Parameterized Regular Expressions and Their Languages

Pablo Barceló[a], Juan Reutter[b], Leonid Libkin[b]

[a]*Department of Computer Science, University of Chile*
[b]*School of Informatics, University of Edinburgh*

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.

Keywords: regular expressions with variables, possibility semantics, certainty semantics, graph databases

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 topics as social networks, biological data, semantic Web and RDF, crime detection and analyzing network traffic; see [1] 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 [2, 3, 4, 5]. Note that in this setting of querying topology of a graph database, it is standard to use a finite alphabet for labeling [1].

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 [6, 7, 8]. 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* [9], 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 [10, 11, 12]. 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 [10, 12]. 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 phasesequence prediction for dynamic memory allocation [13], or as a compact way to express a family of legal behaviors in hardware verification [14], or as a tool to state regular constraints in constraint satisfaction problems [15].

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 but 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 [16, 17, 18]. Unlike the investigation in this paper, [17, 18] concentrated on the $\mathcal{L}_\Diamond(e)$ semantics in the context of *infinite* alphabets. The motivation of [17] 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 [1, 4, 10, 11]. Results in [17] show that under the possibility semantics and infinite alphabets, the resulting languages can also be accepted by non-deterministic register automata [18], and both closure and decidability become problematic. For example, universality and containment are undecidable over infinite alphabets [17]. 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 [7]. 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 [7] 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 [19] and algorithmic problems [20] have been investigated for them. However, even though parameterized regular expressions can be exponentially more succinct than regular expressions, 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 [21] or languages given by expressions with backreferences [22]. These languages need not be regular, and some of the problems (e.g., universality for backreferences) are undecidable [16]. In contrast, we show that under the certainty 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 $\mathcal{L}_\square$ 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 1. *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 $\mathcal{L}_\square(\mathcal{A})$ 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 $\mathcal{L}_\square(e) = \mathcal{L}_\square(\mathcal{A}_e)$ 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 – $\mathcal{L}_\square$ or $\mathcal{L}_\diamond$ – is used. So each problem will have a subscript $*$ that can be interpreted as $\square$ 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 $\mathcal{L}_*(e) = \mathcal{L}(\mathcal{A})$.

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

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

4. Decision problems

In this section we consider the five decision problems – nonemptiness, membership, universality, containment and intersection with a regular language – and provide precise complexity bounds for them. 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*. Infinite languages are another source of complexity, 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 algorithm, since $\mathcal{L}_\Diamond(e) \neq \emptyset$ for every parameterized regular expression e (except when $e = \emptyset$, which can be verified with a single pass over the expression). So we study this problem for the certainty semantics only; 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 1.
- *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 [17]. 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:

Lemma 2. *Given a set $e_1, \dots, e_k$ of parameterized expressions of size at most $n \geq k$, it is possible to build, in time $O(|\Sigma| \cdot k^2 \cdot n)$ an expression e' such that $\mathcal{L}_\Box(e')$ is empty if and only if $\mathcal{L}_\Box(e_1) \cap \dots \cap \mathcal{L}_\Box(e_k)$ 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 [23], 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 2, on the other hand, says that such a construction is possible for parameterized regular expressions under the certainty semantics. Next we prove Lemma 2:

Proof: Assume first that Σ has at least two symbols. Let $e_1, \dots, e_k$ be parameterized regular expressions as stated in the Lemma, and let a, b be different symbols in Σ. We use $(\Sigma - a)$ as a shorthand for the expression whose language is the union of every symbol in Σ different from a, and define $A^i = [(\Sigma - a)^* \cdot a \cdot (\Sigma - a)^*]^i$, for $1 \leq i \leq k - 1$. Finally, let $x_1, \dots, x_{k-1}$ be fresh variables. We define e' as

$$(\Sigma - a)^* \cdot x_1 \cdot (\Sigma - a)^* \cdot x_2 \cdot (\Sigma - a)^* \cdots x_{k-1} \cdot (\Sigma - a)^* \cdot \\ \big(ba^kb \cdot e_1 \mid b \cdot A^1 \cdot ba^kb \cdot e_2 \mid b \cdot A^2 \cdot ba^kb \cdot e_3 \mid \cdots \mid b \cdot A^{k-1} \cdot ba^kb \cdot e_k\big)$$

We prove next that $\mathcal{L}_\Box(e') \neq \emptyset$ if and only if $\mathcal{L}_\Box(e_1) \cap \dots \cap \mathcal{L}_\Box(e_k) \neq \emptyset$. For the *if* direction, consider a word $w \in \Sigma^*$ that belongs to $\mathcal{L}_\Box(e_1) \cap \dots \cap \mathcal{L}_\Box(e_k)$. Then it can be observed from the construction of e' that the word $(\bar{c}^k ab)^{k-1} ba^kbw$ belongs to $\mathcal{L}_\Box(e')$ where $\bar{c}$ is the concatenation (say, in lexicographical order) of all the symbols in Σ different from a.

On the other hand, assume that a word w belongs to $\mathcal{L}_\Box(e')$. It is clear that w must contain the substring ba^kb. Thus, there are words $u, v \in \Sigma^*$ such that $w = u \cdot ba^kb \cdot v$, and u does not contain the word ba^kb as a substring. Our goal is to prove that v belongs to $\mathcal{L}_\Box(e_1) \cap \dots \cap \mathcal{L}_\Box(e_k)$. But first we need to show that u contains exactly $k - 1$ appearances of the symbol a. We prove this statement by contradiction. Assume first that u contains less than $k - 1$ appearances of the symbol a. Then consider a valuation ν that maps each variable in e' to the symbol a. Since $\nu(e')$ is of the form

$$((\Sigma - a)^* a)^{k-1} (\Sigma - a)^* \cdot \big(ba^kb \cdot \nu(e_1) \mid b \cdot A^1 \cdot ba^kb \cdot \nu(e_2) \mid \\ b \cdot A^2 \cdot ba^kb \cdot \nu(e_3) \mid \cdots \mid b \cdot A^{k-1} \cdot ba^kb \cdot \nu(e_k)\big),$$

we conclude that the language of $\nu(e')$ cannot contain any word that starts with $u \cdot ba^kb$, since we have assumed that u contains less than $k - 1$ appearances of the symbol a. Next, assume that u contains more than $k - 1$ appearances of the symbol a, and consider a valuation ν' that maps each variable in e' to the symbol b. Then $\nu'(e')$ is of the form

$$((\Sigma - a)^* b)^{k-1} (\Sigma - a)^* (ba^kb \cdot \nu'(e_1) \mid b \cdot A^1 \cdot ba^kb \cdot \nu'(e_2) \mid \\ b \cdot A^2 \cdot ba^kb \cdot \nu'(e_3) \mid \cdots \mid b \cdot A^{k-1} \cdot ba^kb \cdot \nu'(e_k)).$$

Recall that we define A^i as $A^i = [(\Sigma - a)^* \cdot a \cdot (\Sigma - a)^*]^i$. Then notice that any word in $\mathcal{L}(\nu'(e'))$ is such that the symbol a cannot appear more than $k-1$ times before the substring ba^kb. We conclude that $\mathcal{L}(\nu'(e'))$ cannot contain a word starting with $u \cdot ba^kb$.

We have just proved that w can be decomposed into $u \cdot ba^kb \cdot v$, where u does not contain the substring ba^kb and has exactly $k-1$ appearances of the symbol a. With this observation, it is not difficult to show that, if a valuation ν assigns the symbol a to exactly j variables in $\{x_1, \dots, x_{k-1}\}$ $(0 \leq j \leq k-1)$, then v must belong to $\mathcal{L}_\Box(e_{k-j})$. This proves that v belongs to $\mathcal{L}_\Box(e_1) \cap \dots \cap \mathcal{L}_\Box(e_k)$, which was to be shown.

For the case when Σ contains a single symbol a, notice that for each $1 \leq i \leq k$ it is the case that $\mathcal{L}_\Box(e_i) = \mathcal{L}(e'_i)$, where e'_i is the expression resulting from replacing all parameters in e_i with the symbol a. We perform this replacement, and afterwards augment Σ with a fresh new symbol. The construction previously explained can be then used on input $e'_1, \dots, e'_k$. The correctness of this algorithm follows directly from the proof of the previous case, and the fact that the expressions $e'_1, \dots, e'_k$ contain no variables.

Regarding the size of the expression e', we have that the size of the first part of e', corresponding to $(\Sigma - a)^* \cdot x_1 \cdot (\Sigma - a)^* \cdot x_2 \cdot (\Sigma - a)^* \cdots x_{k-1} \cdot (\Sigma - a)^*$, is $O(|\Sigma| \cdot k)$. Furthermore, the second part comprises of a union of k expressions, each of them of size $O(|\Sigma| \cdot k \cdot n)$. Thus, the size of e' is $O(|\Sigma| \cdot k^2 \cdot n)$. $\Box$

Lower bound for certainty semantics: To complete the proof of Theorem 1, we prove an EXPSPACE lower bound for NONEMPTINESS$_\Box$, using a reduction from the acceptance problem for deterministic Turing machines that work in exponential space. Along the proof we use the shorthand $[i]$ to denote the binary representation of the number $i < 2^n$ as a string of n symbols from $\{0, 1\}$. For example, $[0]$ corresponds to the word 0^n, and $[2]$ corresponds to the word $0^{n-2}10$.

Let $L \subseteq \Sigma^*$ be a language that belongs to EXPSPACE, and let $\mathcal{M}$ be a Turing machine that decides L in EXPSPACE. Given an input $\bar{a} \in \Sigma^*$, we construct in polynomial time with respect to $\mathcal{M}$ and $\bar{a}$ a parameterized regular expression $e_{\mathcal{M},\bar{a}}$ such that $\mathcal{L}_\Box(e_{\mathcal{M},\bar{a}}) \neq \emptyset$ if and only if $\mathcal{M}$ accepts $\bar{a}$.

Assume that $\mathcal{M} = (Q, \Gamma, q_0, \{q_m\}, \delta)$, where $Q = \{q_0, \dots, q_m\}$ is the set of states, $\Gamma = \{0, 1, B\}$ is the tape alphabet (B is the *blank* symbol), the initial state is q_0, q_m is the unique final state, and $\delta : (Q \setminus \{q_m\}) \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function. Notice that we assume without loss of generality that no transition is defined on the final state q_m. Furthermore, we also assume without loss of generality that every accepting run of $\mathcal{M}$ ends after an odd number of computations. Since $\mathcal{M}$ decides L in EXPSPACE, there is a polynomial $S()$ such that, for every input $\bar{a}$ over Σ, $\mathcal{M}$ decides $\bar{a}$ using space of order $2^{S(|\bar{a}|)}$.

Let $\bar{a} = a_0 a_1 \cdots a_{k-1} \in \Sigma^*$ be an input for $\mathcal{M}$ (that is, each a_i, $0 \leq i \leq k-1$, is a symbol in Σ). For notational convenience we will assume from now on that $S(|\bar{a}|) = n$. Due to Lemma 2, it suffices to construct a set E of parameterized regular expressions, such that $\bigcap_{e \in E} \mathcal{L}_\Box(e)$ is empty if and only if $\mathcal{M}$ accepts on input $\bar{a}$.

Consider the alphabet $\Sigma = \{0, 1\}$. The idea of the reduction is to code the run of $\mathcal{M}$ on input $\bar{a}$ into a word in Σ^*, in such a way that $\bigcap_{e \in E} \mathcal{L}_\Box(e)$ contains precisely the words that code an accepting run τ for $\mathcal{M}$ on input $\bar{a}$. In intuitive terms, such a word w represents the sequence of "instant descriptions" of $\mathcal{M}$ with respect to run τ. We do it as follows.

