Parameterized Regular Expressions and Their Languages

Pablo Barceló[1], Leonid Libkin[2], and Juan L. Reutter[2]

1 **Department of Computer Science,**
University of Chile
pbarcelo@dcc.uchile.cl
2 **School of Informatics,**
University of Edinburgh
libkin@inf.ed.ac.uk, juan.reutter@ed.ac.uk

Abstract

We study regular expressions that use variables, or parameters, which are interpreted as alphabet letters. We consider two classes of languages denoted by such expressions: under the possibility semantics, a word belongs to the language if it is denoted by some regular expression obtained by replacing variables with letters; under the certainty semantics, the word must be denoted by every such expression. Such languages are regular, and we show that they naturally arise in several applications such as querying graph databases and program analysis. As the main contribution of the paper, we provide a complete characterization of the complexity of the main computational problems related to such languages: nonemptiness, universality, containment, membership, as well as the problem of constructing NFAs capturing such languages. We also look at the extension when domains of variables could be arbitrary regular languages, and show that under the certainty semantics, languages remain regular and the complexity of the main computational problems does not change.

1998 ACM Subject Classification F.4.3 Formal Languages.

Keywords and phrases Regular expressions, complexity, decision problems, regular languages.

1 Introduction

In this paper we study parameterized regular expressions like $(0x)^*1(xy)^*$ that combine letters from a finite alphabet Σ, such as 0 and 1, and variables, such as x and y. These variables are interpreted as letters from Σ. This gives two ways of defining the language of words over Σ denoted by a parameterized regular expression e. Under the first – possibility – semantics, a word $w \in \Sigma^*$ is in the language $\mathcal{L}_\diamond(e)$ if w is in the language of *some* regular expression e' obtained by substituting alphabet letters for variables. Under the second – certainty – semantics, $w \in \mathcal{L}_\Box(e)$ if w is in the language of *all* regular expressions obtained by substituting alphabet letters for variables. For example, if $e = (0x)^*1(xy)^*$, then $01110 \in \mathcal{L}_\diamond(e)$, as witnessed by the substitution $x \mapsto 1, y \mapsto 0$. The word 1 is in $\mathcal{L}_\Box(e)$, since the starred subexpressions can be replaced by the empty word. As a more involved example of the certainty semantics, the reader can verify that for $e' = (0|1)^*xy(0|1)^*$, the word 10011 is in $\mathcal{L}_\Box(e')$, although no word of length less than 5 can be in $\mathcal{L}_\Box(e')$.

These semantics of parameterized regular expressions arise in a variety of applications, in particular in the fields of querying graph-structured data, and static analysis of programs. We now explain these connections.

Applications in graph databases Graph databases, that describe both data and its topology, have been actively studied over the past few years in connection with such diverse

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

topics as social networks, biological data, semantic Web and RDF, crime detection and analyzing network traffic; see [3] for a survey. The abstract data model is essentially an edge-labeled graph, with edge labels coming from a finite alphabet. This finite alphabet can contain, for example, types of relationships in a social network or a list of RDF properties. In this setting one concentrates on various types of reachability queries, e.g., queries that ask for the existence of a path between nodes with certain properties so that the label of the path forms a word in a given regular language [4, 7, 8, 10]. Note that in this setting of querying topology of a graph database, it is standard to use a finite alphabet for labeling [3].

As in most data management applications, it is common that some information is missing, typically due to using data that is the result of another query or transformation [1, 5, 9]. For example, in a social network we may have edges $a \xmapsto{x} b$ and $a' \xmapsto{x} b'$, saying that the relationship between a and b is the same as that between a' and b'. However, the precise nature of such a relationship is unknown, and this is represented by a variable x. Such graphs G whose edges are labeled by letters from Σ and variables from a set $\mathcal{W}$ can be viewed as automata over $\Sigma \cup \mathcal{W}$. In checking the existence of paths between nodes, one normally looks for *certain answers* [16], i.e., answers independent of a particular interpretation of variables.

In the case of graph databases such certain answers can be found as follows. Let a, b be two nodes of G. One can view (G, a, b) as an automaton, with a as the initial state, and b as the final state; its language, over $\Sigma \cup \mathcal{W}$ is given by some regular expression $e(G, a, b)$. Then we can be certain about the existence of a word w from some language L that is the label of a path from a to b iff w also belongs to $\mathcal{L}_\Box(e(G, a, b))$, i.e., iff $L \cap \mathcal{L}_\Box(e(G, a, b))$ is nonempty. Hence, computing $\mathcal{L}_\Box(e)$ is essential for answering queries over graph databases with missing information.

Applications in program analysis That regular expressions with variables appear naturally in program analysis tasks was noticed, for instance, in [20, 21, 23]. One uses the alphabet that consists of symbols related to operations on variables, pointers, or files, e.g., `def` for defining a variable, `use` for using it, `open` for opening a file, or `malloc` for allocating a pointer. A variable then follows: $\texttt{def}(x)$ means defining variable x. While variables and alphabet symbols do not mix freely any more, it is easy to enforce correct syntax with an automaton. An example of a regular condition with parameters is searching for uninitialized variables: $(\neg\texttt{def}(x))^*\texttt{use}(x)$.

Expressions like this are evaluated on a graph that serves as an abstraction of a program. One considers two evaluation problems: whether under some evaluation of variables, either some path, or every path between two nodes satisfies it. This amounts to computing $\mathcal{L}_\Diamond(e)$ and checking whether all paths, or some path between nodes is in that language. In case of uninitialized variables one would be using 'some path' semantics; the need for the 'all paths' semantics arises when one analyzes locking disciplines or constant folding optimizations [20, 23]. So in this case the language of interest for us is $\mathcal{L}_\Diamond(e)$, as one wants to check whether there is an evaluation of variables for which some property of a program is true.

