiv_r \mathbf{S}^{s'_1}_{\mathrm{len}}$. Having done that, the duplicator returns to the game on $\mathbf{S}^s_{\mathrm{len}}$ and $\mathbf{S}^{s'}_{\mathrm{len}}$, and responds by $x_0 \cdot x'_1$ in $\mathbf{S}^{s'}_{\mathrm{len}}$.

We now show that the duplicator wins the game. Clearly all L_a predicates are preserved. Assume that in $\mathbf{S}^s_{\mathrm{len}}$, $u \prec v$, where u and v are two moves in the game. Let u' and v' be the corresponding moves played on $\mathbf{S}^{s'}_{\mathrm{len}}$. If both u and v are of length at most m, then $u' = u, v' = v$ and $u' \prec v'$. If $|u| \leq m$ and $|v| > m$, then $u' = u$, and v' is of the form $v_0 \cdot v'_1$, where v_0 is the prefix of v of length m, and thus $u' \prec v'$. If $|u|, |v| > m$ then $u' \prec v'$ by the winning strategy on $\mathbf{S}^s_{\mathrm{len}}$ and $\mathbf{S}^{s'}_{\mathrm{len}}$ and the fact that u and v have the same prefix of length m. Next, assume $\mathrm{el}(u, v)$ holds. The case of the length $\leq m$ is trivial. If $|u|, |v| > m$, then $u = u_0 \cdot u_1, v = v_0 \cdot v_1$, where u_0, v_0 are length m prefixes, and by the description of the duplicator's strategy, $u' = u_0 \cdot u'_1$ and $v' = v_0 \cdot v'_1$, where u'_1, v'_1 are moves taken from the auxiliary game on $\mathbf{S}^{s_1}_{\mathrm{len}}$ and $\mathbf{S}^{s'_1}_{\mathrm{len}}$. Since the duplicator wins the auxiliary game, we have $|u_1| = |u'_1|$ and $|v_1| = |v'_1|$, and thus $\mathrm{el}(u', v')$ holds. This completes the proof of the lemma. □

For any set X, let $N^0_p(X) = \{s - s_1 + s_2 \mid s \in X, |s_1|, |s_2| \leq p\}$, and let $N_p(X) = prefix(N^0_p(X))$ (that is, the prefix-closure of $N^0_p(X)$). Note that $N_p(X) = N^0_p(prefix(X))$, and $N_k(N_m(X)) \subseteq N_{k+m}(X)$.

4.22 Lemma *Let $\varphi(x)$ be a $\mathrm{RC}(\mathbf{S}_{\mathrm{left}})$ query. Then there exist numbers $l, m > 0$ such that the following holds. If $D \models \varphi(s)$ for some s with $d(s, N_m(prefix(D))) > l$ then there are infinitely many strings c such that $D \models \varphi(c)$.*

Proof of Lemma 4.22. This follows from the normal form for $\mathbf{S}_{\mathrm{left}}$ (Corollary 3.15) and Lemma 4.20. □

4.23 Lemma *Given a $\mathrm{RC}(\mathbf{S}_{\mathrm{reg}})$ query $\varphi(x)$, there exists $k > 0$ such that whenever $D \models \varphi(s)$ with $d(s, prefix(D)) > k$, there are infinitely many strings c such that $D \models \varphi(c)$.*

Proof of Lemma 4.23. To show this, assume by the restricted quantifier collapse and quantifier-elimination for $\mathbf{S}^{+}_{\mathrm{reg}}$ that φ is of the form

$$Qy_1 \in adom \ldots Qy_l \in adom \bigvee_i \bigwedge_j \alpha_{ij}(x, \vec{y}),$$

where each α_{ij} is either an atomic or negated atomic SC-formula, or an $\mathbf{S}_{\mathrm{reg}}$ formula not involving the variable x, or a formula of the form $P_L(t_1(x, \vec{y}), t_2(x, \vec{y}))$, where t_i is either ϵ or a $\sqcap$-term. Let $L_1, \ldots, L_m$ be the regular languages such that the formulae P_{L_i} appear in φ. We denote the quantifier-free part (that is $\bigvee_i \bigwedge_j \alpha_{ij}$) by $\beta(x, \vec{y})$.

Let $i > 1$ and $D \models \varphi(s)$ with $d(s, prefix(D)) > k$. We apply Lemma 4.19 to $z = s - (\mathrm{Meet}(s, prefix(D)))$, and let $c = (\mathrm{Meet}(s, prefix(D))) \cdot uv^i w, i > 1$. We now show that for every $\vec{y}_0 \in (adom(D) \cup \{\epsilon\})^l$, it is the case that $D \models \beta(s, \vec{y}_0)$ iff $D \models \beta(c, \vec{y}_0)$. This will imply $D \models \varphi(s) \leftrightarrow \varphi(c)$ (see [10]) thus proving the result. To prove $D \models \beta(s, \vec{y}_0) \leftrightarrow \beta(c, \vec{y}_0)$, it suffices to show that $D \models P_L(t_1(c, \vec{y}_0), t_2(c, \vec{y}_0)) \leftrightarrow P_L(t_1(s, \vec{y}_0), t_2(s, \vec{y}_0))$, where $L \in \{L_1, \ldots, L_m\}$, as for all other types of formulae α_{ij} the equivalence is trivial.