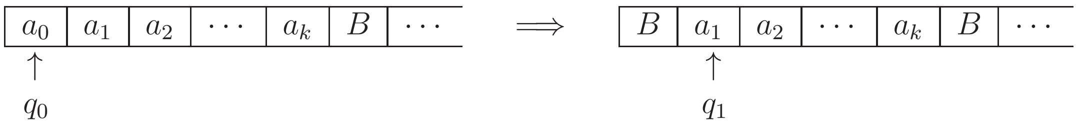
Assume that $\mathcal{M}$ performs m computations according to the run τ. With each $1 \leq i \leq 2^n$ and $i \leq j \leq m$, we associate a symbol $b_{i,j} \in \Gamma \cup (\Gamma \times Q)$, in such a way that $b_{i,j}$ corresponds to the symbol in the i-th cell of the tape in the j-th step of the run τ, if the head of $\mathcal{M}$ in the j-th step of the computation is not pointing into such cell, and otherwise as the pair (c, q), where c is the symbol in the i-th cell of the tape in the j-th step of the run τ, and q is the state of $\mathcal{M}$ in the j-th step of τ. We need each $b_{i,j}$ to be coded as a string over $\{0, 1\}$. In order to do this, let $p = |\Gamma \cup (\Gamma \times Q)|$. We shall code each symbol in $\Gamma \cup (\Gamma \times Q)$ in unary, i.e. as a p-bit string. We denote by $[b_{i,j}]$ the unary representation of the symbol $b_{i,j}$.

We also need to include information about the *action* that was performed in each cell of $\mathcal{M}$ at each step of the computation (i.e. read the cell, point the cell after moving the pointer, or nothing). More precisely, let $[\textit{nothing}] = 100$, $[\textit{read}] = 101$ and $[\textit{head}] = 111$, and define, for each $1 \leq i \leq 2^n$ and $1 \leq j \leq m$, the string $[\mathrm{ac}_{i,j}]$ as $[\textit{read}]$ if $\mathcal{M}$ is to read the content of the i-th cell at the j-th step of the computation; $[\textit{head}]$, if after the j-th computation $\mathcal{M}$ moves the head to point into the i-th cell of the tape, and $[\textit{nothing}]$ otherwise.

Roughly speaking, the idea is to define $w = w_1 \cdot w_2 \cdots w_m$, where each w_j is of the form:

$$\begin{array}{l}
[\mathrm{ac}_{(0,j)}] \cdot [b_{(0,j)}] \cdot [0] \cdot [b_{(0,j+1)}] \cdot \\
\qquad\qquad [\mathrm{ac}_{(1,j)}] \cdot [b_{(1,j)}] \cdot [1] \cdot [b_{(1,j+1)}] \cdot \\
\qquad\qquad\qquad\qquad \vdots \\
\qquad\qquad\qquad\qquad [\mathrm{ac}_{(2^n-1,j)}] \cdot [b_{(2^n-1,j)}] \cdot [2^n - 1] \cdot [b_{(2^n-1,j+1)}]
\end{array} \tag{1}$$

For example, assume that in the first step of the computation, $\mathcal{M}$ reads the first cell of the tape, writes a blank symbol, changes from state q_0 to q_1, and advances to the right. That is, the first and second configurations of $\mathcal{M}$ are as depicted in the following figure:



Then w_1 corresponds to the string

$$\begin{array}{ccccccccc}
[\textit{read}] & \cdot & [(a_0, q_0)] & \cdot & [0] & \cdot & [B] & \cdot \\
[\textit{head}] & \cdot & [a_1] & \cdot & [1] & \cdot & [(a_1, q_1)] & \cdot \\
[\textit{nothing}] & \cdot & [a_2] & \cdot & [2] & \cdot & [a_2] & \cdot \\
 & & \vdots & & & & & \\
[\textit{nothing}] & \cdot & [a_k] & \cdot & [k] & \cdot & [a_k] & \cdot \\
[\textit{nothing}] & \cdot & [B] & \cdot & [k+1] & \cdot & [B] & \cdot \\
 & & \vdots & & & & & \\
[\textit{nothing}] & \cdot & [B] & \cdot & [2^n - 1] & \cdot & [B] & \cdot
\end{array}$$

Essentially, we use 4 substrings to describe the action on each cell of the tape. The first substring, of length 3, refers to the action performed in that computation. In this case, an action [*read*] accompanies the first cell, since it was the cell read in the first step of the computation, and an action [*head*] accompanies the second cell, since as a result of the computation the head of $\mathcal{M}$ is now pointing into that cell. As expected, all other actions in w_1 are set to [*nothing*], since nothing was done to those cells in the first step of the computation. The second substring (of length p) refers to the content of the cell before the computation, the third is of length n, and contains the number identifying a particular cell as the i-th cell, from left to right, where i is binary, and the fourth string, of length p, is the content of that cell right after the computation.

Finally, we also need to explicitly distinguish *even* and *odd* computations of $\mathcal{M}$. Formally, let $[\mathit{even}] = 000$ and $[\mathit{odd}] = 001$. We construct E in such a way that if there is a word w in $\bigcap_{e \in E} \mathcal{L}_\Box(e)$ then it is of the form:

$$[\mathit{even}] \cdot w_1 \cdot [\mathit{even}] \cdot [\mathit{odd}] \cdot w_2 \cdot [\mathit{odd}] \cdot [\mathit{even}] \cdot w_3 \cdot [\mathit{even}] \cdots [\mathit{odd}] \cdot w_m \cdot [\mathit{odd}],$$

where each w_j is of the form (1), as explained above.

The rest of the proof is devoted to construct such set E. We divide the set E into sets E^1, E^2, E^3, E^4 and E^5.

First, E^1 contains only the expression

$$\Big([\mathit{even}]\big([\mathit{action}](0 \mid 1)^p(0 \mid 1)^n(0 \mid 1)^p\big)^*[\mathit{even}][\mathit{odd}]\big([\mathit{action}](0 \mid 1)^p(0 \mid 1)^n(0 \mid 1)^p\big)^*[\mathit{odd}]\Big)^*,$$

where $[\mathit{action}]$ is just a shorthand for the expression $([\mathit{read}] \mid [\mathit{head}] \mid [\mathit{nothing}])$. In intuitive terms, it ensures that all words accepted by $\bigcap_{e \in E} \mathcal{L}_\Box(e)$ are repetitions of sequences of subwords of length $3 + 2p + n$, contained between [*even*] or [*odd*] strings.

So far, we only have that all words in $\bigcap_{e \in E} \mathcal{L}_\Box(e)$ must be of the above form. The next step is to ensure that the number of substrings of the form $\big([\mathit{action}](0 \mid 1)^p(0 \mid 1)^n(0 \mid 1)^p\big)$ between any two strings [*even*] or [*odd*] has to be precisely 2^n (one for each cell used in the tape) and, furthermore, the numbers in binary representation used to code the position of the cell in each of these substrings (i.e., the part corresponding to $(0 \mid 1)^n$) have to be arranged in numerical order. To ensure this we use a set of regular expressions E^2. It is defined in such a way that the language $\bigcap_{e \in E^2} L(e)$ corresponds to the language accepted by the expression:

$$\begin{aligned}\big(([\mathit{even}] \mid [\mathit{odd}]) \cdot [\mathit{action}] \cdot (0 \mid 1)^p \cdot [0] \cdot (0 \mid 1)^p \cdot \\ [\mathit{action}] \cdot (0 \mid 1)^p \cdot [1] \cdot (0 \mid 1)^p \cdot \\ \vdots \\ [\mathit{action}] \cdot (0 \mid 1)^p \cdot [2^n - 1] \cdot (0 \mid 1)^p \cdot ([\mathit{even}] \mid [\mathit{odd}])\big)^*\end{aligned}$$

The definition of the set E^2 is standard, but very technical, and it is therefore omitted. It is based on the idea of representing the string $[0] \cdot [1] \cdots [2^n - 1]$ as an intersection of a polynomial number of regular expressions stating all together that, for each even $i \leq 2^n - 1$, the string $[i]$ has to be followed by the string $[i+1]$, and likewise for each odd number (see e.g. [23]).

Next, we ensure that the state and contents of the cells are carried along the descriptions. More precisely, E^3 must ensure that, if for some $1 \leq i < 2^n$ and $1 \leq j \leq m$, the word w_j features a substring of the form:

$$[\mathit{even}] \cdots [\mathit{action}] \cdot (0 \mid 1)^p \cdot [i] \cdot [b_{(i,j)}] \cdots [\mathit{even}],$$

with $b_{(i,j)} \in \Gamma \cup (\Gamma \times Q)$, then it must be directly followed by a string of form

$$[\mathit{odd}] \cdots [\mathit{action}] \cdot [b_{(i,j)}] \cdot [i] \cdots [\mathit{odd}],$$

so that the slots representing the content of the i-th cell after the j-th computation coincide with the slots representing the content of the i-th cell before the $j+1$-th computation.

It is straightforward to state such a condition by enumerating all cases, for each $0 \leq i \leq 2^n - 1$, but this would yield exponentially many equations. Instead, we exploit the use of parameters in our expression. We include in E^3 parameterized expressions E^3_1 and E^3_2, where E^3_1 (and, correspondingly, E^3_2) force that the content of the cell $x_1 \cdot x_2 \cdots x_n$ after an even (correspondingly, odd) computation corresponds exactly to the state before the next computation. To define E^3_1, consider the following expressions, for each $b \in \Gamma \cup (\Gamma \times Q)$:

$$\begin{aligned} E^3_{(1,b,\mathrm{even})} = [\mathit{even}] \cdot ([\mathit{action}] \cdot (0 \mid 1)^p \cdot (0 \mid 1)^n \cdot (0 \mid 1)^p)^* \cdot \\ [\mathit{action}] \cdot (0 \mid 1)^p \cdot x_1 \cdots x_n \cdot [b] \cdot \\ ([\mathit{action}] \cdot (0 \mid 1)^p \cdot (0 \mid 1)^n \cdot (0 \mid 1)^p)^* \cdot [\mathit{even}] \end{aligned}$$

$$\begin{aligned} E^3_{(1,b,\mathrm{odd})} = [\mathit{odd}] \cdot ([\mathit{action}] \cdot (0 \mid 1)^p \cdot (0 \mid 1)^n \cdot (0 \mid 1)^p)^* \cdot \\ [\mathit{action}] \cdot [b] \cdot x_1 \cdots x_n \cdot (0 \mid 1)^p \cdot \\ ([\mathit{action}] \cdot (0 \mid 1)^p \cdot (0 \mid 1)^n \cdot (0 \mid 1)^p)^* \cdot [\mathit{odd}] \end{aligned}$$

Then we define E^3_1 as follows:

$$E^3_1 = \Big(\bigcup_{b \in \Gamma \cup (\Gamma \times Q)} \big(E^3_{(1,b,\mathrm{even})} \cdot E^3_{(1,b,\mathrm{odd})} \big) \Big)^*$$

Expression E^3_2 is defined accordingly, simply by interchanging the order of $[\mathit{even}]$ and $[\mathit{odd}]$ strings, carefully checking that the first step of the computation is even, and allowing for the possibility that a word representing a computation ends in an odd configuration (that is, an odd configuration may be followed by an even configuration with the aforementioned properties, or may be the last configuration of the computation).

All that is left to do is to construct regular expressions that ensure that each of the substrings w_j ($1 \leq j \leq m$) of the word w in $\bigcap_{e \in E} \mathcal{L}_\Box(e)$ represent valid computations of $\mathcal{M}$. This is done by set E^4 of expressions, accepting all words such that:

- Between each two consecutive [*even*] or [*odd*] strings there is exactly one [*read*] and one [*head*] in the slots devoted to [*action*] in form (1).
- No other cell can change its context, except for those marked with [*read*] or [*head*], and
- The content that changes in the cells marked by [*read*] and [*head*] respects the transition function δ of $\mathcal{M}$.

Moreover, we also add a set of expressions E^5, accepting words such that:

- The initial configuration of $\mathcal{M}$ is encoded as the first step of the computation represented by w.
- The last computation ends in a final state of $\mathcal{M}$.

It is a tedious, but straightforward task to define the sets E^4 and E^5 of expressions. Furthermore, the fact that $\bigcap_{e \in E} \mathcal{L}_\Box(e)$ is empty if and only if $\mathcal{M}$ accepts on input $\bar{a}$ follows immediately from the remarks given along the construction. This finishes the proof. $\Box$

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 linear size in $|e|$).