Parameterized regular expressions appeared in other applications as well, e.g., in phase-sequence prediction for dynamic memory allocation [25], or as a compact way to express a family of legal behaviors in hardware verification [6], or as a tool to state regular constraints in constraint satisfaction problems [24].

At the same time, however, very little is known about the basic properties of the languages $\mathcal{L}_\Box(e)$ and $\mathcal{L}_\Diamond(e)$. Thus, our main goal is to determine the exact complexity of the key problems related to languages $\mathcal{L}_\Box(e)$ and $\mathcal{L}_\Diamond(e)$. We consider the standard language-theoretic decision problems, such as membership of a word in the language, language nonemptiness, universality, and containment. Since the languages $\mathcal{L}_\Box(e)$ and $\mathcal{L}_\Diamond(e)$ are regular, we also consider the complexity of constructing NFAs, over the finite alphabet Σ, that define them.

For all the decision problems, we determine their complexity. In fact, all of them are complete for various complexity classes, from NLogspace to Expspace. We establish upper bounds on the running time of algorithms for constructing NFAs, and then prove matching lower bounds for the sizes of NFAs representing $\mathcal{L}_\Box(e)$ and $\mathcal{L}_\Diamond(e)$. Finally, we look at extensions where the range of variables need not be just Σ. Under the possibility semantics, such languages need not be regular, but under the certainty semantics, we prove regularity and establish complexity bounds.

Related work There are several related papers on the possibility semantics, notably [11, 14, 18]. Unlike the investigation in this paper, [14, 18] concentrated on the $\mathcal{L}_\Diamond(e)$ semantics in the context of *infinite* alphabets. The motivation of [14] comes from the study of infinite-state systems with finite control (e.g., software with integer parameters). In contrast, for the applications outlined in the introduction, finite alphabets are more appropriate [3, 8, 20, 21]. Results in [14] show that under the possibility semantics and infinite alphabets, the resulting languages can also accepted by non-deterministic register automata [18], and both closure and decidability become problematic. For example, universality and containment are undecidable over infinite alphabets [14]. In contrast, in the classical language-theoretic framework of finite alphabets, closure and decidability are guaranteed, and the key questions are related to the precise complexity of the main decision problems, with most of them requiring new proof techniques.

An analog of the $\mathcal{L}_\Box$ semantics was studied in the context of graph databases in [5]. The model used there is more complex than the simple model of parameterized regular expressions. Essentially, it boils down to automata in which transitions can be labeled with such parameterized expressions, and labels can be shared between different transitions. Motivations for this model come from different ways of incorporating incompleteness into the graph database model. Due to the added complexity, lower bounds for the model of [5] do not extend automatically to parameterized regular expressions, and in the cases when complexity bounds happen to be the same, new proofs are required.

Different forms of succinct representations of regular languages, for instance with squaring, complement, and intersection, are known in the literature, and both decision problems [22] and algorithmic problems [12] have been investigated for them. However, it appears that parameterized regular expressions cannot be used to succinctly define an arbitrary regular expression, nor any arbitrary union or intersection of them. Thus, the study of these expressions requires the development of new tools for understanding the lower bounds of their decision problems.

When we let variables range over words rather than letters, under the possibility semantics $\mathcal{L}_\Diamond$ we may obtain, for example, pattern languages [17] or languages given by expressions with backreferences [2]. These languages need not be regular, and some of the problems (e.g., universality for backreferences) are undecidable [11]. In contrast, we show that under the certainly semantics $\mathcal{L}_\Box$ regularity is preserved, and complexity is similar to the case of variables ranging over letters.

Organization Parameterized regular expressions and their languages are formally defined in Section 2. In Section 3 we define the main problems we study. Complexity of the main decision problems is analyzed in Section 4, and complexity of automata construction in Section 5. In Section 6 we study extensions when domains of variables need not be single letters.

2 Preliminaries

Let Σ be a finite alphabet, and $\mathcal{V}$ a countably infinite set of variables, disjoint from Σ. Regular expressions over $\Sigma \cup \mathcal{V}$ will be called *parameterized regular expressions.* Regular expressions, as usual, are built from $\emptyset$, the empty word ε, symbols in Σ and $\mathcal{V}$, by operations of concatenation $(\cdot)$, union $(|)$, and the Kleene star $(*)$. Of course each such expression only uses finitely many symbols in $\mathcal{V}$. The size of a regular expression is measured as the total number of symbols needed to write it down (or as the size of its parse tree).

We write $\mathcal{L}(e)$ for the language defined by a regular expression e. If e is a parameterized regular expression that uses variables from a finite set $\mathcal{W} \subset \mathcal{V}$, then $\mathcal{L}(e) \subseteq (\Sigma \cup \mathcal{W})^*$. We are interested in languages $\mathcal{L}_\Box(e)$ and $\mathcal{L}_\Diamond(e)$, which are subsets of Σ^*. To define them, we need the notion of a valuation ν which is a mapping from $\mathcal{W}$ to Σ, where $\mathcal{W}$ is the set of variables mentioned in e. By $\nu(e)$ we mean the regular expression over Σ obtained from e by simultaneously replacing each variable $x \in \mathcal{W}$ by $\nu(x)$. For example, if $e = (0x)^*1(xy)^*$ and ν is given by $x \mapsto 1, y \mapsto 0$, then $\nu(e) = (01)^*1(10)^*$.

We now formally define the certainty and possibility semantics for parameterized regular expressions.