We now fix $\vec{y}_0 \in (adom(D) \cup \{\epsilon\})^l$ and consider the atomic formula $\chi(x) = P_L(t_1(x, \vec{y}_0), t_2(x, \vec{y}_0))$. If t_j, $j = 1, 2$ involves meets of x with some of the components of $\vec{y}_0$, then the value of t_j will be the same on s and on c, as $\mathrm{Meet}(s, prefix(D)) = \mathrm{Meet}(c, prefix(D))$. Thus, if both t_1 and t_2 involve such meets, we have $D \models \chi(s) \leftrightarrow \chi(c)$.

The other case is when t_2 is simply x, and in this case t_1 is either ϵ or $x \sqcap y_0^{i_1} \sqcap \ldots \sqcap y_0^{i_p}$, for some components of $\vec{y}_0$ (we can include x in the $\sqcap$-term without loss of generality, since its value must be a prefix of x, by the definition of P_L). Since $\mathrm{Meet}(s, prefix(D)) = \mathrm{Meet}(c, prefix(D))$, $t_1(s)$ equals $t_1(c)$ and belongs to $prefix(D)$. To prove $D \models \chi(s) \leftrightarrow \chi(c)$, it then suffices to show that $s - s_0 \in L$ iff $c - s_0 \in L$, which follows immediately from Lemma 4.19. This completes the proof of the lemma. □

Finally, we need a lemma for $\mathbf{S}_{\mathrm{reg,left}}$. Its proof follows from the normal form for $\mathbf{S}_{\mathrm{reg,left}}$ (Corollary 3.25) and Lemma 4.23.

4.24 Lemma *Let $\varphi(x)$ be a $\mathrm{RC}(\mathbf{S}_{\mathrm{reg,left}})$ query. Then there exist numbers $l, m > 0$ such that the following holds. Assume that $D \models \varphi(s)$ for some s with $d(s, N_m(prefix(D))) > l$. Then there are infinitely many strings c such that $D \models \varphi(c)$.*

Proof of Theorem 4.18, completed. To prove the theorem, take an arbitrary query $\psi(\vec{y})$ and form

$\varphi(x)$ that defines the active domain of the output of ψ, that is, $\varphi(x)$ is

$$\exists y_2, \ldots, y_n \psi(x, y_2, \ldots, y_n) \ \vee \ldots \vee \ \exists y_1, \ldots, y_{n-1} \psi(y_1, \ldots, y_{n-1}, x).$$

It then suffices to prove the theorem for $\varphi(x)$, since ψ is safe for D iff φ is safe for D, and thus for any γ such that (γ, φ) is equivalent to φ on all D for which φ is safe, the same would be true for (γ, ψ) and ψ.

Having reduced the problem to queries on one variable, simply apply the corresponding lemmas. For RC($\mathbf{S}$), given $\varphi(x)$, find the number k as in Lemma 4.20, and let $\gamma(x, y)$ say that x is a prefix of the string of the form $y \cdot s$ with $|s| \leq k$. From Lemma 4.20 it follows that (γ, φ) is equivalent to φ on any D for which φ is safe. Finally, γ is clearly algebraic, and expressible over $\mathbf{S}$ for any fixed k.

For RC($\mathbf{S}_{\text{len}}$), given $\varphi(x)$, we get k from Lemma 4.21 and let $\gamma(x, y)$ be an $\mathbf{S}_{\text{len}}$ formula saying that the length of x is at most the length of y plus k. Clearly, this is expressible for each fixed k, and (γ, φ) coincides with φ on any D for which φ is safe. This completes the proof of the theorem.

The proof for $\mathbf{S}_{\text{left}}$ is similar: one gets l, t from Lemma 4.22, and the formula $\gamma(x, y)$ says that x is at the distance at most l from a prefix of a string of the form $y - e + f$, with $|e|, |f| \leq t$. The proofs for $\mathbf{S}_{\text{reg}}$ and $\mathbf{S}_{\text{reg,left}}$ follow the same idea. This concludes the proof of Theorem 4.18. □

4.25 Corollary *For each of*

- RC($\mathbf{S}$),
- RC($\mathbf{S}_{\text{left}}$),
- RC($\mathbf{S}_{\text{reg}}$),
- RC($\mathbf{S}_{\text{reg,left}}$),
- RC($\mathbf{S}_{\text{len}}$),

the classes of range-restricted and safe queries coincide, and safe queries have effective syntax.

Note that for queries in RC($\mathbf{S}$) and RC($\mathbf{S}_{\text{len}}$) that use a restricted form of quantification (prefix or length), the proof gives us a stronger result: namely, the formula γ can be effectively found for a given φ.

4.4.3 Relational algebras

It is a classical result of relational database theory that the set of safe relational calculus queries is precisely the set of relational algebra queries [1]. This result extends to string calculi considered here: safety theorems proved earlier can be used to show that safe queries in RC($\mathbf{S}$) and RC($\mathbf{S}_{\text{len}}$) can be captured by appropriate extensions of relational algebra.