Corollary 1. *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}.*

Before explaining the proof, we note that the single-exponential bound is easy to see (it was hinted at in the first paragraph of the introduction, and which was in fact used in connection with querying incomplete graph data in [7]). For each n, consider an expression $e_n = (0|1)^* x_1 \ldots x_n (0|1)^*$. 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 $2^n + (n-1)$.

Proof of Corollary 1: Clearly, for each $n \in \mathbb{N}$, it is possible to construct a deterministic Turing machine $\mathcal{M}_n$ over alphabet $\Sigma = \{0,1\}$ that on input 1^n works for exactly 2^{2^n} steps, using 2^n cells. Furthermore, it is possible to specify this machine using polynomial size with respect to n.

Next, using the construction in the reduction of Theorem 1, construct a set of parameterized regular expressions $E_{(\mathcal{M}_n,1^n)}$ such that the single word $w_n \in \bigcap_{e \in E_{(\mathcal{M}_n,1^n)}} \mathcal{L}_\Box(e)$ represents a run (or, more precisely, a sequence of configurations) of $\mathcal{M}_n$ on input 1^n. Note that each set $E_{(\mathcal{M}_n,1^n)}$ is of size polynomial with respect to n. Moreover, according to the reduction in the proof of Theorem 1, $\bigcap_{e \in E_{(\mathcal{M}_n,1^n)}} \mathcal{L}_\Box(e)$ contains a single word, of length greater than 2^{2^n}, representing the single run of $\mathcal{M}_n$ on input 1^n.

For each n, we define e_n as the expression such that $\mathcal{L}_\Box(e_n)$ is empty if and only if $\bigcap_{e \in E_{(\mathcal{M}_n,1^n)}} \mathcal{L}_\Box(e)$ is empty, constructed as in the proof of Lemma 2, so that e_n is of size polynomial with respect to $E_{(\mathcal{M}_n,1^n)}$. It follows from the proof of such lemma that every word accepted by $\mathcal{L}_\Box(e_n)$ has size at least 2^{2^n}. $\Box$

It is also possible to show that the problem NONEMPTINESS$_\Box$ remains EXPSPACE-hard over the class of simple regular expressions. Indeed, the reduction on the proof of Theorem 1 can be modified so that all the expressions in E are simple. As expected, the proof then becomes much more technical, and we have omitted it for the sake of space. We can also show that the use of Kleene star has a huge impact on complexity.

Proposition 2. *The problem* NONEMPTINESS$_\Box$ *is* Σ_2^p*-complete over the class of expressions of star-height* 0*.*

Proof: It is easy to see that, if e does not use Kleene star, then all the words $w \in \mathcal{L}_\Box(e)$ are of size polynomial with respect to the size of e. This immediately gives a Σ_2^P algorithm for the emptiness problem: Given a a parameterized regular expression e not using Kleene star, guess a word w, and check that $w \in \mathcal{L}_\Box(e)$. The proof then follows from the easy fact that MEMBERSHIP$_\Box$ can be solved in coNP, by guessing a valuation ν such that $w \notin \mathcal{L}(\nu(e))$ (we shall show in Section 4.2 that this bound turns out to be tight).

The Σ_2^P hardness is established via a reduction from the complement of the $\forall\exists$ 3-SAT satisfiability problem, which is known to be Π_2^P-complete. This problem is defined as follows: A formula φ is given as the conjunction of clauses $\{C_1,\ldots,C_p\}$, each of which has 3 variables taken from the union of disjoint sets $\{x_1,\ldots,x_m\}$ and $\{y_1,\ldots,y_t\}$. The problem asks whether there exists an assignment $\sigma_{\bar{x}}$ for $\{x_1,\ldots,x_m\}$ such that for every assignment $\sigma_{\bar{y}}$ for $\{y_1,\ldots,y_t\}$ it is the case that φ is not satisfiable.

Let $\varphi := \forall x_1 \cdots \forall x_m \exists y_1 \ldots \exists y_t \; C_1 \wedge \cdots \wedge C_p$ be an instance of $\forall\exists$ 3-SAT. From φ we construct in polynomial time a parameterized regular expression e over alphabet $\Sigma = \{0,1\}$ such that there exists an assignment $\sigma_{\bar{x}}$ for $\{x_1,\ldots,x_m\}$ such that for every assignment $\sigma_{\bar{y}}$ for $\{y_1,\ldots,y_t\}$ it is the case that φ is not satisfiable if and only if $\mathcal{L}_\Box(e)$ is not empty.

Let each C_j $(1 \leq j \leq p)$ be of the form $(\ell_j^1 \vee \ell_j^2 \vee \ell_j^3)$, where each literal ℓ_j^i, for $1 \leq j \leq p$ and $1 \leq i \leq 3$, is either a variable in $\{x_1,\ldots,x_m\}$ or $\{y_1,\ldots,y_t\}$, or its negation. We associate with each propositional variable x_k, $1 \leq k \leq m$, a fresh variable X_k (representing the positive literal) and a fresh variable $\hat{X}_k$ (representing the negation of such literal). In the same way, with each propositional variable y_k , $1 \leq k \leq t$, we associate fresh variables Y_k and $\hat{Y}_k$. Then let $\mathcal{W} = \{X_1,\ldots,X_m,\hat{X}_1,\ldots,\hat{X}_m\} \cup \{Y_1,\ldots,Y_t,,\hat{Y}_1,\ldots,\hat{Y}_t\} \cup \{Z\}$, where Z is a fresh variable as well.

We define an expression e over $\Sigma = \{0,1\}$ and $\mathcal{W}$ as follows:

$$e \;:=\; (Z \cdot 0 \cdot e_1) \mid (1 \cdot Z \cdot e_2),$$

where e_1 is the regular expression $1100 \cdot (0 \mid 1)^m \cdot 000$, and

$$e_2 \;:= e_{2,1,1} \mid \cdots \mid e_{2,1,m} \mid e_{2,2,1} \mid \cdots \mid e_{2,2,m} \mid e_{2,3,1} \mid \cdots \mid e_{2,3,t} \mid e_{2,4},$$

where

- for each $1 \leq k \leq m$, we have that $e_{2,1,k} = 1100 \cdot (0 \mid 1)^{k-1} \cdot X_k \cdot (0 \mid 1)^{m-k} \cdot 000$;
- for each $1 \leq k \leq m$, $e_{2,2,k} = \big(X_k \cdot \hat{X}_k \cdot 00 \cdot (0 \mid 1)^m \cdot 000\big) \mid \big(11 \cdot X_k \cdot \hat{X}_k \cdot (0 \mid 1)^m \cdot 000\big)$;
- for each $1 \leq k \leq t$, $e_{2,3,k} = \big(Y_k \cdot \hat{Y}_k \cdot 00 \cdot (0 \mid 1)^m \cdot 000\big) \mid \big(11 \cdot Y_k \cdot \hat{Y}_k \cdot (0 \mid 1)^m \cdot 000\big)$;
- Let h be a function that maps each literal ℓ_j^i to the variable X_k, if ℓ_j^i corresponds to x_k or to $\hat{X}_k$, if ℓ_j^i corresponds to $\neg x_k$ ($1 \leq j \leq p$, $1 \leq i \leq 3$ and $1 \leq k \leq m$); or to Y_k, if ℓ_j^i corresponds to y_k or to $\hat{Y}_k$, if ℓ_j^i corresponds to $\neg y_k$. Then define $e_{2,4} = 1100 \cdot (0 \mid 1)^m \cdot \big(h(\ell_1^1) \cdot h(\ell_1^2) \cdot h(\ell_1^3) \mid \cdots \mid h(\ell_p^1) \cdot h(\ell_p^2) \cdot h(\ell_p^3)\big)$.

We prove that $\mathcal{L}_\Box(e) \neq \emptyset$ if and only if there exists an assignment $\sigma_{\bar{x}}$ for $\{x_1, \dots, x_m\}$ such that for every assignment $\sigma_{\bar{y}}$ for $\{y_1, \dots, y_t\}$ it is the case that φ is not satisfiable.

($\Leftarrow$): Assume first that there exists such an assignment. Define $a_1, \dots, a_m \in \{0, 1\}$ as follows: for each $1 \leq k \leq m$, $a_k = 0$ if and only if $\sigma_{\bar{x}}$ assigns the value 1 to the variable x_k. We claim the word $w = 101100 \cdot a_1 \cdots a_m \cdot 000$ belongs to $\mathcal{L}_\Box(e)$. To prove this claim, let $\nu : \mathcal{W} \to \Sigma$ be an arbitrary valuation for the variables in e. We show that $w \in \mathcal{L}(\nu(e))$. The proof is done via a case analysis:

- Assume first that $\nu(Z) = 1$. Then, since $1100 \cdot a_1 \cdots a_m \cdot 000$ clearly belongs to the language defined by e_1, we have that $w \in \mathcal{L}(\nu(e))$.
- Next, assume that $\nu(Z) = 0$, and for some $1 \leq k \leq m$ it is the case that $\nu(X_k) = \nu(\hat{X}_k)$. Then, it is easy to see that $1100 \cdot a_1 \cdots a_m \cdot 000$ belongs to the language defined by the expression

$$\nu(e_{2,2,k}) = \big(\nu(X_k) \cdot \nu(\hat{X}_k) \cdot 00 \cdot (0 \mid 1)^m \cdot 000\big) \mid \big(11 \cdot \nu(X_k) \cdot \nu(\hat{X}_k) \cdot (0 \mid 1)^m \cdot 000\big).$$

  Thus we have that $w \in \mathcal{L}(\nu(e))$.
- Assume now that $\nu(Z) = 0$, and for some $1 \leq k \leq t$ it is the case that $\nu(Y_k) = \nu(\hat{Y}_k)$. Then, it is easy to see that $1100 \cdot a_1 \cdots a_m \cdot 000$ belongs to the language defined by the expression

$$\nu(e_{2,3,k}) = \big(\nu(Y_k) \cdot \nu(\hat{Y}_k) \cdot 00 \cdot (0 \mid 1)^m \cdot 000\big) \mid \big(11 \cdot \nu(Y_k) \cdot \nu(\hat{Y}_k) \cdot (0 \mid 1)^m \cdot 000\big).$$

  Thus we have that $w \in \mathcal{L}(\nu(e))$.

The remaining valuations are such that $\nu(Z) = 0$, $\nu(X_k) \neq \nu(\hat{X}_k)$ for each $1 \leq k \leq m$, and for each $1 \leq k \leq t$ we have that $\nu(Y_k) \neq \nu(\hat{Y}_k)$. We continue with those cases.

- Assume first that for some $1 \leq k \leq m$ it is the case that $\nu(X_k) = 1$ but $\sigma_{\bar{x}}(x_k) = 0$. Then $a_k = 1$, and thus we have that that the word $1100 \cdot a_1 \cdots a_m \cdot 000$ belongs to the language defined by $\nu(e_{2,1,k})$, that corresponds to the expression $1100 \cdot (0 \mid 1)^{k-1} \cdot \nu(X_k) \cdot (0 \mid 1)^{m-k} \cdot 000$. This implies that w belongs to the language defined by $\nu(e)$.

- The case where for some $1 \leq k \leq m$ it is the case that $\nu(X_k) = 0$ but $\sigma_{\bar{x}}(x_k) = 1$ is analogous to the previous one.