▶ **Definition 1** (Acceptance). Let e be a parameterized regular expression. Then:

- $\mathcal{L}_\Box(e) := \bigcap\{\mathcal{L}(\nu(e)) \mid \nu \text{ is a valuation for } e\}$ (certainty semantics)
- $\mathcal{L}_\Diamond(e) := \bigcup\{\mathcal{L}(\nu(e)) \mid \nu \text{ is a valuation for } e\}$ (possibility semantics).

Since each parameterized regular expression uses finitely many variables, the number of possible valuations is finite as well, and thus both $\mathcal{L}_\Box(e)$ and $\mathcal{L}_\Diamond(e)$ are regular languages over Σ^*.

The usual connection between regular expressions and automata extends to the parameterized case. Each parameterized regular expression e over $\Sigma \cup \mathcal{W}$, where $\mathcal{W}$ is a finite set of variables in $\mathcal{V}$, can of course be translated, in polynomial time, into an NFA $\mathcal{A}_e$ over $\Sigma \cup \mathcal{W}$ such that $\mathcal{L}(\mathcal{A}_e) = \mathcal{L}(e)$. Such equivalences extend to $\mathcal{L}_\Box$ and $\mathcal{L}_\Diamond$. Namely, for an NFA $\mathcal{A}$ over $\Sigma \cup \mathcal{W}$, and a valuation $\nu : \mathcal{W} \to \Sigma$, define $\nu(\mathcal{A})$ as the NFA over Σ that is obtained from $\mathcal{A}$ by replacing each transition of the form (q, x, q') in $\mathcal{A}$ (for q, q' states of $\mathcal{A}$ and $x \in \mathcal{W}$) with the transition $(q, \nu(x), q')$. The following is just an easy observation:

▶ **Lemma 2.** *Let e be a parameterized regular expression, and $\mathcal{A}_e$ be an NFA over $\Sigma \cup \mathcal{V}$ such that $\mathcal{L}(\mathcal{A}_e) = \mathcal{L}(e)$. Then, for every valuation ν, we have $\mathcal{L}(\nu(e)) = \mathcal{L}(\nu(\mathcal{A}_e))$.*

Hence, if we define $\mathcal{L}_\Box(\mathcal{A})$ as $\bigcap_\nu \mathcal{L}(\nu(\mathcal{A}))$, and $\mathcal{L}_\Diamond(\mathcal{A})$ as $\bigcup_\nu \mathcal{L}(\nu(\mathcal{A}))$, then the lemma implies that $\mathcal{L}_\Box(e) = \mathcal{L}_\Box(\mathcal{A}_e)$ and $\mathcal{L}_\Diamond(e) = \mathcal{L}_\Diamond(\mathcal{A}_e)$. Since one can go from regular expressions to NFAs in polynomial time, this will allow us to use both automata and regular expressions interchangeably to establish our results.

3 Basic Problems

We now describe the main problems we study here. For each problem we shall have two versions, depending on which semantics – $\mathcal{L}_\Box$ or $\mathcal{L}_\Diamond$ is used. So each problem will have a subscript $*$ that can be interpreted as $\Box$ or $\Diamond$.

We start with decision problems:

NONEMPTINESS$_*$ Given a parameterized regular expression e, is $\mathcal{L}_*(e) \neq \emptyset$?

MEMBERSHIP$_*$ Given a parameterized regular expression e and a word $w \in \Sigma^*$, is $w \in \mathcal{L}_*(e)$?

UNIVERSALITY$_*$ Given a parameterized regular expression e, is $\mathcal{L}_*(e) = \Sigma^*$?
CONTAINMENT$_*$ Given parameterized regular expressions e_1 and e_2, is $\mathcal{L}_*(e_1) \subseteq \mathcal{L}_*(e_2)$?

A special version of nonemptiness is the problem of intersection with a regular language (used in the database querying example in the introduction):

NONEMPTYINTREG$_*$ Given a parameterized regular expression e, and a regular expression e' over Σ, is $\mathcal{L}(e') \cap \mathcal{L}_*(e) \neq \emptyset$?

The last problem is computational rather than a decision problem:

CONSTRUCTNFA$_*$ Given a parameterized regular expression e, construct an NFA $\mathcal{A}$ over Σ such that $\mathcal{L}_*(e) = \mathcal{L}(\mathcal{A})$.

4 Decision problems

In this section we consider the five decision problems – nonemptiness, membership, universality and containment – and provide precise complexity for them.

Restrictions on regular expressions We shall also consider two restrictions on regular expressions; these will indicate when the problems are inherently very hard or when their complexity can be lowered in some cases. One source of complexity is the repetition of variables in expressions like $(0x)^*1(xy)^*$. When no variable appears more than once in a parameterized regular expression, we call it *simple*. Another source of complexity is infinite languages, so we consider a restriction to expressions of *star-height* 0, in which no Kleene star is used: these denote finite languages, and each finite language is denoted by such an expression.

4.1 Nonemptiness

The problem NONEMPTINESS$_\diamond$ has a trivial solution, since $\mathcal{L}_\diamond(e) \neq \emptyset$ for every parameterized regular expression e (except $e = \emptyset$). So we study this problem for the certainty semantics; for the possibility semantics, we look at the related problem NONEMPTINESS-AUTOMATA$_\diamond$, which, for a given NFA $\mathcal{A}$ over $\Sigma \cup \mathcal{V}$ asks whether $\mathcal{L}_\diamond(\mathcal{A}) \neq \emptyset$.