Let safe_RC($\mathcal{M}$) be the class of all safe queries in RC($\mathcal{M}$). To define algebras capturing safe_RC($\mathcal{M}$) for the previous two structures, we need a number of operations extending the usual relational algebra (that is, selection σ, projection π, cartesian product $\times$, difference $-$, union $\cup$):

R_ϵ: is the constant unary relation $\{\epsilon\}$.

σ_α: for a formula $\alpha(x_1, \ldots, x_n)$. On an n-attribute relation R, it returns the set of tuples $(s_1, \ldots, s_n)$ from R such that $\alpha(s_1, \ldots, s_n)$ holds.

$\texttt{prefix}_i$: On an m-attribute relation R, it returns the $m+1$-attribute relation $\{(s_1, \ldots, s_{m+1}) \mid (s_1, \ldots, s_m) \in R, s_{m+1} \preceq s_i\}$.

$\texttt{addl}_i^a$, $a \in \Sigma$: On an m-attribute relation R, it returns the $m+1$-attribute relation $\{(s_1, \ldots, s_{m+1}) \mid (s_1, \ldots, s_m) \in R, s_{m+1} = s_i \cdot a\}$.

$\downarrow_i$: Given an m-attribute relation R, $\downarrow_i(R)$ returns $\{(s_1, \ldots, s_{m+1}) \mid (s_1, \ldots, s_m) \in R, |s_{m+1}| \leq |s_i|\}$.

$\texttt{addf}_i^a$, $a \in \Sigma$: On an m-attribute relation R, it returns the $m+1$-attribute relation $\{(s_1, \ldots, s_{m+1}) \mid (s_1, \ldots, s_m) \in R, s_{m+1} = a \cdot s_i\}$.

$\texttt{trim}_i^a$, $a \in \Sigma$: On an m-attribute relation R, it returns the $m+1$-attribute relation $\{(s_1, \ldots, s_{m+1}) \mid (s_1, \ldots, s_m) \in R, s_{m+1} = s_i - a\}$.

It should be pointed out that the formula α in σ_α does not refer to the database.

We now define the relational algebras:

RA($\mathbf{S}$) extends relational algebra with R_ϵ, σ_α, where α ranges over FO($\mathbf{S}$) formulae, $\texttt{prefix}_i$ and $\texttt{addl}_i^a$.

RA($\mathbf{S}_{\text{len}}$) extends relational algebra with R_ϵ, σ_α, where α ranges over FO($\mathbf{S}_{\text{len}}$) formulae, $\downarrow_i$, $\texttt{prefix}_i$, and $\texttt{addl}_i^a$.

RA($\mathbf{S}_{\text{left}}$) is the extension of relational algebra with σ_α (where α ranges over $\mathbf{S}_{\text{left}}$ formulae), $\texttt{prefix}$, $\texttt{addf}_i^a$ and $\texttt{trim}_i^a$.

RA($\mathbf{S}_{\text{reg}}$) extends relational algebra with R_ϵ, σ_α, where α ranges over FO($\mathbf{S}_{\text{reg}}$) formulae, $\texttt{prefix}_i$ and $\texttt{addl}_i^a$.

RA($\mathbf{S}_{\text{reg,left}}$) extends relational algebra with R_ϵ, σ_α, where α ranges over FO($\mathbf{S}_{\text{reg,left}}$) formulae, $\texttt{prefix}_i$, $\texttt{addl}_i^a$ and $\texttt{trim}_i^a$.

4.26 Theorem

- safe_RC($\mathbf{S}$) = RA($\mathbf{S}$);
- safe_RC($\mathbf{S}_{\text{len}}$) = RA($\mathbf{S}_{\text{len}}$);
- safe_RC($\mathbf{S}_{\text{left}}$) = RA($\mathbf{S}_{\text{left}}$);
- safe_RC($\mathbf{S}_{\text{reg}}$) = RA($\mathbf{S}_{\text{reg}}$);
- safe_RC($\mathbf{S}_{\text{reg,left}}$) = RA($\mathbf{S}_{\text{reg,left}}$).

Proof. We start with RA($\mathbf{S}$). Every RA($\mathbf{S}$) expression produces a finite result, and the standard translation from algebra to calculus (extended with rules for $\texttt{addl}$ and $\texttt{prefix}$) shows RA($\mathbf{S}$) $\subseteq$ RC($\mathbf{S}$).

For the converse, let $\varphi(\vec{x})$ be a safe RC($\mathbf{S}$) query. By Theorem 4.18, on every database D, the active domain of the output of φ on D is contained in the set $V_k[D] = \{x \mid d(x, \mathit{prefix}(D)) \leq k\}$ for some $k \geq 0$.

We first note that $V_k[D]$ is definable by an RA($\mathbf{S}$) expression. Indeed, the active domain of D is definable in relational algebra. Next, for each fixed string s and a finite set S, there is an expression $\texttt{addl}^s$ that defines the set $\{(s', s' \cdot s) \mid s' \in S\}$ simply by composing $\texttt{addl}^a$ operations. Thus, for $S = \mathit{adom}(D)$, we define $S' = \bigcup_{|s| \leq k} \texttt{addl}^s(S)$, and note that $V_k[D] = \pi_3(\texttt{prefix}_2(S'))$.