- The only remaining possibilities are such that $\nu(Z) = 0$, $\nu(X_k) \neq \nu(\hat{X}_k)$ for each $1 \leq k \leq m$, for each $1 \leq k \leq t$ we have that $\nu(Y_k) \neq \nu(\hat{Y}_k)$, and for each $1 \leq k \leq m$ it is the case that $\nu(X_k) = \sigma_{\bar{x}}(x_k)$. Define the following valuation $\sigma_{\bar{y}}$ for the variables in $\{y_1, \ldots, y_t\}$: $\sigma_{\bar{y}}(y_k) = \nu(Y_k)$, for each $1 \leq k \leq t$. From our initial assumption, there exists at least a clause C_j, $1 \leq j \leq p$, that is falsified under the assignment $\sigma_{\bar{x}}, \sigma_{\bar{y}}$. Then using the fact that $\nu(X_k) \neq \nu(\hat{X}_k)$ for each $1 \leq k \leq m$, for each $1 \leq k \leq t$ we have that $\nu(Y_k) \neq \nu(\hat{Y}_k)$, and for each $1 \leq k \leq m$ it is the case that $\nu(X_k) = \sigma_{\bar{x}}(x_k)$, we conclude that $\nu(h(\ell_j^1)) \cdot \nu(h(\ell_j^2)) \cdot \nu(h(\ell_j^3))$ corresponds to the string 000, which proves that $1100 \cdot a_1 \cdots a_m \cdot 000$ belongs to $\mathcal{L}(\nu(e_{2,4}))$, and thus w is denoted by $\nu(e)$.

($\Rightarrow$): Assume now that there exists a word $w \in \Sigma^*$ that belongs to $\mathcal{L}_\Box(e)$. We first state some facts about the general form of w. It is straightforward to show that w must begin with the prefix 10: if w begins with 00 then it cannot be denoted by $\nu_1(e)$, where ν_1 is the valuation that assigns a letter 1 to all variables in e; if w begins with 11 then it is not in the language of $\nu_0(e)$, where ν_0 is the valuation that assigns the letter 0 to all variables in e and if w begins with 01 then it cannot be denoted by any of the $\nu(e)$'s, for any valuation ν. We can then assume that $w = 10 \cdot v$, with $v \in \Sigma^*$. Furthermore, let ν be an arbitrary valuation such that $\nu(Z) = 1$. Since w belongs to $\mathcal{L}(\nu(e))$, from the form of w it follows that v belongs to $\nu(e_1)$ (which is in fact e_1, since this expression does not contain any variables). So we have that v is of form $1100 \cdot (0 \mid 1)^m \cdot 000$.

Define from v the following valuation $\sigma_{\bar{x}}$ for the propositional variables $\{x_1, \ldots, x_m\}$: $\sigma_{\bar{x}}(x_k) = 1$ if v is of form $1100 \cdot (0 \mid 1)^{k-1} \cdot 0 \cdot (0 \mid 1)^{m-k} \cdot 000$ (that is, if the $k+4$-th bit of v is 0), and $\sigma_{\bar{x}}(x_k) = 0$ if v is of form $1100 \cdot (0 \mid 1)^{k-1} \cdot 1 \cdot (0 \mid 1)^{m-k} \cdot 000$ (the $k+4$-th bit of v is 1).

Next we show that for each valuation $\sigma_{\bar{y}}$ for $\{y_1, \ldots, y_t\}$ it is the case that φ is not satisfied with valuation $\sigma_{\bar{x}}, \sigma_{\bar{y}}$. Assume for the sake of contradiction that there is a valuation $\sigma_{\bar{y}}$ for $\{y_1, \ldots, y_t\}$ such that $\sigma_{\bar{x}}, \sigma_{\bar{y}}$ satisfies φ. Define the following valuation $\nu : \mathcal{W} \to \Sigma$:

- $\nu(Z) = 0$
- $\nu(X_k) = \sigma_{\bar{x}}(x_k)$, for each $1 \leq k \leq m$,
- $\nu(Y_k) = \sigma_{\bar{y}}(y_k)$, for each $1 \leq k \leq t$,
- $\nu(\hat{X}_k) = 1$ if and only if $\nu(X_k) = 0$,
- $\nu(\hat{Y}_k) = 1$ if and only if $\nu(Y_k) = 0$

Let us now show that $10 \cdot v \notin \mathcal{L}(\nu(e))$, contradicting our initial assumption that $w \in \mathcal{L}_\Box(e)$.

Since $\nu(Z) = 0$, one concludes that v must belong to $\mathcal{L}_\Box(e_2)$. Since $\nu(X_k) \neq \nu(\hat{X}_k)$, for each $1 \leq k \leq m$, and for each $1 \leq k \leq t$ we have that $\nu(Y_k) \neq \nu(\hat{Y}_k)$, it is easy to see that the word v cannot be in $\mathcal{L}(\nu(e_{2,2,k}))$, for $1 \leq k \leq m$, or in $\mathcal{L}(\nu(e_{2,3,k}))$, for $1 \leq k \leq t$.

Moreover, since $\sigma_{\bar{x}}(x_k) = 1 = \nu(X_k)$ if v is of form $1100(0 \mid 1)^{k-1}0(0 \mid 1)^{m-k}000$, and $\sigma_{\bar{x}}(x_k) = 0 = \nu(X_k)$ if v is of form $1100(0 \mid 1)^{k-1}1(0 \mid 1)^{m-k}000$, we have that v cannot be in $\mathcal{L}(\nu(e_{2,1,k}))$ for $1 \leq k \leq m$. The only remaining possibility is that the word v belongs to $\mathcal{L}(\nu(e_{2,4}))$. This implies that for some $1 \leq j \leq p$, it is the case that $\nu(h(\ell_1^1))\cdot\nu(h(\ell_1^2))\cdot\nu(h(\ell_1^3))$ corresponds to the string 000. From the definition of ν, this implies that the j-th clause of φ is not satisfied under $\sigma_{\bar{x}}, \sigma_{\bar{y}}$, which is a contradiction. We conclude that for each valuation $\sigma_{\bar{y}}$ for $\{y_1, \ldots, y_t\}$ it is the case that φ is not satisfied with valuation $\sigma_{\bar{x}}, \sigma_{\bar{y}}$, which was to be shown. □

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 3.
- *The problem* MEMBERSHIP$_\Box$ *is* coNP*-complete.*
- *The problem* MEMBERSHIP$_\Diamond$ *is* NP*-complete.*

Note that for the case of the possibility semantics, the bound is the same as for languages over the infinite alphabets [17] (for all problems other than nonemptiness and membership, the bounds will be different). The hardness proof in [17] 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.

Instead of directly proving the above Theorem, we show that intractability follows already from much simpler cases. Indeed, the restrictions to expressions without repetitions, or to finite languages, by themselves do not lower the complexity, but together they make it polynomial.

Proposition 4. *The complexity of the membership problem remains as in Theorem 3 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).*

Proof: For the sake of readability, in this proof we use $\cup$ – instead of $\mid$ – for representing the operation of union between regular expressions.

1) $\Diamond$-semantics: We first consider the $\Diamond$-semantics. We start by showing NP-hardness of MEMBERSHIP$_\Diamond$, for regular expressions of star-height 0. We use a reduction from POSITIVE 1-3 3-SAT, which is the following NP-hard decision problem: Given a conjunction φ of clauses, with exactly three literals each, and in which no negated variable occurs, is there a truth assignment to the variables so that each clause has exactly one true variable?

The reduction is as follows. Let $\varphi = C_1 \wedge \cdots \wedge C_m$ be a formula in CNF, where each C_i $(1 \leq i \leq m)$ is a clause consisting of exactly tree positive literals. Let $\{p_1, \ldots, p_n\}$ be the variables that appear in φ. With each propositional variable p_i $(1 \leq i \leq n)$ we associate a different variable $x_i \in \mathcal{V}$. We show next how to construct, in polynomial time from φ, a parameterized regular expression e over alphabet $\Sigma = \{a, 0, 1\}$ and a word w over the same alphabet, such that there is an assignment to the variables of φ for which each clause has exactly one true variable if and only if $w \in \mathcal{L}_\diamond(e)$.

The parameterized regular expression e is defined as $ae_1ae_2a \cdots ae_ma$, where the regular expression e_i, for $1 \leq i \leq m$, is defined as follows: Assume that $C_i = (p_j \vee p_k \vee p_\ell)$, where $1 \leq j, k, \ell \leq n$. Then e_i is defined as $(x_jx_kx_\ell \,|\, x_jx_\ell x_k \,|\, x_kx_jx_\ell \,|\, x_kx_\ell x_j \,|\, x_\ell x_jx_k \,|\, x_\ell x_kx_j)$. That is, e_i is just the union of all the possible forms in which the variables in $\mathcal{V}$ that correspond to the propositional variables that appear in C_i can be ordered. Further, the word w is defined as $(a100)^ma$. Clearly, e and w can be constructed in polynomial time from φ. Next we show that there is an assignment for variables $\{p_1, \ldots, p_n\}$ for which each clause has exactly one true variable if and only if $w \in \mathcal{L}_\diamond(e)$.

Assume first that $w \in \mathcal{L}_\diamond(e)$. Then there exists a valuation $\nu : \{x_1, \ldots, x_n\} \to \Sigma$ such that $w \in \mathcal{L}(\nu(e))$. Thus, it must be the case that the word $a100$ belongs to $\nu(ae_i)$, for each $1 \leq i \leq m$. But this implies that if $C_i = (x_j \vee x_k \vee x_\ell)$, then ν assigns value 1 to exactly one of the variables in the set $\{x_j, x_k, x_\ell\}$ and it assigns value 0 to the other two variables. Let us define now a propositional assignment $\sigma : \{p_1, \ldots, p_n\} \to \{0, 1\}$ such that $\sigma(p_i) = \nu(x_i)$, for each $1 \leq i \leq n$. It is not hard to see then that for each clause C_j, $1 \leq j \leq m$, σ assigns value 1 to exactly one of its propositional variables.

Assume, on the other hand, that there is a propositional assignment $\sigma : \{p_1, \ldots, p_n\} \to \{0, 1\}$ that assigns value 1 to exactly one variable in each clause C_i, $1 \leq i \leq m$. Let us define ν as a valuation from $\{x_1, \ldots, x_n\}$ into $\{0, 1\}$ such that $\nu(x_i) = 1$ if and only if $\sigma(p_i) = 1$. Clearly then $100 \in \mathcal{L}(\nu(e_i))$, for each $1 \leq i \leq m$. Thus, $(a100)^ma \in \mathcal{L}(\nu(e))$. We conclude that $w \in \mathcal{L}_\diamond(e)$.

Next we prove NP-hardness of Membership$_\diamond$ for simple expressions. We use a reduction from 3-SAT. Let $\varphi = \bigwedge_{1 \leq i \leq n} (\ell_i^1 \vee \ell_i^2 \vee \ell_i^3)$ be a propositional formula in 3-CNF over variables $\{p_1, \ldots, p_m\}$. That is, each literal ℓ_i^j, for $1 \leq i \leq n$ and $1 \leq j \leq 3$, is either p_k or $\neg p_k$, for $1 \leq k \leq m$. Next we show how to construct in polynomial time from φ, a simple regular expression e over alphabet $\Sigma = \{a, b, c, d, 0, 1\}$ and a word w over the same alphabet such that φ is satisfiable if and only if $w \in \mathcal{L}_\diamond(e)$.

The regular expression e is defined as f^*, where $f := a(f_1 \cup g_1 \cup \cdots \cup f_m \cup g_m)b$, and the regular expressions f_i and g_i are defined as follows: Intuitively, f_i (resp. g_i) codifies p_i (resp. $\neg p_i$) and the clauses in which p_i (resp. $\neg p_i$) appears. Formally, we define f_i $(1 \leq i \leq m)$ as

$$\Big(\big(c^i \cup \bigcup_{\{1 \leq j \leq n \mid p_i = \ell_j^1 \text{ or } p_i = \ell_j^2 \text{ or } p_i = \ell_j^3\}} d^j\big) \cdot x_i\Big),$$

where x_i is a fresh variable in $\mathcal{V}$. In the same way we define g_i as

$$\Big(\big(c^i \cup \bigcup_{\{1 \leq j \leq n \mid \neg p_i = \ell_j^1 \text{ or } \neg p_i = \ell_j^2 \text{ or } \neg p_i = \ell_j^3\}} d^j\big) \cdot \bar{x}_i\Big),$$

where $\bar{x}_i$ is a fresh variable in $\mathcal{V}$. The variable x_i (resp. $\bar{x}_i$) is said to be *associated* with p_i (resp. $\neg p_i$) in e. Clearly, e is a simple regular expression and can be constructed in polynomial time from φ.