▶ **Theorem 3.**
- *The problem* NONEMPTINESS$_\Box$ *is* EXPSPACE-*complete.*
- *The problem* NONEMPTINESS-AUTOMATA$_\diamond$ *is* NLOGSPACE-*complete.*

The result for the possibility semantics is by a standard reachability argument. Note that the bound is the same here as in the case of infinite alphabets studied in [14]. To see the upper bound for NONEMPTINESS$_\Box$, note that there are exponentially many valuations ν, and each automaton $\nu(\mathcal{A}_e)$ is of polynomial size, so we can use the standard algorithm for checking nonemptiness of the intersection of a family of regular languages which can be solved in polynomial space in terms of the size of its input; since the input to this problem is of exponential size in terms of the original input, the EXPSPACE bound follows. The hardness is by a generic (Turing machine) reduction.

In the proof we use the following property of the certainty semantics, which shows a striking difference with the case of standard regular expressions:

▶ **Lemma 4.** *Given a set $e_1, \ldots, e_k$ of parameterized expressions of size at most $n \geq k$, it is possible to build, in time $O(k \cdot n)$ an expression e' such that $\mathcal{L}_\Box(e')$ is empty if and only if $\mathcal{L}_\Box(e_1) \cap \cdots \cap \mathcal{L}_\Box(e_k)$ is empty.*

The reason the case of the $\mathcal{L}_\Box(e)$ semantics is so different from the usual semantics of regular languages is as follows. It is well known that checking whether the intersection of the languages defined by a finite set S of regular expressions is nonempty is PSPACE-complete [19], and hence under widely held complexity-theoretical assumptions no regular expression r can be constructed in polynomial time from S such that $\mathcal{L}(r)$ is nonempty if and only if $\bigcap_{s \in S} \mathcal{L}(s)$ is nonempty. Lemma 4, on the other hand, says that such a construction is possible for parameterized regular expressions under the certainty semantics.

The generic reduction used in the proof of EXPSPACE-hardness of NONEMPTINESS$_\Box$ also provides lower bounds on the minimal sizes of words in languages $\mathcal{L}_\Box(e)$ (note that the language $\mathcal{L}_\Diamond(e)$ always contains a word of the size linear in the size of e).

▶ **Corollary 5.** *There exists a polynomial $p : \mathbb{N} \to \mathbb{N}$ and a sequence of parameterized regular expressions $\{e_n\}_{n \in \mathbb{N}}$ such that each e_n is of size at most $p(n)$, and every word in the language $\mathcal{L}_\Box(e_n)$ has size at least 2^{2^n}.*

The construction is somewhat involved, but it is easy to see the single-exponential bound (which was hinted at in the first paragraph of the introduction, and which was in fact used in connection with querying incomplete graph data in [5]). For each n, consider an expression $e_n = (0|1)^* x_1 \dots x_n (0|1)^*$. If a word w is in $\mathcal{L}_\Box(e_n)$, then w must contain every word in $\{0,1\}^n$ as a subword, which implies that its length must be at least $2^n + 1$.

We can also show that the use of Kleene star has a huge impact on complexity, which is not at the same time affected by variable repetitions.

▶ **Proposition 6.** *The problem* NONEMPTINESS$_\Box$ *remains* EXPSPACE*-hard over the class of simple regular expressions, but it is Σ_2^p-complete over the class of expressions of star-height 0.*

4.2 Membership

It is easy to see that MEMBERSHIP$_\Box$ can be solved in coNP, and MEMBERSHIP$_\Diamond$ in NP: one just guesses a valuation witnessing $w \in \mathcal{L}(v(e))$ or $w \notin \mathcal{L}(v(e))$. These bounds turn out to be tight.

▶ **Theorem 7.**
- *The problem* MEMBERSHIP$_\Box$ *is* coNP*-complete.*
- *The problem* MEMBERSHIP$_\Diamond$ *is* NP*-complete.*

Proof sketch: We only sketch the proof of NP-hardness (which also works for simple expressions). We use a reduction from 3-SAT. Let $\varphi = \bigwedge_{1 \leq j \leq n} (\ell_j^1 \vee \ell_j^2 \vee \ell_j^3)$ be a 3-CNF propositional formula over variables $\{p_1, \dots, p_m\}$. That is, each literal ℓ_j^k, for $1 \leq j \leq n$ and $1 \leq k \leq 3$, is either p_i or $\neg p_i$, for some $i \leq m$. From φ we construct, in polynomial time, a simple parameterized regular expression e over alphabet $\Sigma = \{a, b, c, d, 0, 1\}$ and variables $x_i, \bar{x}_i$, for $1 \leq i \leq m$, and a word w over Σ such that φ is satisfiable if and only if $w \in \mathcal{L}_\Diamond(e)$.

The regular expression e is defined as f^*, where $f := a(f_1|g_1|\dots|f_m|g_m)b$, and the regular expressions f_i and g_i are defined as follows. Intuitively, f_i (resp. g_i) codes the clauses in which p_i occurs positively (resp. negatively). Let $j_1, \dots, j_r$ enumerate the clauses where the variable p_i appears positively. The expression f_i is defined as

$$f_i \;=\; (c^i \mid d^{j_1} \mid \dots \mid d^{j_r}) \cdot x_i.$$

The expression g_i is defined similarly except using indices of clauses where p_i occurs negatively, and the variable $\bar{x}_i$ in place of x_i. Note that e is a simple expression and can be constructed in polynomial time from φ.

The word w is $ac1b\,ac0b\,acc1b\,acc0b\ldots ac^m1b\,ac^m0b\,ad1b\,add1b\ldots ad^n1b$; it too can be constructed in polynomial time from φ. It is now not hard to prove that φ is satisfiable if and only if $w \in \mathcal{L}_\Diamond(e)$. □

Note that for the case of the possibility semantics, the bound is the same as for languages over the infinite alphabets [14] (for all problems other than nonemptiness and membership, the bounds will be different). The hardness proof in [14] relies on the infinite size of the alphabet, but one can find an alternative proof that uses only finitely many symbols. Both proofs are by variations of 3-SAT or its complement.