Let $D_{V_k[D]}$ be the extension of D by one unary predicate interpreted as $V_k[D]$. Since φ is safe, every element of every tuple in $\varphi(D)$ belongs to $V_k[D]$. We know that in order to evaluate $\varphi(\vec{x})$, it suffices to restrict quantification to the prefix-closure of $\mathit{adom}(D)$ and $\vec{x}$. Since $V_k[D]$ is prefix-closed, this implies that there is an active-domain query $\varphi'(\vec{x})$ over the schema extended with one unary symbol such that $\varphi'(D_{V_k[D]}) = \varphi(D)$ (here active-domain means that all quantification is restricted to the active domain, and that the output is only considered within the active domain of the input). By [10], φ' can be expressed by relational algebra extended with σ_α, for α ranging over $\mathbf{S}$ formulae. Since $D_{V_k[D]}$ is expressible in RA($\mathbf{S}$) and $\varphi(D) = \varphi'(D_{V_k[D]})$, we conclude that φ is expressible in RA($\mathbf{S}$).

The proof for $\mathbf{S}_{\text{len}}$ is almost identical, except that one defines $V_k[D]$ as $\{x \mid |x| \leq |y| + k, y \in \mathit{adom}(D)\}$, which is expressible in RA($\mathbf{S}_{\text{len}}$) using the $\texttt{addl}^s$ operations and the operations $\downarrow_i$.

The proof for $\mathbf{S}_{\text{reg}}$ is identical to the proof of for $\mathbf{S}$, as the set $V_k[D]$ is expressible in RA($\mathbf{S}_{\text{reg}}$). For $\mathbf{S}_{\text{left}}$, the proof again follows the same lines: all that is needed is that the set $N_p(\mathit{adom}(D))$ is expressible in RA($\mathbf{S}_{\text{left}}$) for a fixed p. But this follows from the fact that $\mathit{adom}(D)$ is definable in relational algebra, using $\texttt{prefix}$, $\texttt{addf}_i^a$ and $\texttt{trim}_i^a$ it is then possible to define $N_p(\mathit{adom}(D))$. The proof for $\mathbf{S}_{\text{reg,left}}$ follows from the expressibility of $V_k[D]$ and $N_p(\mathit{adom}(D))$. □

One of the operations in RA($\mathbf{S}_{\text{len}}$), $\downarrow_i$, is very expensive, as it may create sets whose size is exponential in the size of the input. This seems, however, unavoidable, as there are very expensive (e.g., NP-complete) safe queries in RC($\mathbf{S}_{\text{len}}$).

4.4.4 Deciding Safety Properties of Queries

Although query safety is undecidable for pure relational calculus (and hence for any extension), state-safety (given a query φ and a database D, is $\varphi(D)$ finite?) is decidable [64]. State safety is also known to be decidable for various extensions of the form RC($\mathcal{M}$) (for example, for the natural numbers with successor [64] or the real field [11]). For RC($\mathbf{S}$) and RC($\mathbf{S}_{\text{len}}$), this decidability holds as well:

4.27 Proposition *State-safety is decidable for* RC($\mathcal{M}$), *where* $\mathcal{M}$ *is one of* $\mathbf{S}, \mathbf{S}_{\text{left}}, \mathbf{S}_{\text{reg}}, \mathbf{S}_{\text{reg,left}}, \mathbf{S}_{\text{len}}$.

Proof. Given a query $\varphi(\vec{x})$ and a database D, we obtain a formula $\varphi'(\vec{x})$ by replacing each occurrence of a schema predicate $S(\vec{z})$ by a disjunction $\vec{z} = \vec{t}_1 \vee \ldots \vee \vec{z} = \vec{t}_m$ where $\{t_1, \ldots, t_m\}$ is the

interpretation of S in D. Since the formula $z = s$ is definable in all the structures for every fixed s, φ' can thus be viewed as a formula over $\mathbf{S}_{\text{len}}$ such that $\mathbf{S}_{\text{len}} \models \varphi'(\vec{x})$ iff $D \models \varphi(\vec{x})$. We now consider the sentence Φ defined as

$$\exists \vec{y}\ \forall \vec{x}\ (\varphi'(\vec{x}) \rightarrow \exists \vec{z}(\bigwedge_i z_i \prec y_i \wedge \mathrm{el}(z_i, x_i))).$$

Then $\varphi(D)$ is finite iff $\{\vec{a} \mid \mathbf{S}_{\text{len}} \models \varphi'(\vec{a})\}$ is finite iff $\mathbf{S}_{\text{len}} \models \Phi$, and thus the state-safety is decidable, since the theory of $\mathbf{S}_{\text{len}}$ is decidable. □

As query safety is undecidable, one often considers restrictions for which decidability can be obtained. Here we look at one of the most fundamental classes of queries – *conjunctive queries*. We take their definition in the context of interpreted operations from [11, 46]. A conjunctive query in $\mathrm{RC}(\mathcal{M})$ is a query of the form

$$\varphi(\vec{x}) \quad \equiv \quad \exists \vec{y}\ \bigwedge_{i=1}^{k} S_i(\vec{u}_i)\ \wedge\ \gamma(\vec{x}, \vec{y}),$$

where $k \geq 0$, each S_i is a schema relation, $\vec{u}_i$ is a subtuple of $(\vec{x}, \vec{y})$ of the same arity as S_i, and γ is an $\mathcal{M}$ formula. A Datalog-like notation for such a query would be $\varphi(\vec{x}) :- S_1(\vec{u}_1), \ldots, S_k(\vec{u}_k), \gamma(\vec{x}, \vec{y})$.