The word w is defined as:

$$ac1b\,ac0b\,acc1b\,acc0b \cdots ac^m1b\,ac^m0b\,ad1b\,add1b \cdots ad^n1b.$$

Clearly, w can be constructed in polynomial time from φ. Next we show that φ is satisfiable if and only if $w \in \mathcal{L}_\diamond(e)$.

Assume first that $w \in \mathcal{L}_\diamond(e)$. That is, there is a valuation ν for the variables in the set $\{x_1, \bar{x}_1, \dots, x_m, \bar{x}_m\}$ over Σ such that $w \in \mathcal{L}(\nu(e))$. But then, given the form of w, it is clear that ac^i1b and ac^i0b belong to $\mathcal{L}(\nu(f))$, for each $1 \leq i \leq m$. Notice that the only way for this to happen is that both $\nu(x_i)$ and $\nu(\bar{x}_i)$ take its value in the set $\{0,1\}$, and, further, $\nu(x_i) \neq \nu(\bar{x}_i)$. For the same reasons, $ad^j1b \in \mathcal{L}(\nu(f))$, for each $1 \leq j \leq n$. But the only way for this to happen is that for each $1 \leq j \leq n$ it is the case that either the variable associated with ℓ_j^1 or with ℓ_j^2 or with ℓ_j^3 in e is assigned value 1 by ν. Thus, the propositional assignment $\sigma : \{p_1, \dots, p_m\} \to \{0,1\}$, defined as $\sigma(p_i) = 1$ if and only if $\nu(x_i) = 1$, is well-defined and satisfies φ.

Assume, on the other hand, that there is a satisfying propositional assignment $\sigma : \{p_1, \dots, p_m\} \to \{0,1\}$ for φ. Consider the following valuation ν for e: For each $1 \leq i \leq m$ it is the case that $\nu(x_i) = \sigma(p_i)$ and $\nu(\bar{x}_i) = 1 - \sigma(x_i)$. Using essentially the same techniques as in the previous paragraph it is possible to show that $w \in \mathcal{L}(\nu(e))$, and, therefore, that $w \in \mathcal{L}_\diamond(e)$.

Next we show that Membership$_\diamond$ can be solved in time $O(mn \cdot \log^2 n)$ for simple expressions of star-height 0. Given a regular expression $e \in \text{REG}(\Sigma, \mathcal{V})$ that is simple and of star-height 0, one can construct in time $O(n \cdot \log^2 n)$ [24] an ε-free NFA $\mathcal{A}$ over $\Sigma \cup \mathcal{V}$ that accepts precisely $\mathcal{L}(e)$, and satisfies the following two properties: (1) Its underlying directed graph is acyclic (this is because e does not mention the Kleene star), and (2) for each $x \in \mathcal{V}$ that is mentioned in e there is at most one pair (q, q') of states of $\mathcal{A}$ such that $\mathcal{A}$ contains a transition from q to q' labeled x (this is because e is simple). From Lemma 1, checking whether $w \in \mathcal{L}_\diamond(e)$, for a given word $w \in \Sigma^*$, is equivalent to checking whether $w \in \mathcal{L}(\nu(\mathcal{A}))$, for some valuation ν for $\mathcal{A}$. We show how the latter can be done in polynomial time.

First, construct in time $O(m)$ a deterministic finite automaton (DFA) $\mathcal{B}$ over Σ such that $\mathcal{L}(\mathcal{B}) = \{w\}$. We assume without loss of generality that the set Q of states of $\mathcal{A}$ is disjoint from the set P of states of $\mathcal{B}$. Next we construct, the following NFA $\mathcal{A}'$ over the alphabet $\Sigma \cup (\mathcal{V} \times \Sigma)$ as follows: The set of states of $\mathcal{A}'$ is $Q \times P$. The initial state of $\mathcal{A}'$ is the pair (q_0, p_0), where q_0 is the initial state of $\mathcal{A}$ and p_0 is the initial state of $\mathcal{B}$. The final states of $\mathcal{A}'$ are precisely the pairs $(q, p) \in Q \times P$ such that q is a final state of $\mathcal{A}$ and p is a final state of $\mathcal{B}$. Finally, there is a transition in $\mathcal{A}'$ from state (q, p) to state (q', p') labeled $a \in \Sigma$ if and only if there is a transition in $\mathcal{A}$ from q to q' labeled a and there is a transition in $\mathcal{B}$ from p to p' labeled a. There is a transition in $\mathcal{A}'$ from state (q, p) to state (q', p') labeled $(x, a) \in \mathcal{V} \times \Sigma$ if and only there is a transition in $\mathcal{A}$ from q to q' labeled x and there is a transition in $\mathcal{B}$ from p to p' labeled a. Clearly, such construction can be performed by checking all combinations of transitions of both $\mathcal{A}$ and $\mathcal{B}$, and thus it can be performed in time $O(mn \cdot \log^2 n)$. Checking whether $\mathcal{L}(\mathcal{A}') \neq \emptyset$ can easily be done in linear time with respect to the size of $\mathcal{A}'$, thus obtaining the $O(mn \cdot \log^2 n)$ bound. We prove next that checking this is equivalent to checking whether $w \in L(\nu(\mathcal{A}))$, for some valuation ν for $\mathcal{A}$, which finishes the proof of the proposition in terms of the $\Diamond$-semantics.

Assume first that $\mathcal{L}(\mathcal{A}') \neq \emptyset$. Let $(q_0, p_0) \xrightarrow{u_1} (q_1, p_1) \xrightarrow{u_2} \cdots \xrightarrow{u_{n-1}} (q_{n-1}, p_{n-1}) \xrightarrow{u_n} (q_n, p_n)$ be an accepting run of $\mathcal{A}'$. That is, $u_1 u_2 \cdots u_n \in (\Sigma \cup (\mathcal{V} \times \Sigma))^*$ and (q_n, p_n) is a final state of $\mathcal{A}'$. Since the underlying directed graph of $\mathcal{A}$ is acyclic, and each variable x mentioned in e appears in at most one transition of $\mathcal{A}$, it must be the case that for each $1 \leq i < j \leq n$, if $u_i = (x_i, a_i) \in \mathcal{V} \times \Sigma$ and $u_j = (x_j, a_j) \in \mathcal{V} \times \Sigma$ then $x_i \neq x_j$. This implies that we can define a mapping $\nu : \mathcal{W} \to \Sigma$, where $\mathcal{W}$ is the set of variables used in transitions of $\mathcal{A}$, such that $\nu(x) = a$, if $u_i = (x, a)$ for some $1 \leq i \leq n$, and $\nu(x)$ is an arbitrary element $a' \in \Sigma$, otherwise. It is not hard to see that $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_{n-1}} q_{n-1} \xrightarrow{a_n} q_n$ is also an accepting run of $\mathcal{L}(\nu(\mathcal{A}))$ and that $a_1 a_2 \cdots a_n = w$. The latter can be proved as follows: Let $f : \{u_1, \ldots, u_n\} \to \Sigma$ be the mapping such that $f(u_i) = u_i$, if $u_i = a \in \Sigma$, and $f(u_i) = a$, if $u_i = (x, a) \in \mathcal{V} \times \Sigma$. Then clearly $p_0 \xrightarrow{f(u_1)} p_1 \xrightarrow{f(u_2)} \cdots \xrightarrow{f(u_{n-1})} p_{n-1} \xrightarrow{f(u_n)} p_n$ is an accepting run of $\mathcal{B}$, and, therefore, $w = f(u_1) \cdots f(u_n)$. Further, let $g : \{u_1, \ldots, u_n\} \to \Sigma$ be the mapping such that $g(u_i) = u_i$, if $u_i = a \in \Sigma$, and $g(u_i) = \nu(x) = a$, if $u_i = (x, a) \in \mathcal{V} \times \Sigma$. Then clearly $f(u_i) = g(u_i)$, for each $1 \leq i \leq n$, and, further, $q_0 \xrightarrow{g(u_1)} q_1 \xrightarrow{g(u_2)} \cdots \xrightarrow{g(u_{n-1})} q_{n-1} \xrightarrow{g(u_n)} q_n$ is an accepting run of $\mathcal{L}(\nu(\mathcal{A}))$. We conclude that $w \in \mathcal{L}(\nu(\mathcal{A}))$.

Assume, on the other hand, that $w \in \mathcal{L}(\nu(\mathcal{A}))$, for some valuation ν for $\mathcal{A}$. Suppose that $w = a_1 a_2 \cdots a_n$, where each $a_i \in \Sigma$ $(1 \leq i \leq n)$, and let $q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_{n-1}} q_{n-1} \xrightarrow{a_n} q_n$ be an accepting run of $\mathcal{L}(\nu(\mathcal{A}))$; i.e. q_n is a final state of $\mathcal{A}$. Assume that $i_1 < i_2 < \cdots < i_m$ are the only indexes in the set $\{0, 1, \ldots, n-1\}$ such that, for each $1 \leq j \leq m$, there is no transition labeled a_{i_j+1} from q_{i_j} to q_{i_j+1} in $\mathcal{A}$. Then there must be a transition in $\mathcal{A}$ from q_{i_j} to q_{i_j+1} labeled $x_{i_j} \in \mathcal{V}$. Consider an arbitrary accepting run $p_0 \xrightarrow{a_1} p_1 \xrightarrow{a_2} \cdots \xrightarrow{a_{n-1}} p_{n-1} \xrightarrow{a_n} p_n$ of $\mathcal{B}$; i.e. p_n is a final state of $\mathcal{B}$. Then it is clear that

$$(q_0, p_0) \xrightarrow{a_1} (q_1, p_1) \cdots (q_{i_1}, p_{i_1}) \xrightarrow{(x_{i_1}, a_{i_1+1})} (q_{i_1+1}, p_{i_1+1}) \cdots$$
$$(q_{i_m}, p_{i_m}) \xrightarrow{(x_{i_m}, a_{i_m+1})} (q_{i_m+1}, p_{i_m+1}) \cdots (q_{n-1}, p_{n-1}) \xrightarrow{a_n} (q_n, p_n)$$

is an accepting run of $\mathcal{A}'$. Thus, $\mathcal{L}(\mathcal{A}') \neq \emptyset$.

2) $\Box$-semantics: Now we deal with the $\Box$-semantics. That MEMBERSHIP$_\Box$ is coNP-hard, even over the class of expressions of star-height 0, follows from Theorem 5. In fact, such theorem proves something stronger: MEMBERSHIP$_\Box$ is coNP-hard for the class of expressions of star-height 0, even for a fixed word w. Next we prove that MEMBERSHIP$_\Box$ is coNP-hard, even over the class of simple regular expressions.

We use a reduction from 3-SAT to the complement of MEMBERSHIP$_\Box$ over the class of simple expressions. Let $\varphi = \bigwedge_{1 \leq i \leq n} (\ell_i^1 \vee \ell_i^2 \vee \ell_i^3)$ be a propositional formula in 3-CNF over variables $\{p_1, \ldots, p_m\}$. That is, each literal ℓ_i^j, for $1 \leq i \leq n$ and $1 \leq j \leq 3$, is either p_k or $\neg p_k$, for $1 \leq k \leq m$. Next, we show how to construct in polynomial time from φ, a simple regular expression e over alphabet $\Sigma = \{a, b, 0, 1\}$ and a word w over the same alphabet such that φ is satisfiable if and only if $w \notin \mathcal{L}_\Box(e)$.