The restrictions to expressions without repetitions, or to finite languages, by themselves do not lower the complexity, but together they make it polynomial.

▶ **Proposition 8.** *The complexity of the membership problem remains as in Theorem 7 over the classes of simple expressions, and expressions of star-height* 0*. Over the class of simple expressions of star-height* 0*,* Membership$_\Diamond$ *can be solved in polynomial time (actually, in time* $O(nm\log^2 n)$*, where* n *is the size of the expression and* m *is the size of the word).*

The $\log^2 n$ factor appears due to the complexity of the algorithm for converting regular expressions into ε-free NFAs [15].

Membership for fixed words We next consider a variation of the membership problem: Membership$_*(w)$ asks, for a parameterized regular expression e, whether $w \in \mathcal{L}_*(e)$. In other words, w is fixed. It turns out that for the $\Diamond$-semantics, this version is efficiently solvable, but for the $\Box$-semantics, it remains intractable unless restricted to simple expressions.

▶ **Theorem 9.**
- *There is a word* $w \in \Sigma^*$ *such that the problem* Membership$_\Box(w)$ *is* coNP*-hard (even over the class of expressions of star-height* 0*).*
- *For each word* $w \in \Sigma^*$*, the problem* Membership$_\Box(w)$ *is solvable in linear time, if restricted to the class of simple expressions.*
- *For each word* $w \in \Sigma^*$*, the problem* Membership$_\Diamond(w)$ *is solvable in time* $O(n\log^2 n)$*.*

On the other hand, it is straightforward to show that the membership problem for fixed expressions can be solved efficiently for both semantics.

4.3 Universality

Somewhat curiously, the universality problem is more complex for the possibility semantics $\mathcal{L}_\Diamond$. Indeed, consider a parameterized expression e over Σ, with variables in $\mathcal{W}$. For the certainty semantics, it suffices to guess a word w and a valuation $\nu : \Sigma \to \mathcal{W}$ such that $w \notin \mathcal{L}(\nu(e))$. This gives a Pspace upper bound for this problem, which is the best that we can do, as the universality problem is Pspace-hard even for complete regular expressions. On the other hand, when solving this problem for the possibility semantics, one can expect that all possible valuations for e will need to be analyzed, which increases the complexity by one exponential. (In fact, when one moves to infinite alphabets, this problem becomes undecidable [14]). The lower bound proof is again by a generic reduction.

▶ **Theorem 10.**
- *The problem* Universality$_\Box$ *is* Pspace*-complete.*
- *The problem* Universality$_\Diamond$ *is* Expspace*-complete.*

Similarly to the nonemptiness problem (studied in Section 4.1), the Expspace bound for Universality$_\Diamond$ is quite resilient, as it holds even if for simple expressions (note that it makes no sense to study expressions of star-height 0, as they denote finite languages and thus cannot be universal).

▶ **Proposition 11.** *The problem* $\textsc{Universality}_\Diamond$ *remains* $\textsc{Expspace}$*-hard over the class of simple parameterized regular expressions.*

4.4 Containment

The bounds for the containment problem are easily obtained from the fact that both nonemptiness and universality can be cast as its versions. That is, we have:

▶ **Theorem 12.** *Both* $\textsc{Containment}_\Box$ *and* $\textsc{Containment}_\Diamond$ *are* $\textsc{Expspace}$*-complete.*

Proof: Since $\Sigma^* \subseteq \mathcal{L}_\Diamond(e)$ iff $\textsc{Universality}_\Diamond(e)$ is true, and $\mathcal{L}_\Box(e) \subseteq \emptyset$ iff $\textsc{Nonemptiness}_\Box(e)$ is false, we get $\textsc{Expspace}$-hardness for both containment problems. To check whether $\mathcal{L}_\Box(e_1) \subseteq \mathcal{L}_\Box(e_2)$, we must check that $\bigcap_\nu \mathcal{L}(\nu(e_1)) \cap \overline{\mathcal{L}(\nu'(e_2))} = \emptyset$ for each valuation ν' on e_2. This is doable in $\textsc{Expspace}$, since one can construct exponentially many automata for $\mathcal{L}(\nu(e_1))$ in $\textsc{Exptime}$, as well as the automaton for the complement $\overline{\mathcal{L}(\nu'(e_2))}$, and checking nonemptiness of the intersection of those is done in polynomial space in terms of their size, i.e., in $\textsc{Expspace}$. Since this needs to be done for exponentially many valuations ν', the overall $\textsc{Expspace}$ bound follows. The proof for the $\mathcal{L}_\Diamond$ semantics is almost identical.

Containment with one fixed expression We look at two variations of the containment problem, when one of the expressions is fixed: $\textsc{Containment}_*(e_1, \cdot)$ asks, for a parameterized regular expression e_2, whether $\mathcal{L}_*(e_1) \subseteq \mathcal{L}_*(e_2)$; and $\textsc{Containment}_*(\cdot, e_2)$ is defined similarly. The reductions proving Theorem 12 show that $\textsc{Containment}_\Box(\cdot, e_2)$ and $\textsc{Containment}_\Diamond(e_1, \cdot)$ remain $\textsc{Expspace}$-complete. For the other two versions of the problem, the proposition below shows that the complexity is lowered by at least one exponential.