In [11], safety of conjunctive queries was shown decidable for $\mathrm{RC}(\mathcal{M})$, for various structures $\mathcal{M}$ on the reals with numerical operations. We now show a general result from which the decidability results for string structures $\mathbf{S}, \mathbf{S}_{\text{len}}$ as well as those considered in [11] follow. We say that *finiteness is definable with parameters* in $\mathcal{M}$ if for each formula $\psi(\vec{x}, \vec{y})$ in $\mathcal{M}$, there exists another formula $\psi_{\text{fin}}(\vec{y})$ such that $\mathcal{M} \models \psi_{\text{fin}}(\vec{a})$ iff the set $\{\vec{b} \mid \mathcal{M} \models \psi(\vec{b}, \vec{a})\}$ is finite. Furthermore, $\psi_{\text{fin}}(\vec{y})$ can be computed effectively.

4.28 Theorem *Assume that $\mathcal{M}$ can be expanded to $\mathcal{M}'$ such that the theory of $\mathcal{M}'$ is decidable, and finiteness is definable with parameters in $\mathcal{M}'$. Then safety of Boolean combinations of conjunctive queries in* $\mathrm{RC}(\mathcal{M})$ *is decidable.*

Proof. We start with a few easy observations about Boolean combinations of conjunctive queries in $\mathrm{RC}(\mathcal{M})$. First, if $\alpha(\vec{x})$ is a conjunctive query, it can be represented in the form $\exists \vec{z} \in adom \bigwedge_i S_i(\vec{u}_i) \wedge \gamma(\vec{x}, \vec{z})$. Indeed, given a query $\exists \vec{y} \bigwedge_i S_i(\vec{u}_i) \wedge \gamma'(\vec{x}, \vec{y})$, let $\vec{z}$ be the subtuple of $\vec{y}$ that consists of y_js appearing in the S_i atoms. Then the query can be rewritten to the one with active-domain quantification only, where $\gamma(\vec{x}, \vec{z}) \equiv \exists \vec{v} \gamma'(\vec{x}, \vec{y})$ – here $\vec{v}$ lists those variables in $\vec{y}$ that do not belong to $\vec{z}$. We also note that every conjunctive query is monotone.

Next, every Boolean combination of conjunctive queries is equivalent to a union of queries of the form $\alpha(\vec{x}) \wedge \neg\beta_1(\vec{x}) \wedge \ldots \wedge \neg\beta_k(\vec{x})$, where $k > 0$, and $\alpha, \beta_1, \ldots, \beta_k$ are conjunctive queries. Indeed, one puts a given Boolean combination in DNF, and observes that a conjunction of two conjunctive queries is a conjunctive query again, and since *true* and *false* are by definition conjunctive queries, we can assume that $k > 0$ and that one conjunctive query is present without negation.

Thus, we must show that it is decidable whether a query $q(\vec{x})$ of the form $\alpha(\vec{x}) \wedge \neg\beta_1(\vec{x}) \wedge \ldots \wedge \neg\beta_k(\vec{x})$ is safe. Let $\alpha(\vec{x})$ be $\exists \vec{z} \in adom \bigwedge_{i=1}^{l} S_i(\vec{u}_i) \wedge \gamma(\vec{x}, \vec{z})$.

We show the following claim: if there exists a database D such that $q(D)$ is infinite, then there exists a database D' with at most l tuples such $q(D')$ is finite. This in turn follows from the

Model	Data complexity	Data complexity of generic queries	Effective syntax for safe queries	Relational algebra	Safety of CQ
RC($\mathbf{S}$)	AC^0	FO(<)	yes	yes	decidable
RC($\mathbf{S}_{\mathrm{len}}$)	PH	AC^0	yes	yes	decidable
RC($\mathbf{S}_{\mathrm{left}}$)	AC^0	FO(<)	yes	yes	decidable
RC($\mathbf{S}_{\mathrm{reg}}$)	NC^1	FO(<)	yes	yes	decidable
RC($\mathbf{S}_{\mathrm{reg,left}}$)	NC^1	FO(<)	yes	yes	decidable
RC$_{concat}$	undecidable	undecidable	no	no	undecidable

Table 2: Summary of results on query languages

following: let $\mathcal{D}_l$ be the set of all databases D' with at most l tuples such that $D' \subseteq D$. Then $\alpha(D) = \cup_{D' \in \mathcal{D}_l} \alpha(D')$. Indeed, the $\supseteq$ inclusion follows from monotonicity, and the $\subseteq$ inclusion from the fact that to witness $\vec{a} \in \alpha(D)$, it suffices to find $\vec{b}$ such that $\bigwedge_{i=1}^{l} S_i(\vec{u}_i) \wedge \gamma(\vec{a}, \vec{b})$ holds; if such $\vec{b}$ exists, the l tuples $S_i(\vec{u}_i)$ form a database D' for which $\vec{a} \in \alpha(D')$.