We start by defining w as the following word, where we have distinguished several prefixes that will be used in the rest of the proof:

$$\underbrace{1111a\,11111a\,b\,1110a\,11110a\,b\,111111a\,1111111a\,b\,111110a\,1111110a\,b\cdots}_{w',w'',w_i^+,w_i^-,w_j^u}$$

$$\underbrace{\overbrace{\underbrace{\cdots 1^{2i+1}1a\,1^{2i+2}1a}_{w_i^+}\,b\,1^{2i+1}0a\,1^{2i+2}0a}^{w_i^-}\,b\cdots 1^{2m+1}1a\,1^{2m+2}1a\,b\,1^{2m+1}0a\,1^{2m+2}0a\,b}_{w',w'',w_j^u}$$

$$\underbrace{1^{3(m+1)+1}0a\,1^{3(m+1)+2}0a\,1^{3(m+1)+3}0a\,b\,1^{6(m+1)+1}0a\,1^{6(m+1)+2}0a\,1^{6(m+1)+3}0a\,b\cdots}_{w',w'',w_j^u}$$

$$\underbrace{\overbrace{\cdots 1^{3j(m+1)+1}0a\,1^{3j(m+1)+2}0a\,1^{3j(m+1)+3}0a}^{w_j^u}\,b\cdots}_{w',w''}$$

$$\overbrace{\underbrace{\cdots 1^{3n(m+1)+1}0a\,1^{3n(m+1)+2}0a\,1^{3n(m+1)+3}0a\,b}_{w''}\,b}^{w'}aa. \qquad (2)$$

As it is shown above, we denote by w' the prefix of w such that $w = w'aa$ and by w'' the prefix of w such that $w = w''baa$. Clearly, w can be constructed in polynomial time from φ.

The regular expression e is defined as $(\Sigma^* b \cup \varepsilon) f (b\Sigma^* \cup \varepsilon)$, where f is defined as:

$$\big((f_1 \cup g_1 \cup \cdots f_m \cup g_m)(a \cup \varepsilon)\big)^*.$$

Intuitively f_i (resp. g_i) codifies p_i (resp. $\neg p_i$) and the clauses in which p_i (resp. $\neg p_i$) appears. Formally, we define f_i $(1 \leq i \leq m)$ as

$$\Big(\big(\{w'\} \cup \{w''\} \cup 1^{2i+1} \cup \bigcup_{\{1\leq j\leq n \mid p_i = \ell_j^1\}} 1^{3j(m+1)+1} \cup \bigcup_{\{1\leq j\leq n \mid p_i = \ell_j^2\}} 1^{3j(m+1)+2} \cup \bigcup_{\{1\leq j\leq n \mid p_i = \ell_j^3\}} 1^{3j(m+1)+3}\big) \cdot x_i a\Big),$$

where x_i is a fresh variable in $\mathcal{V}$. In the same way we define g_i as

$$\Big(\big(\{w'\} \cup \{w''\} \cup 1^{2i+2} \cup \bigcup_{\{1\leq j\leq n \mid \neg p_i = \ell_j^1\}} 1^{3j(m+1)+1} \cup \bigcup_{\{1\leq j\leq n \mid \neg p_i = \ell_j^2\}} 1^{3j(m+1)+2} \cup \bigcup_{\{1\leq j\leq n \mid \neg p_i = \ell_j^3\}} 1^{3j(m+1)+3}\big) \cdot \bar{x}_i a\Big),$$

where $\bar{x}_i$ is a fresh variable in $\mathcal{V}$. The variable x_i (resp. $\bar{x}_i$) is said to be *associated* with p_i (resp. $\neg p_i$) in e. Clearly, e is a simple regular expression and can be constructed in polynomial time from φ.

Next we show that φ is satisfiable if and only if $w \notin \mathcal{L}_\Box(e)$.

We prove first that if $w \notin \mathcal{L}_\Box(e)$ then φ is satisfiable. Assume that $w \notin \mathcal{L}_\Box(e)$. Then there exists a valuation $\nu : \{x_1, \bar{x}_1, \dots, x_m, \bar{x}_m\} \to \Sigma$ such that $w \notin \mathcal{L}(\nu(e))$. First of all, we prove that for each $1 \leq i \leq m$ both $\nu(x_i)$ and $\nu(\bar{x}_i)$ belong to the set $\{0, 1\}$. Assume, for the sake of contradiction, that this is not the case. Suppose first that $\nu(x_i) = a$, for some $1 \leq i \leq m$. Then it is clear that $\mathcal{L}(w'aa) \subseteq \mathcal{L}(\nu(e))$ (because $\mathcal{L}(w'\nu(x_i)a) \subseteq \mathcal{L}(\nu(e))$). But $w = w'aa$, and, therefore, $w \in \mathcal{L}(\nu(e))$, which is a contradiction. Suppose now that $\nu(x_i) = b$, for some $1 \leq i \leq m$. Then, again, it is clear that $\mathcal{L}(w''baa) \subseteq \mathcal{L}(\nu(e))$ (because $\mathcal{L}(w''\nu(x_i)aa) \subseteq \mathcal{L}(\nu(e))$). As in the previous case, $w = w''baa$, and, therefore, $w \in \mathcal{L}(\nu(e))$, which is a contradiction. The other case, when $\nu(\bar{x}_i) \in \{a, b\}$, for some $1 \leq i \leq m$, is completely analogous.

Next we prove that for each $1 \leq i \leq m$ it is the case that $\nu(x_i) = 1 - \nu(\bar{x}_i)$. Assume otherwise. Then for some $1 \leq i \leq m$ it is the case that $\nu(x_i) = \nu(\bar{x}_i)$. Suppose first that $\nu(x_i) = \nu(\bar{x}_i) = 1$. Consider the unique prefix w_1 of w that is of the form $u1^{2i+1}1a1^{2i+2}1a$, for $u \in \Sigma^*$. Then w is of the form w_1w_2, where $w_2 \in b\Sigma^*$. Since $w \notin \mathcal{L}(\nu(e))$, it must be the case that $w \notin \mathcal{L}((\Sigma^*b \cup \varepsilon)\nu(f)(b\Sigma^* \cup \varepsilon))$. It follows that $w_1 \notin \mathcal{L}((\Sigma^*b \cup \varepsilon)\nu(f))$. But since w_1 is of the form $u1^{2i+1}1a1^{2i+2}1a$, it follows that $u = \varepsilon$ or $u = u'b$, for some $u' \in \Sigma^*$. In any case it must hold that $1^{2i+1}1a1^{2i+2}1a \notin \mathcal{L}(\nu(f))$. Notice, however, that $\mathcal{L}(1^{2i+1}\nu(x_i)a1^{2i+2}\nu(\bar{x}_i)a) \subseteq \mathcal{L}(\nu(f))$. Hence, $1^{2i+1}1a1^{2i+2}1a \in \mathcal{L}(\nu(f))$, which is a contradiction. Suppose, on the other hand, that $\nu(x_i) = \nu(\bar{x}_i) = 0$. Consider the unique prefix w_1 of w that is of the form $u1^{2i+1}0a1^{2i+2}0a$, for $u \in \Sigma^*$. Then w is of the form w_1w_2, where $w_2 \in b\Sigma^*$. Since $w \notin \mathcal{L}(\nu(e))$, it must be the case that $w \notin \mathcal{L}((\Sigma^*b \cup \varepsilon)\nu(f)(b\Sigma^* \cup \varepsilon))$. It follows that $w_1 \notin \mathcal{L}((\Sigma^*b \cup \varepsilon)\nu(f))$. But since w_1 is of the form $u1^{2i+1}0a1^{2i+2}0a$, it follows that $u = \varepsilon$ or $u = u'b$, for some $u' \in \Sigma^*$. In any case it must hold that $1^{2i+1}0a1^{2i+2}0a \notin \mathcal{L}(\nu(f))$. Notice, however, that $\mathcal{L}(1^{2i+1}\nu(x_i)a1^{2i+2}\nu(\bar{x}_i)a) \subseteq \mathcal{L}(\nu(f))$. Hence, $1^{2i+1}0a1^{2i+2}0a \in \mathcal{L}(\nu(f))$, which is a contradiction.

We can then define a propositional assignment $\sigma : \{p_1, \dots, p_m\} \to \{0, 1\}$ such that $\sigma(p_i) = \nu(x_i)$, for each $1 \leq i \leq m$. Notice, from our previous remarks, that $\sigma(\neg p_i) = 1 - \nu(x_i) = \nu(\bar{x}_i)$, for each $1 \leq i \leq m$. We prove next that σ satisfies φ. Assume this is not the case. Then for some $1 \leq j \leq n$ it is the case that $\sigma(\ell_j^1) = \sigma(\ell_j^2) = \sigma(\ell_j^3) = 0$. Consider now the unique prefix w_1 of w such that w_1 is of the form $ub1^{3j(m+1)+1}0a1^{3j(m+1)+2}0a1^{3j(m+1)+3}0a$, for $u \in \Sigma^*$. Then w is of the form w_1w_2, where $w_2 \in b\Sigma^*$. Since $w \notin \mathcal{L}(\nu(e))$, it must be the case that $w \notin \mathcal{L}(\Sigma^*b\nu(f)b\Sigma^*)$. It follows that $w_1 \notin \mathcal{L}(\Sigma^*b\nu(f))$. But since w_1 is of the form $ub1^{3j(m+1)+1}0a1^{3j(m+1)+2}0a1^{3j(m+1)+3}0a$, it is the case that

$$1^{3j(m+1)+1}0a1^{3j(m+1)+2}0a1^{3j(m+1)+3}0a \notin \mathcal{L}(\nu(f)).$$

Let q_1, q_2 and q_3 be the variables in e associated with ℓ_j^1, ℓ_j^2 and ℓ_j^3, respectively. Then it cannot be the case that $\nu(q_1) = \nu(q_2) = \nu(q_3) = 0$. Assume otherwise. It is clear that $\mathcal{L}(1^{3j(m+1)+1}\nu(q_1)a1^{3j(m+1)+2}\nu(q_2)a1^{3j(m+1)+3}\nu(q_3)a) \subseteq \mathcal{L}(\nu(f))$, and, therefore,

$$1^{3j(m+1)+1}0a1^{3j(m+1)+2}0a1^{3j(m+1)+3}0a \in \mathcal{L}(\nu(f)),$$

which is a contradiction. Thus, either $\nu(q_1) = \sigma(\ell_j^1) = 1$ or $\nu(q_2) = \sigma(\ell_j^2) = 1$ or $\nu(q_3) = \sigma(\ell_j^3) = 1$. This is our desired contradiction.

We prove second that if φ is satisfiable then $w \notin \mathcal{L}_\Box(e)$. Assume that φ is satisfiable. Then there exists a propositional assignment $\sigma : \{p_1, \dots, p_m\} \to \{0, 1\}$ that satisfies φ. We define a valuation $\nu : \{x_1, \bar{x}_1, \dots, x_m, \bar{x}_m\} \to \{0, 1\}$ for e as follows: For each $1 \le i \le m$ it is the case that $\nu(x_i) = \sigma(p_i)$ and $\nu(\bar{x}_i) = 1 - \sigma(p_i)$. We prove next that $w \notin \mathcal{L}(\nu(e))$.