▶ **Proposition 13.**

- $\textsc{Containment}_\Box(e_1, \cdot)$ *is* $\textsc{Pspace}$*-complete.*
- $\textsc{Containment}_\Diamond(\cdot, e_2)$ *is* $\textsc{coNP}$*-complete.*

4.5 Intersection with a regular language

This problem is a natural analog of the standard decision problem solved in automata-based verification; we also saw in the introduction that it arises when one computes certain answers to queries over incompletely specified graph databases.

Checking whether $\mathcal{L}(e') \cap \mathcal{L}_\Box(e) \neq \emptyset$ can be done in $\textsc{Expspace}$ using the same brute-force algorithm as for the nonemptiness problem (intersection of exponentially many regular languages). Since the nonemptiness problem is a special case with $e' = \Sigma^*$, we get the matching lower bound by Theorem 3. For $\mathcal{L}_\Diamond(e)$, an NP upper bound is easy: one just guesses a valuation so that $\mathcal{L}(e') \cap \mathcal{L}(\nu(e)) \neq \emptyset$. If e' denotes a single word w, we have an instance of the membership problem, and hence there is a matching lower bound, by Theorem 7. Summing up, we have:

▶ **Corollary 14.**

- *The problem* $\textsc{NonemptyIntReg}_\Box$ *is* $\textsc{Expspace}$*-complete.*
- *The problem* $\textsc{NonemptyIntReg}_\Diamond$ *is* NP*-complete.*

5 Computing automata

In this section, we first provide upper bounds for algorithms for building NFAs over Σ capturing $\mathcal{L}_\Diamond(e)$ and $\mathcal{L}_\Box(e)$, and then prove their optimality, by showing matching lower bounds on the sizes of such NFAs. Recall that we are dealing with the problem $\textsc{ConstructNFA}_*$: Given a parameterized regular expression e, construct an NFA $\mathcal{A}$ over Σ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}_*(e)$.

▶ **Proposition 15.** *The problem* $\textsc{ConstructNFA}_\diamond$ *can be solved in single-exponential time, and the problem* $\textsc{ConstructNFA}_\Box$ *can be solved in double-exponential time.*

These bounds are achieved by using naive algorithms for constructing automata: namely, one converts a parameterized regular expression e over variables in a finite set $\mathcal{W}$ into an automaton $\mathcal{A}_e$, and then for $|\Sigma|^{|\mathcal{W}|}$ valuations ν computes the automata $\nu(\mathcal{A}_e)$. This takes exponential time. To obtain an NFA for $\mathcal{L}_\diamond(e)$ one simply combines them with a nondeterministic choice; for $\mathcal{L}_\Box(e)$ one takes the product of them, resulting in double-exponential time.

We now show that these complexities are unavoidable, as the smallest NFAs capturing $\mathcal{L}_\diamond(e)$ or $\mathcal{L}_\Box(e)$ can be of single or double-exponential size, respectively. We say that the *sizes of minimal NFAs for $\mathcal{L}_*$ are necessarily exponential (resp., double-exponential)* if there exists a family $\{e_n\}_{n\in\mathbb{N}}$ of parameterized regular expressions such that:

- the size of each e_n is $O(n)$, and
- every NFA $\mathcal{A}$ satisfying $\mathcal{L}(\mathcal{A}) = \mathcal{L}_*(e_n)$ has at least 2^n (resp., 2^{2^n}) states.

▶ **Theorem 16.** *The sizes of minimal NFAs are necessarily double-exponential for $\mathcal{L}_\Box$, and necessarily exponential for $\mathcal{L}_\diamond$.*

Proof sketch: We begin with the double exponential bound for $\mathcal{L}_\Box$. For each $n \in \mathbb{N}$, let e_n be the following parameterized regular expression over alphabet $\Sigma = \{0, 1\}$ and variables $x_1, \ldots, x_{n+1}$:

$$e_n = ((0 \mid 1)^{n+1})^* \cdot x_1 \cdots x_n \cdot x_{n+1} \cdot ((0 \mid 1)^{n+1})^*.$$

Notice that each e_n uses $n + 1$ variables, and is of linear size in n. In order to show that every NFA deciding $\mathcal{L}_\Box(e_n)$ has 2^{2^n} states, we use the following result from [13]: if $L \subset \Sigma^*$ is a regular language, and there exists a set of pairs $P = \{(u_i, v_i) \mid 1 \leq i \leq m\} \subseteq \Sigma^* \times \Sigma^*$ such that (1) $u_i v_i \in L$, for every $1 \leq i \leq m$, and (2) $u_j v_i \notin L$, for every $1 \leq i, j \leq m$ and $i \neq j$, then every NFA accepting L has at least m states.

Given a collection S of words over $\{0, 1\}$, let w_S denote the concatenation, in lexicographical order, of all the words that belong to S, and let $w_{\bar{S},n}$ denote the concatenation of all words in $\{0, 1\}^{n+1}$ that are not in S.

Then, define a set of pairs $P_n = \{(w_S, w_{\bar{S},n}) \mid S \subset \{0, 1\}^{n+1}$ and $|S| = 2^n\}$. Since there are 2^{n+1} binary words of length $n + 1$, there are $\binom{2^{n+1}}{2^n}$ different subsets of $\{0, 1\}^{n+1}$ of size 2^n, and thus P_n contains $\binom{2^{n+1}}{2^n} \geq 2^{2^n}$ pairs. Moreover, one can show that $\mathcal{L}_\Box(e_n)$ and P_n satisfy properties (1) and (2) above, which proves the double exponential lower bound.