Now, suppose $q(D)$ is infinite, and D has more than l tuples. We have $\alpha(D) = \bigcup_{D' \in \mathcal{D}_l} \alpha(D')$, and thus $q(D) = \bigcup_{D' \in \mathcal{D}_l} (\alpha(D') \cap \bigcap_i \neg\beta_i(D)) \subseteq \bigcup_{D' \in \mathcal{D}_l} (\alpha(D') \cap \bigcap_i \neg\beta_i(D'))$, since $\neg\beta_i$s are antimonotone. Since $q(D)$ is infinite, for some $D' \in \mathcal{D}_l$, $q(D') = \alpha(D') \cap \bigcap_i \neg\beta_i(D')$ is infinite. This proves the claim.

Let $\vec{\mathbf{t}}$ stand for $\vec{t}_1^1, \ldots, \vec{t}_l^1, \ldots, \vec{t}_1^p, \ldots, \vec{t}_l^p$, where p is the number of relation symbols in SC, and $\vec{t}_j^i$ is a tuple of variables of the same length as the arity of S_i. For a query q of the form $\alpha(\vec{x}) \wedge \neg\beta_1(\vec{x}) \wedge \ldots \wedge \neg\beta_k(\vec{x})$, let $q'(\vec{x}, \vec{\mathbf{t}})$ be the $\mathcal{M}$ formula obtained by replacing each $S_i(\vec{u})$ with $\bigvee_{j=1}^{l} \vec{u} = \vec{t}_j^i$. Then $\mathcal{M} \models q'(\vec{x}, \vec{\mathbf{t}})$ iff $D_{\vec{\mathbf{t}}} \models q(\vec{x})$, where $D_{\vec{\mathbf{t}}}$ is the database in which S_i is interpreted as $\{\vec{t}_1^i, \ldots, \vec{t}_l^i\}$. By the assumptions on $\mathcal{M}$, we know that in the expanded model we have a formula $q'_{\mathrm{fin}}(\vec{\mathbf{t}})$ such that $\mathcal{M}' \models q'_{\mathrm{fin}}(\vec{\mathbf{t}})$ iff the set of $\vec{x}$ such that $q'(\vec{x}, \vec{\mathbf{t}})$ holds is finite. In other words, it holds iff $q(D_{\vec{\mathbf{t}}})$ is finite. Hence, the sentence $\forall\vec{\mathbf{t}} q'_{\mathrm{fin}}(\vec{\mathbf{t}})$ is true in $\mathcal{M}$ iff $q(D)$ is finite for every database with at most l tuples, which by the previous claim means that q is safe. The decidability of the theory of $\mathcal{M}'$ now implies the decidability of the safety of q. The theorem is proved. □

We know that Th($\mathbf{S}_{\mathrm{len}}$) is decidable [14]. Moreover, finiteness is definable with parameters: for $\psi(\vec{x}, \vec{y})$, $\psi_{\mathrm{fin}}(\vec{y})$ is $\exists\vec{u}(\forall\vec{x}\psi(\vec{x}, \vec{y}) \rightarrow \exists\vec{z} \bigwedge_i z_i \prec u_i \ \mathrm{el}(z_i, x_i))$. Thus:

4.29 Corollary *The safety of Boolean combinations of conjunctive queries in* RC($\mathbf{S}$), RC($\mathbf{S}_{\mathrm{left}}$), RC($\mathbf{S}_{\mathrm{len}}$), RC($\mathbf{S}_{\mathrm{reg}}$) *and* RC($\mathbf{S}_{\mathrm{reg,left}}$) *is decidable.*

Table 2 summarizes the results of the section.

5 Conclusion

There has been significant interest in theoretical computer science in understanding the structure of the regular languages, and in identifying subclasses of the regular languages that have special properties [67, 65]. Our work can be seen as an extension of this program, where we consider subclasses of the regular n-ary relations rather than the regular sets. In our approach, however, we

do not focus on properties that hold of one particular regular relation by itself, but rather look at some desirable properties of a whole algebra of relations within the structure $\mathbf{S}_{\text{len}}$.

We have shown a sharp contrast between the behavior of the full algebra of regular relations of $\mathbf{S}_{\text{len}}$, and those of various submodels such as $\mathbf{S}$, $\mathbf{S}_{\text{left}}$, $\mathbf{S}_{\text{reg}}$, and $\mathbf{S}_{\text{reg,left}}$. We show that the latter are more tractable in many respects. Furthermore, we show that the behavior of an algebra of relations is not at all determined by the one-dimensional sets (subsets of Σ^*) in the algebra: for example, one can have fairly complex binary relations definable, yet still maintain the property that all definable subsets of Σ^* are star-free. Figure 1 summarizes the relationships between the star-free and regular algebras we considered here.

We have also studied extensions of the standard relational calculus with various sets of string operations. We were interested in languages that were not computationally complete, but rather shared the attractive complexity-theoretic and static analysis properties of relational calculus.

The language $\text{RC}(\mathbf{S})$ can be seen as a nice foundation over which other languages should be built. It covers the most rudimentary string operations, but its expressive power is quite limited. The extension $\text{RC}(\mathbf{S}_{\text{len}})$ is too powerful (but still not computationally complete). We therefore considered the languages in between – $\text{RC}(\mathbf{S}_{\text{left}}), \text{RC}(\mathbf{S}_{\text{reg}}), \text{RC}(\mathbf{S}_{\text{reg,left}})$ – that can express some important operations found in $\text{RC}(\mathbf{S}_{\text{len}})$, but still have low data complexity. All the calculi have effective syntax for safe queries, and corresponding relational algebras.