Clearly, $w \notin \mathcal{L}(\nu(e))$ if and only if for each words $w_1, w_2, w_3 \in \Sigma^*$ such that $w = w_1w_2w_3$ it is the case that $w_1 \notin \mathcal{L}(\Sigma^*b \cup \varepsilon)$ or $w_2 \notin \mathcal{L}(\nu(f))$ or $w_3 \notin \mathcal{L}(b\Sigma^* \cup \varepsilon)$. Thus, in order to prove that $w \notin \mathcal{L}(\nu(e))$ it is enough to prove that for each words $w_1, w_2, w_3 \in \Sigma^*$ such that $w = w_1w_2w_3$,

$$(*) \text{ if } w_1 \in \mathcal{L}(\Sigma^*b \cup \varepsilon) \text{ and } w_3 \in \mathcal{L}(b\Sigma^* \cup \varepsilon) \text{ then } w_2 \notin \mathcal{L}(\nu(f)).$$

Take arbitrary words $w_1, w_2, w_3 \in \Sigma^*$ such that $w = w_1w_2w_3$. We consider several cases:

1. Either $w_1 \notin \mathcal{L}(\Sigma^*b \cup \varepsilon)$ or $w_3 \notin \mathcal{L}(b\Sigma^* \cup \varepsilon)$. Then (*) is trivially true.
2. It is the case that $w_1 \in \mathcal{L}(\Sigma^*b \cup \varepsilon)$, $w_3 \in \mathcal{L}(b\Sigma^* \cup \varepsilon)$, and w_2 is of the form $1^{2i+1}1a1^{2i+2}1au$, for some $1 \le i \le m$ and $u \in \Sigma^*$. Assume, for the sake of contradiction, that $w_2 \in \mathcal{L}(\nu(f))$. Since clearly there is no word accepted by $\mathcal{L}(\nu(f))$ with prefix baa, it must be the case that w_3 is not the empty word, and, therefore, that $w_3 \in \mathcal{L}(b\Sigma^*)$. Thus, the only possibility for w_2 to belong to $\mathcal{L}(\nu(f))$ is that $1^{2i+1}1a \in \mathcal{L}(\nu(f_i))$ and $1^{2i+2}1a \in \mathcal{L}(\nu(g_i))$. But this can only happen if $\nu(x_i) = 1$ and $\nu(\bar{x}_i) = 1$, which is our desired contradiction (since $\nu(x_i) = 1 - \nu(\bar{x}_i)$).
3. It holds that $w_1 \in \mathcal{L}(\Sigma^*b \cup \varepsilon)$, $w_3 \in \mathcal{L}(b\Sigma^* \cup \varepsilon)$, and w_2 is of the form $1^{2i+1}0a1^{2i+2}0au$, for some $1 \le i \le m$ and $u \in \Sigma^*$. This case is completely analogous to the previous one.
4. It is the case that $w_1 \in \mathcal{L}(\Sigma^*b \cup \varepsilon)$, $w_3 \in \mathcal{L}(b\Sigma^* \cup \varepsilon)$, and w_2 is of the form
$$1^{3j(m+1)+1}0a1^{3j(m+1)+2}0a1^{3j(m+1)+3}0au,$$
for some $1 \le j \le n$ and $u \in \Sigma^*$. Assume, for the sake of contradiction, that $w_2 \in \mathcal{L}(\nu(f))$. It is easy to see that the only way in which this can happen is that $\nu(q_1) = \nu(q_2) = \nu(q_3) = 0$, where q_1, q_2 and q_3 are the variables in e that are associated with ℓ_j^1, ℓ_j^2 and ℓ_j^3, respectively. Thus, $\sigma(\ell_j^1) = \sigma(\ell_j^2) = \sigma(\ell_j^3) = 0$, which is or desired contradiction.

This finishes the proof of the proposition. □

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 5. 1. *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*).*
2. *For each word $w \in \Sigma^*$, the problem* MEMBERSHIP$_\Box(w)$ *is solvable in linear time, if restricted to the class of simple expressions.*

3. *For each word $w \in \Sigma^*$, the problem* $\textsc{Membership}_\Diamond(w)$ *is solvable in time $O(n \log^2 n)$, where n is the size of the expression.*

Proof: We prove each item separately:

(1) We proved in [7] that there exists a word $w \in \Sigma^*$ and a class $\mathfrak{A}$ of NFAs over alphabet $(\{0,1\} \cup \mathcal{V})$ such that the problem of checking, for a given $\mathcal{A} \in \mathfrak{A}$, whether $w \in \mathcal{L}(\nu(\mathcal{A}))$, for each valuation ν for $\mathcal{A}$, is a coNP-hard problem. It is clear from the construction in [7] that there is a polynomial time algorithm that, given an NFA $\mathcal{A} \in \mathfrak{A}$, constructs a star-free regular expression e over alphabet $(\{0,1\} \cup \mathcal{V})$ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(e)$. We can conclude, from Lemma 1, that the problem $\textsc{Membership}_\Box(w)$ is coNP-hard, even if restricted to the class of star-free regular expressions.

(2) Second, we prove that, for each word $w \in \Sigma^*$, the problem $\textsc{Membership}_\Box(w)$ can be solved in polynomial time (actually, in linear time with respect to the size of the expression) if restricted to the class of simple expressions. In order to do this, we first define a high-level procedure $\texttt{CheckSimpleNonMemb}_\Box$, that takes as input a simple parameterized regular expression e over Σ and a nonempty finite set $\mathfrak{W} \subset \Sigma^*$, and checks whether there exists an assignment ν for e such that no word from $\mathfrak{W}$ belongs to $\mathcal{L}(\nu(e))$. Then the answer to $\textsc{Membership}_\Box(w)$ for an expression e is $\neg\texttt{CheckSimpleNonMemb}_\Box(e, \{w\})$.

The procedure $\texttt{CheckSimpleNonMemb}_\Box$ works recursively on input e and $\mathfrak{W}$. For each internal node of the parse tree of e it iterates over some sets $\mathfrak{W}_1$ (or pairs of sets $(\mathfrak{W}_1, \mathfrak{W}_2)$ respectively), and for each such set (or pair) calls itself recursively on the children of the analyzed node. If the returned answers of one of the sets (or pairs) satisfy a given condition, the call accepts.

The details of the definition of $\texttt{CheckSimpleNonMemb}_\Box$ are following:

1. If $e = \varepsilon$, then $\texttt{CheckSimpleNonMemb}_\Box(e, \mathfrak{W})$ accepts if and only if $\varepsilon \notin \mathfrak{W}$.
2. If $e = a$, for $a \in \Sigma$, then $\texttt{CheckSimpleNonMemb}_\Box(e, \mathfrak{W})$ accepts if an only if $a \notin \mathfrak{W}$.
3. If $e = x$, for $x \in \mathcal{V}$, then $\texttt{CheckSimpleNonMemb}_\Box(e, \mathfrak{W})$ accepts if and only if $\mathfrak{W}$ does not contain all one-letter words.
4. If e is of the form $e_1 \cup e_2$, then $\texttt{CheckSimpleNonMemb}_\Box(e, \mathfrak{W})$ accepts if and only if $\texttt{CheckSimpleNonMemb}_\Box(e_1, \mathfrak{W})$ accepts and $\texttt{CheckSimpleNonMemb}_\Box(e_2, \mathfrak{W})$ accepts.
5. If e is of the form $e_1 e_2$, then $\texttt{CheckSimpleNonMemb}_\Box(e, \mathfrak{W})$ accepts if and only if there exist finite sets $\mathfrak{W}_1 \subset \Sigma^*$, $\mathfrak{W}_2 \subset \Sigma^*$ such that: (1) For each word $w_1 w_2 \in \mathfrak{W}$ it is the case that $w_1 \in \mathfrak{W}_1$ or $w_2 \in \mathfrak{W}_2$, (2) $\texttt{CheckSimpleNonMemb}_\Box(e_1, \mathfrak{W}_1)$ accepts and (3) $\texttt{CheckSimpleNonMemb}_\Box(e_2, \mathfrak{W}_2)$ accepts.
6. If e is of the form $(e_1)^*$, then $\texttt{CheckSimpleNonMemb}_\Box(e, \mathfrak{W})$ accepts if and only if $\varepsilon \notin \mathfrak{W}$ and there is a finite set $\mathfrak{W}_1 \subset \Sigma^*$ such that: (1) For each $w_1, w_2, \ldots, w_k \in \Sigma^+$, if $w_1 w_2 \cdots w_k \in \mathfrak{W}$ then at least one w_i $(1 \le i \le k)$ belongs to $\mathfrak{W}_1$, and (2) $\texttt{CheckSimpleNonMemb}_\Box(e_1, \mathfrak{W}_1)$ accepts.

It is interesting to see why $\texttt{CheckSimpleNonMemb}_\Box$ needs to operate on sets of words instead of single words. The above procedure may construct non-singleton sets in case of concatenation and Kleene star and we cannot analyse their elements separately, because in each case we must judge existence of a valuation ν, which would simultaneously prevent *all* possible matches of w on $\nu(e)$ from being accepting.

Next we prove that the procedure descibed above is sound and complete; that is, we prove that for each simple expression e over Σ and $\mathfrak{W} \subset \Sigma^*$, $\texttt{CheckSimpleNonMemb}_\Box$ accepts input e and $\mathfrak{W}$ if and only if there exists a valuation ν for e such that no word in $\mathfrak{W}$ belongs to $\mathcal{L}(\nu(e))$. We do this by induction:

1. The basis cases – when $e = \varepsilon$, $e = a$, for $a \in \Sigma$, or $e = x$, for $x \in \mathcal{V}$ – are trivial.
2. Assume e is of the form $e_1 \cup e_2$. Then there is a valuation ν for e such that no word in $\mathfrak{W}$ belongs to $\mathcal{L}(\nu(e))$ if and only if there is a valuation ν for e such that for each $w \in \mathfrak{W}$ we have $w \notin \mathcal{L}(\nu(e_1))$ and $w \notin \mathcal{L}(\nu(e_2))$. But since we consider only simple expressions here, the latter holds if and only if there are valuations ν_1 for e_1 and ν_2 for e_2 such that (a) no word $w \in \mathfrak{W}$ belongs to $\mathcal{L}(\nu_1(e_1))$, and (b) no word $w \in \mathfrak{W}$ belongs to $\mathcal{L}(\nu_2(e_2))$. By the inductive hypothesis, the latter holds if and only if $\texttt{CheckSimpleNonMemb}_\Box(e_1, \mathfrak{W})$ accepts and $\texttt{CheckSimpleNonMemb}_\Box(e_2, \mathfrak{W})$ accepts, which, by definition, is equivalent to the fact that $\texttt{CheckSimpleNonMemb}_\Box(e, \mathfrak{W})$ accepts.
3. Assume e is of the form e_1e_2. Then there is a valuation ν for e such that no word $w \in \mathfrak{W}$ belongs to $\mathcal{L}(\nu(e))$ if and only if there is a valuation ν for e such that for each word $w_1w_2 \in \mathfrak{W}$ it is the case that $w_1 \notin \mathcal{L}(\nu(e_1))$ or $w_2 \notin \mathcal{L}(\nu(e_2))$. But since we consider only simple expressions here, the latter holds if and only if there are valuations ν_1 for e_1 and ν_2 for e_2 such that for each $w_1w_2 \in \mathfrak{W}$ it is the case $w_1 \notin \mathcal{L}(\nu_1(e_1))$ or $w_2 \notin \mathcal{L}(\nu_2(e_2))$.
   Clearly, the latter holds if and only if there are valuations ν_1 for e_1 and ν_2 for e_2 and there are finite sets $\mathfrak{W}_1, \mathfrak{W}_2 \subset \Sigma^*$ such that: (1) For each $w_1w_2 \in \mathfrak{W}$ it is the case that $w_1 \in \mathfrak{W}_1$ or $w_2 \in \mathfrak{W}_2$, and (2) no word $w_1 \in \mathfrak{W}_1$ belongs to $\mathcal{L}(\nu_1(e_1))$, and (3) no word $w_2 \in \mathfrak{W}_2$ belongs to $\mathcal{L}(\nu_2(e_2))$. By the inductive hypothesis, the latter holds if and only if there are finite sets $\mathfrak{W}_1, \mathfrak{W}_2 \subset \Sigma^*$ such that for each word $w_1w_2 \in \mathfrak{W}$ it is the case that $w_1 \in \mathfrak{W}_1$ or $w_2 \in \mathfrak{W}_2$, and both $\texttt{CheckSimpleNonMemb}_\Box(e_1, \mathfrak{W}_1)$ and $\texttt{CheckSimpleNonMemb}_\Box(e_2, \mathfrak{W}_2)$ accept. By definition, the latter is equivalent to the fact that $\texttt{CheckSimpleNonMemb}_\Box(e, \mathfrak{W})$ accepts.
4. Assume e is of the form $(e_1)^*$. Then there is a valuation ν for e such that no word $w \in \mathfrak{W}$ belongs to $\mathcal{L}(\nu(e))$ if and only if $\varepsilon \notin \mathfrak{W}$ and there is a valuation ν_1 for e_1 such that for each $w_1, w_2, \ldots, w_k \in \Sigma^+$, if $w_1w_2 \cdots w_k \in \mathfrak{W}$ then some w_i $(1 \leq i \leq k)$ does not belong to $\mathcal{L}(\nu_1(e_1))$. Clearly, the latter holds if and only if $\varepsilon \notin \mathfrak{W}$ and there is a valuation ν_1 for e_1 and a finite set $\mathfrak{W}_1 \subset \Sigma^*$ such that: (1) For each $w_1, w_2, \ldots, w_k \in \Sigma^+$, if $w_1w_2 \cdots w_k \in \mathfrak{W}$ then some w_i $(1 \leq i \leq k)$ belongs to $\mathfrak{W}_1$, and (2) no word from $\mathfrak{W}_1$ belongs to $\mathcal{L}(\nu_1(e_1))$. By the inductive hypothesis, the latter holds if and only if $\varepsilon \notin \mathfrak{W}$ and $\texttt{CheckSimpleNonMemb}_\Box(e_1, \mathfrak{W}_1)$ accepts for some finite set $\mathfrak{W}_1 \subset \Sigma^*$ that satisfies the following: For each $w_1, w_2, \ldots, w_k \in \Sigma^+$, if $w_1w_2 \cdots w_k \in \mathfrak{W}$ then some w_i $(1 \leq i \leq k)$ belongs to $\mathfrak{W}_1$. By definition this is equivalent to the fact that $\texttt{CheckSimpleNonMemb}_\Box(e, \mathfrak{W})$ accepts.