To show the exponential lower bound for $\mathcal{L}_\diamond$, define $e_n = (x_1 \cdots x_n)^*$, and let $P_n = \{(w, w) \mid w \in \{0, 1\}^n\}$. Clearly, P_n contains 2^n pairs. All that is left to do is to show that $\mathcal{L}_\diamond(e_n)$ and P_n satisfy properties (1) and (2) above. Details are omitted. □

Note that the bounds of Theorem 16 apply to simple regular expressions.

The table in Fig. 1 summarizes the main results in Sections 4 and 5.

6 Extending domains of variables

So far we assumed that variables take values in Σ: our valuations were partial maps $\nu : \mathcal{V} \to \Sigma$. We now consider a more general case when the range of each variable is a regular subset

Semantics Problem	Certainty $\Box$	Possibility $\Diamond$
Nonemptiness	Expspace-complete	NLogspace-complete (for automata)
Membership	coNP-complete	NP-complete
Containment	Expspace-complete	Expspace-complete
Universality	Pspace-complete	Expspace-complete
NonemptyIntReg	Expspace-complete	NP-complete
ConstructNFA	double-exponential	single-exponential

Figure 1 Summary of complexity results

of Σ^*.

Let e be a parameterized regular expression with variables $x_1, \ldots, x_n$, and let $L_1, \ldots, L_n \subseteq \Sigma^*$ be nonempty regular languages. We shall write $\bar{L}$ for $(L_1, \ldots, L_n)$. A valuation in $\bar{L}$ is a map $\nu : \bar{x} \to \bar{L}$ such that $\nu(x_i) \in L_i$ for each $i \leq n$. Under such a valuation, each parameterized regular expression e is mapped into a usual regular expression $\nu(e)$ over Σ, in which each variable x_i is replaced by the word $\nu(x_i)$. Hence we can still define

$$\begin{aligned}
\mathcal{L}_\Box(e;\bar{L}) &= \bigcap\{\mathcal{L}(\nu(e)) \mid \nu \text{ is a valuation over } \bar{L}\} \\
\mathcal{L}_\Diamond(e;\bar{L}) &= \bigcup\{\mathcal{L}(\nu(e)) \mid \nu \text{ is a valuation over } \bar{L}\}
\end{aligned}$$

According to this notation, $\mathcal{L}_\Box(e) = \mathcal{L}_\Box(e;(\Sigma,\ldots,\Sigma))$, and likewise for $\mathcal{L}_\Diamond$.

Note however that intersections and unions are now infinite, if some of the languages L_i's are infinite, so we cannot conclude, as before, that we deal with regular languages. And indeed they are not: for example, $\mathcal{L}_\Diamond(xx;\Sigma^*)$ is the set of square words, and thus not regular.

We now consider two cases. If each L_i is a finite language, we show that all the complexity results in Fig. 1 remain true. Then we look at the case of arbitrary regular L_i's. Languages $\mathcal{L}_\Diamond(e;\bar{L})$ need not be regular anymore, but languages $\mathcal{L}_\Box(e;\bar{L})$ still are, and we prove that the complexity bounds from the certainty column of Fig. 1 remain true. For complexity results, we assume that in the input $(e;\bar{L})$, each domain L_i is given either as a regular expression or an NFA over Σ.

6.1 The case of finite domains

If all domain languages L_i's are finite, all the lower bounds apply (they were shown when each $L_i = \Sigma$). For upper bounds, note that each finite L_i contains at most exponentially many words in the size of either a regular expression or an NFA that gives it, and each such word is polynomial size. Thus, the number of valuations is at most exponential in the size of the input, and each valuation can be represented in polynomial time. The following is then straightforward.

▶ **Proposition 17.** *If domains L_i's of all variables are finite nonempty subsets of Σ^*, then both $\mathcal{L}_\Box(e;\bar{L})$ and $\mathcal{L}_\Diamond(e;\bar{L})$ are regular languages, and all the complexity bounds on the problems related to them are exactly the same as stated in Fig. 1.*

6.2 The case of infinite domains

We have already seen that if just one of the domains is infinite, then $\mathcal{L}_\Diamond(e;\bar{L})$ need not be regular (the $\mathcal{L}_\Diamond(xx;\Sigma^*)$ example). Somewhat surprisingly, however, in the case of the certainty semantics, we recover not only regularity but also all the complexity bounds.

▶ **Theorem 18.** *For each parameterized regular expression e using variables $x_1,\ldots,x_n$ and for each an n-tuple $\bar{L}$ of regular languages over Σ, the language $\mathcal{L}_\Box(e;\bar{L})\subseteq\Sigma^*$ is regular. Moreover, the complexity bounds are exactly the same as in the $\Box$ column of the table in Fig. 1.*

Proof sketch: We only need to be concerned about regularity of $\mathcal{L}_\Box(e;\bar{L})$ and upper complexity bounds, as the proofs of lower bounds apply for the case when all $L_i=\Sigma$. For this, it suffices to prove that there is a finite set U of NFAs so that $\mathcal{L}_\Box(e;\bar{L})=\bigcap_{\mathcal{A}\in U}\mathcal{L}(\mathcal{A})$. Moreover, it follows from analyzing the proofs of upper complexity bounds, that the complexity results will remain the same if the following can be shown about the set U:

- its size is at most exponential in the size of the input;
- checking whether $\mathcal{A}\in U$ can be done in time polynomial in the size of $\mathcal{A}$;
- each $\mathcal{A}\in U$ is of size polynomial in the size of the input $(e;\bar{L})$.