A key question is how many relations one can add to the models $\mathbf{S}_{\text{left}}$ or $\mathbf{S}_{\text{reg}}$ and still have the attractive properties like QE, finite VC-dimension, and a nicely-behaved relational calculus. Is there a model that is somehow maximal with respect to these properties? We would very much like to know the answer to this question. There are also several natural candidate models that would seem amenable to the approach taken here, and where one would expect the same results to go through: for example, if one allows the operation of concatenating a fixed sequence "in the middle" of a string, rather than on the left or on the right, is the resulting model still tractable?

Acknowledgments We thank Wolfgang Thomas, Scott Weinstein, Emmanuel Waller, and Jan Van den Bussche for fruitful discussions on the subject, and the anonymous referees for numerous helpful comments.

References

[1] S. Abiteboul, R. Hull and V. Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[2] M. Ajtai. Σ^1_1 formulae on finite structures. *Annals of Pure and Applied Logic*, 24 (1983), 1–48.

[3] M. Ajtai, R. Fagin and L. Stockmeyer. The closure of monadic NP. *JCSS* 60(3): 660–716 (2000).

[4] D. Angluin, D. N. Hoover. Regular prefix relations. *Mathematical Systems Theory* 17(3),167–191,1984.

[5] M. Anthony and N. Biggs. *Computational Learning Theory.* Cambridge Univ. Press, 1992.

[6] A. Atserias, Ph. Kolaitis. First-order logic vs. fixed-point logic in finite set theory. In *LICS'98*, pages 275–284.

[7] D.A. Barrington, N. Immerman, H. Straubing. On uniformity within NC^1. *JCSS*, 41:274–306,1990.

[8] O. Belegradek, A. Stolboushkin, M. Taitslin. Extended order-generic queries. *Annals of Pure and Applied Logic* 97 (1999), 85–125.

[9] M. Benedikt, G. Dong, L. Libkin, L. Wong. Relational expressive power of constraint query languages. *Journal of the ACM* 45 (1998), 1–34.

[10] M. Benedikt, L. Libkin. Relational queries over interpreted structures. *Journal of the ACM* 47 (2000), 644–680.

[11] M. Benedikt, L. Libkin. Safe constraint queries. *SIAM J. Comput.* 29 (2000), 1652–1682.

[12] M. Benedikt, L. Libkin, T. Schwentick, L. Segoufin. A model-theoretic approach to regular string relations. In *LICS'01*, pages 431–440.

[13] M. Benedikt, L. Libkin, T. Schwentick, L. Segoufin. String operations in query languages. In *PODS'01*, pages 183–194.

[14] A. Blumensath and E. Grädel. Automatic structures. In *LICS'00*, pages 51–62.

[15] A. Blumer, A. Ehrenfeucht, D. Haussler, M. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM* 36 (1989), 929–965.

[16] N. Bjørner. Integration of Decision Procedures in Temporal Verification. PhD Thesis, Stanford University, 2000.

[17] A. Bonner and G. Mecca. Sequences, datalog, and transducers. *JCSS* 57 (1998), 234–259.

[18] A. Bonner and G. Mecca. Querying string databases with transducers. In *DBPL'97*, pages 118–135.

[19] V. Bruyère, G. Hansel, C. Michaux, R. Villemaire. Logic and p-recognizable sets of integers. *Bull. Belg. Math. Soc.* 1 (1994), 191–238.

[20] J.R. Büchi. Weak second-order arithmetic and finite automata. *Zeit. Math. Logik Grundl. Math.* 6 (1960), 66–92.

[21] C.C. Chang and H.J. Keisler Model Theory. North Holland, 1990.

[22] G. Cherlin and F. Point. On extensions of Presburger arithmetic. In *Proc. 4th Easter Model Theory Conf.*, Humboldt Univ. Berlin, 1986.

[23] H. Comon, R. Treinen. The first-order theory of lexicographic path orderings is undecidable. *TCS* 176 (1997), 67–87.

[24] M. Consens and T. Milo. Algebras for querying text regions: expressive power and optimization. *JCSS* 57 (1998), 272–288.

[25] E. Dantsin, A. Voronkov. Expressive power and data complexity of query languages for trees and lists. In *PODS'2000*, pages 157–165.

[26] L. Denenberg, Y. Gurevich and S. Shelah. Definability by constant-depth polynomial-size circuits. *Information and Control* 70 (1986), 216–240.

[27] H.-D. Ebbinghaus and J. Flum. *Finite Model Theory.* Springer Verlag, 1995.

[28] A. Ehrenfeucht. An application of games to the completeness problem for formalized theories. *Fund. Math.*, 49:129–141, 1961.

[29] C. Elgot and J. Mezei. On relations defined by generalized finite automata. *IBM J. Res. Develop.* 9 (1965), 47–68.

[30] D. Epstein et al. *Word Processing in Groups.* Jones and Bartlett Publ., 1992.

[31] M. Fischer and M. Rabin. Super-exponential complexity of Presburger arithmetic. SIAM-AMS Proceedings 7, 27–41 (1974).