To show these, take $\mathcal{A}_e$ and from it construct a reduced automaton $\mathcal{A}'_e$ in which all transitions (q,x_i,q') are eliminated whenever L_i is infinite. We then show that $\mathcal{L}_\Box(\mathcal{A}_e;\bar{L})=\mathcal{L}_\Box(\mathcal{A}'_e;\bar{L})$ (the definition of $\mathcal{L}_\Box$ extends naturally from regular expressions to automata for arbitrary domains). This observation generates a finite set U of NFAs which results from applying valuations with finite codomains to $\mathcal{A}'_e$. It is now possible to show that these automata satisfy the required properties.

7 Future work

For most bounds (except universality and containment), the complexity under the possibility semantics is reasonable, while for the certainty semantics it is quite high (i.e., double-exponential in practice). At the same time, the concept of $\mathcal{L}_\Box(e)$ captures many query answering scenarios over graph databases with incomplete information [5]. One of the future directions of this work is to devise better algorithms for problems related to the certainty semantics under restrictions arising in the context of querying graph databases.

Another line of work has to do with closure properties: we know that results of Boolean operations on languages $\mathcal{L}_\Box(e)$ and $\mathcal{L}_\Diamond(e)$ are regular and can be represented by NFAs; the bounds on sizes of such NFAs follow from the results shown here. However, it is conceivable that such NFAs can be succinctly represented by parameterized regular expressions. To be concrete, one can easily derive from results in Section 5 that there is a doubly-exponential size NFA $\mathcal{A}$ so that $\mathcal{L}(\mathcal{A})=\mathcal{L}_\Box(e_1)\cap\mathcal{L}_\Box(e_2)$, and that this bound is optimal. However, it leaves open a possibility that there is a much more succinct parameterized regular expression e so that $\mathcal{L}_\Box(e)=\mathcal{L}_\Box(e_1)\cap\mathcal{L}_\Box(e_2)$; in fact, nothing that we have shown contradicts the existence of a polynomial-size expression with this property. We plan to study bounds on such regular expressions in the future.

Acknowledgment We thank Marian Kędzierski for helpful comments. Partial support provided by Fondecyt grant 1110171, EPSRC grant G049165 and FET-Open Project FoX, grant agreement 233599.

References

1 S. Abiteboul, S. Cluet, T. Milo. Correspondence and translation for heterogeneous data. *TCS* 275 (2002), 179–213.

2 A. Aho. Algorithms for finding patterns in strings. *Handbook of Theoretical Computer Science*, Volume A: 255-300, 1990.

3 R. Angles, C. Gutiérrez. Survey of graph database models. *ACM Comp. Surv.* 40(1):(2008).

4 P. Barceló, C. Hurtado, L. Libkin, P. Wood. Expressive languages for path queries over graph-structured data. In *PODS'10*, pages 3-14.

5 P. Barceló, L. Libkin, J. Reutter. Querying graph patterns. In *PODS'11*, pages 199–210.

6 J. Bhadra, A. Martin, J. Abraham. A formal framework for verification of embedded custom memories of the Motorola MPC7450 microprocessor. *Formal Methods in System Design* 27(1-2): 67-112 (2005).

7 D. Calvanese, G. de Giacomo, M. Lenzerini, M. Y. Vardi. Containment of conjunctive regular path queries with inverse. In *KR'00*, pages 176–185.

8 D. Calvanese, G. de Giacomo, M. Lenzerini, M. Y. Vardi. Rewriting of regular expressions and regular path queries. *JCSS*, 64(3):443–465, 2002.

9 D. Calvanese, G. de Giacomo, M. Lenzerini, M. Y. Vardi. Simplifying schema mappings. In *ICDT 2011*.

10 M. P. Consens, A. O. Mendelzon. Low complexity aggregation in GraphLog and Datalog. *TCS* 116 (1993), 95–116.

11 D. Freydenberger. Extended regular expressions: succinctness and decidability. in *STACS 2011*, pages 507-518.

12 W. Gelade, F. Neven. Succinctness of the complement and intersection of regular expressions. In *STACS 2008*, pages 325–336.

13 I. Glaister, J. Shallit. A lower bound technique for the size of nondeterministic finite automata. *IPL* 59:75-77, 1996.

14 O. Grumberg, O. Kupferman, S. Sheinvald. Variable automata over infinite alphabets. In *LATA'10*, pages 561–572.

15 C. Hagenah, A. Muscholl. Computing epsilon-free NFA from regular expressions in $O(n\log^2(n))$ time. In *MFCS'98*, pages 277–285.

16 T. Imielinski, W. Lipski. Incomplete information in relational databases. *J. ACM* 31 (1984), 761–791.

17 L. Kari, A. Mateescu, G. Paun, A. Salomaa. Multi-pattern languages. *TCS* 141 (1995), 253-268.

18 M. Kaminsky, D. Zeitlin. Finite-memory automata with non-deterministic reassignment. *IJFCS* 21 (2010), 741-760.

19 D. Kozen. Lower bounds for natural proof systems. In *FOCS'77*, pages 254-266.

20 Y. Liu, T. Rothamel, F. Yu, S. Stoller, N. Hu. Parametric regular path queries. In *PLDI'04*, pages 219–230.

21 Y. Liu, S. Stoller. Querying complex graphs. In *PADL'06*, pages 199–214.

22 A. R. Meyer, L. J. Stockmeyer. Word problems requiring exponential time. In *STOC 1973*, pages 1–9.

23 O. de Moor, D. Lacey, E. Van Wyk. Universal regular path queries. *Higher-Order and Symbolic Computation* 16(1-2): 15-35 (2003).

24 G. Pesant. A regular language membership constraint for finite sequences of variables. In *CP'04*, pages 482–295.

25 X. Shen, Y. Zhong, C. Ding. Predicting locality phases for dynamic memory optimization. *J. Parallel Distrib. Comput.* 67(7): 783-796 (2007).