[32] J. Flum and M. Ziegler. Pseudo-finite homogeneity and saturation. *J. Symb. Logic* 64 (1999), 1689–1699.

[33] R. Fraïssé. Sur quelques classifications des systèmes de relations. *Publ. Sci. Univ. Alger. Sér. A*, 1:35–182, 1954.

[34] C. Frougny and J. Sakarovitch. Synchronized rational relations of finite and infinite words. *TCS* 108 (1993), 45–82.

[35] M. Furst, J. Saxe, M. Sipser. Parity, circuits, and the polynomial-time hierarchy. *Math. Systems Theory* 17 (1984), 13–27.

[36] S. Ginsburg and X.S. Wang. Pattern matching by rs-operations: toward a unified approach to querying sequenced data. In *PODS'92*, pages 293–300.

[37] E. Grädel and Y. Gurevich. Metafinite model theory. *Information and Computation* 140 (1998), 26–81.

[38] G. Grahne and M. Nykänen. Safety, translation and evaluation of alignment calculus. In *Proc. of the First East-European Symp. on Advances in Databases and Information Systems (ADBIS'97)*, 295–304, 1997.

[39] G. Grahne, M. Nykänen, E. Ukkonen. Reasoning about strings in databases. *JCSS* 59 (1999), 116–162.

[40] G. Grahne, E. Waller. How to make SQL stand for string query language. In *Proceedings of DBPL'99*, Springer LNCS vol. 1949, 2000, pages 61–79.

[41] P. Gulutzan and S. Pelzer. *SQL-99 Complete, Really.* R&D Books, 1999.

[42] R. Hakli, M. Nykänen, H. Tamm, and E. Ukkonen. Implementing a declarative string query language with string restructuring. In *PADL'99*, pages 179–195.

[43] D. Harel. Towards a theory of recursive structures. In *MFCS'98*, pages 36–53.

[44] M. Hodges. Model Theory. Cambridge, 1993.

[45] B. Hodgson. Décidabilité par automate fini. *Ann. Sc. Math. Québec* 7(1) (1983), 39–57.

[46] O. Ibarra, J. Su. A technique for proving decidability of containment and equivalence of linear constraint queries. *JCSS* 59 (1999), 1–28.

[47] N. Immerman. *Descriptive Complexity.* Springer, 1999.

[48] B. Khoussainov and A. Nerode. Automatic presentations of structures. In *LCC'94*, pages 367–392.

[49] Ph. Kolaitis and M. Vardi. Fixpoint logic vs. infinitary logic in finite-model theory. In *LICS'92*, pages 46–57.

[50] G. Kuper, L. Libkin, J. Paredaens, editors. *Constraint Databases.* Springer, 2000.

[51] M. C. Laskowski. Vapnik-Chervonenkis classes of definable sets. *J. London Math. Soc.*, 45:377–384, 1992.

[52] H. Läuchli and C. Savioz. Monadic second order definable relations on the binary tree. *J. Symb. Logic* 51(1) (1987), 219–226.

[53] A. Malćev. On the elementary theories of locally free universal algebras. *Soviet Math. Doklady* 2 (1961), 768–771.

[54] R. McNaughton and S. Papert. *Counter-Free Automata.* MIT Press, 1971.

[55] C. Michaux, R. Villemaire. Open questions around Büchi and Presburger arithmetics. In *Logic: From Foundations to Applications*, Oxford Univ. Press, 1996, pages 353–383.

[56] F. Neven and J. Van den Bussche. Expressiveness of structured document query languages based on attribute grammars. *Journal of the ACM*, 49(1): 56–100 (2002).

[57] C. H. Papadimitriou. *Computational Complexity* Addison-Wesley, 1994.

[58] M. Rabin. Weakly definable relations and special automata. In *Mathematical Logic and Foundations of Set Theory*, North Holland, Amsterdam, 1970, pages 1–23.

[59] A. Rajasekar. String-oriented databases. *SPIRE/CRIWG'99*, pages 158–167.

[60] T. Rybina, A. Voronkov. A decision procedure for term algebras with queues. *ACM TOCL* 2(2): 155–181 (2001).

[61] A. Salomaa. *Formal Languages.* Academic Press, 1973.

[62] T. Schwentick. On diving in trees. In *Proceedings of the 23rd Symposium on Mathematical Foundations of Computer Science (MFCS 2000), Bratislava, 2000*, pages 660–669, 2000.

[63] S. Shelah. Stability, the f.c.p., and superstability. *Ann. of Math. Logic* 3 (1971), 271–362.

[64] A. Stolboushkin, M. Taitslin. Finite queries do not have effective syntax. *Information and Computation*, 153(1) (1999), 99–116.

[65] H. Straubing. *Finite Automata, Formal Logic, and Circuit Complexity.* Birkhäuser, 1994.

[66] W. Thomas. Infinite trees and automaton-definable relations over ω-words. *TCS* 103 (1992), 143–159.

[67] W. Thomas. Languages, automata, and logic. *Handbook of Formal Languages, Vol. 3*, Springer, 1997.

[68] K. Venkataraman. Decidability of the purely existential fragment of the theory of term algebras. *Journal of the ACM* 34 (1987), 492–510.