Next we show that there is an implementation of the procedure $\texttt{CheckSimpleNonMemb}_\square$ that works in $O(|e|)$ time, if we assume that the input consists of a simple parameterized regular expression e and a *fixed* set of words $\mathfrak{W}$.

The implementation works recursively as follows: If e is of the form ε, or a, for $a \in \Sigma$, or $x \in \mathcal{V}$, or $e_1 \cup e_2$, then we implement recursively in the same way as it is described in $\texttt{CheckSimpleNonMemb}_\square$. If, on the other hand, e is of the form $e_1 e_2$ or $(e_1)^*$, then we have to be slightly more careful since we have to define how to search for sets $\mathfrak{W}_1$ and $\mathfrak{W}_2$. We do this as follows:

1. Assume first that e is of the form $e_1 e_2$. Then $\texttt{CheckSimpleNonMemb}_\square$ accepts e and $\mathfrak{W}$ if and only if there are finite sets $\mathfrak{W}_1, \mathfrak{W}_2 \subset \Sigma^*$ such that: (1) If $w_1 w_2 \in \mathfrak{W}$, then $w_1 \in \mathfrak{W}_1$ or $w_2 \in \mathfrak{W}_2$, (2) $\texttt{CheckSimpleNonMemb}_\square(e_1, \mathfrak{W}_1)$ accepts, and (3) $\texttt{CheckSimpleNonMemb}_\square(e_2, \mathfrak{W}_2)$ accepts. Our implementation, however, does not look over arbitrary sets $\mathfrak{W}_1$ and $\mathfrak{W}_2$, but only over the sets which can be constructed as follows: For each $w \in \mathfrak{W}$ and for each $w_1, w_2 \in \Sigma^*$ such that $w = w_1 w_2$, either pick up w_1 and place it in $\mathfrak{W}_1$ or pick up w_2 and place it in $\mathfrak{W}_2$. If for some pair $(\mathfrak{W}_1, \mathfrak{W}_2)$ constructed in this way it is the case that $\texttt{CheckSimpleNonMemb}_\square(e_1, \mathfrak{W}_1)$ accepts and $\texttt{CheckSimpleNonMemb}_\square(e_2, \mathfrak{W}_2)$ accepts, then $\texttt{CheckSimpleNonMemb}_\square(e, \mathfrak{W})$ accepts.
2. Assume second that e is of the form $(e_1)^*$. Then $\texttt{CheckSimpleNonMemb}_\square$ accepts if and only if $\varepsilon \notin \mathfrak{W}$ and there is a finite set $\mathfrak{W}_1 \subset \Sigma^*$ such that: (1) For each $w_1, w_2, \ldots, w_k \in \Sigma^+$, if $w_1 w_2 \cdots w_k \in \mathfrak{W}$ then some w_i $(1 \leq i \leq k)$ belongs to $\mathfrak{W}_1$, and (2) $\texttt{CheckSimpleNonMemb}_\square(e_1, \mathfrak{W}_1)$ accepts. Again, our implementation does not look over arbitrary sets $\mathfrak{W}_1$, but only over the sets which can be constructed as follows: For each decomposition $w_1 w_2 \cdots w_k$ of a word in $\mathfrak{W}$, where $w_i \in \Sigma^+$ for each $1 \leq i \leq k$, pick up an arbitrary $1 \leq i \leq k$ and place w_i in $\mathfrak{W}_1$. If for some set $\mathfrak{W}_1$ constructed in this way $\texttt{CheckSimpleNonMemb}_\square(e_1, \mathfrak{W}_1)$ accepts, then $\texttt{CheckSimpleNonMemb}_\square(e, \mathfrak{W})$ accepts. Clearly, our implementation continues being sound and complete.

To estimate the time complexity of the above implementation, first we need to see that all elements of all $\mathfrak{W}$ encountered in the algorithm are subwords of w and thus every $\mathfrak{W}$ belongs to $\mathbb{W}$, where $\mathbb{W}$ is the powerset of all subwords of w. Clearly the size of $\mathbb{W}$ is dependent only on $|w|$. Also the number of cases tried by each subcall of $\texttt{CheckSimpleNonMemb}_\square$ and the number of steps needed to construct each $\mathfrak{W}_1$ (and $\mathfrak{W}_2$ respectively) is dependent only on $|w|$. Hence all these values are constant.

If along the algorithm we memorize the answers to subcalls, then our complexity will be upper-bounded by the complexity of a dynamic version of the above algorithm, which would calculate $\texttt{CheckSimpleNonMemb}_\square(e_1, \mathfrak{W}_1)$ for all subexpressions e_1 of e' and all $\mathfrak{W}_1 \in \mathbb{W}$ in a bottom-up order. In this approach, the computation of $\texttt{CheckSimpleNonMemb}_\square(e_1, \mathfrak{W}_1)$ would take constant time, because answers to subcalls would have been precomputed. Thus the total complexity of $\texttt{CheckSimpleNonMemb}_\square$ is linear with respect to the size of the parse tree of e' and thus with respect to e as well.

(3) We finally prove that, for each word $w \in \Sigma^*$, the problem $\textsc{Membership}_\diamond(w)$ can be solved in time $O(n \log^2 n)$. Obviously, we can consider all labels $a \in \Sigma$ which do not occur

in w as equal. This simple observation makes the alphabet size fixed: $|\Sigma| \leq |w| + 1$. Now, let w be a word over Σ. Next we construct an algorithm that, given a parameterized regular expression e over Σ, checks whether $w \in \mathcal{L}_\diamond(e)$.

Assume that $\mathcal{W} \subset \mathcal{V}$ is the set of variables that appear in e. Using techniques from [24] the algorithm first constructs an NFA $\mathcal{A}$ over $\Sigma \cup \mathcal{W}$ that is equivalent to e, with $O(n)$ states and $O(n \log^2 n)$ transitions, and then performs a nondeterministic logarithmic space algorithm on $\mathcal{A}$. Let us assume, without loss of generality, that q_0 is the unique initial state of $\mathcal{A}$. Further, assume that $w = a_1 a_2 \cdots a_m$, where each a_i $(1 \leq i \leq m)$ is a symbol in Σ. Then we perform the following nondeterministic algorithm over $\mathcal{A}$: The algorithm works in at most $m+1$ steps. At each step $0 \leq i \leq m$ the state of the algorithm consists of a pair (q_i, μ_i), where $q_i \in Q$ and μ_i is a mapping from some subset $\mathcal{W}_i$ of $\mathcal{W}$ into Σ. The initial state of the algorithm is (q_0, μ_0), where $\mu_0 : \emptyset \to \Sigma$ (recall that q_0 is the initial state of $\mathcal{A}$). Assume that the state of the algorithm in step $i < m$ is (q_i, μ_i). Then in step $i+1$ the algorithm nondeterministically picks up a pair (q_{i+1}, μ_{i+1}) and checks that at least one of the following conditions holds:

- There exists a transition labeled $a_i \in \Sigma$ from q_i to q_{i+1} in $\mathcal{A}$ and $\mu_i = \mu_{i+1}$; that is, both μ_i and μ_{i+1} are mappings from $\mathcal{W}_i$ into Σ, and $\mu_{i+1}(x) = \mu_i(x)$, for each $x \in \mathcal{W}_i$.
- There exists a transition labeled $x \in \mathcal{V}$ from q_i to q_{i+1} in $\mathcal{A}$, $x \notin \mathcal{W}_i$ and $\mu_{i+1} : \mathcal{W}_i \cup \{x\} \to \Sigma$ is defined as follows: $\mu_{i+1}(y) = \mu_i(y)$, for each $y \in \mathcal{W}_i$, and $\mu_{i+1}(x) = a_i$.
- There exists a transition labeled $x \in \mathcal{V}$ from q_i to q_{i+1} in $\mathcal{A}$, $x \in \mathcal{W}_i$, $\mu_i(x) = a_i$ and $\mu_i = \mu_{i+1}$.

The procedure accepts if it reaches step n in state (q_n, μ_n), for some accepting state q_n of $\mathcal{A}$. Notice that, since w is fixed, the size of each mapping μ from a subset of $\mathcal{W}$ into Σ is also fixed: $|\mu| \leq \min\{|w|, |\mathcal{W}|\}$. That is because the initial mapping is empty and in each step of the algorithm it can grow only by one. This means that the nondeterministic procedure described above works in NLogspace.

It is not hard to prove (esentially using the same techniques than in the second part of the proof of Proposition 4) that the procedure described above accepts the parameterized regular expression e if and only if $w \in \mathcal{L}_\diamond(e)$.

Now let M be the set of all mappings μ from subsets of $\mathcal{W}$ to Σ, which can occur in the algorithm presented above, and V be the number of all states (q, μ), which can occur therein. To see the precise time complexity, we need to estimate $|M|$ and $|V|$. First, $|M| = O\left(|\Sigma|^{\min\{|w|,|\mathcal{W}|\}}\right)$, which is fixed in our case. Then, $V = O(n \cdot |M|)$, which is linear in the number of states of $\mathcal{A}$.

Now let us imagine a directed graph G with the set of vertices V, in which there is an edge from state (q, μ) to state (q', μ') if and only if the pair (q', μ') can be picked up from pair (q, μ) according to the algorithm presented above. Each edge $(q, \mu) \to (q', \mu')$ corresponds to an edge $q \to q'$ in $\mathcal{A}$ and for each edge $q \to q'$ in $\mathcal{A}$ there are at most $|M|$ edges $(q, \mu) \to (q', \mu')$ (one for each $\mu \in M$). Therefore, G has $O(n \log^2 n)$ edges, since $|M|$ is fixed. We can also construct G in $O(n \log^2 n)$ time and space.

Finally, it suffices to perform a reachability search in G to see whether an accepting state can be reached from node (q_0, μ_0) in G, which can clearly be done in linear time with respect to the size of G, which gives us an algorithm with $O(n \log^2 n)$ time complexity or, by dropping the assumption of w being fixed — $O\left(|w| \cdot |\Sigma|^{\min\{|w|,|\mathcal{W}|\}} \cdot n \log^2 n\right)$ time complexity. Moreover, we can spare space by not constructing G and by computing it "on the fly", because standard graph search algorithms run in $O(V)$ space. It might be also useful in terms of time, because a fixed word w either attains an accepting state within a short path or does not do so at all, so usually most parts of G would not be touched by the search algorithm at all.

It is worth analysing the gain in performance, which the above method gives in comparison to the direct approach. The straightforward algorithm calculates an NFA accepting $\mathcal{L}_\diamond(e)$ and runs reachability search on it. The size of such NFA is $O(|\Sigma|^{|\mathcal{W}|})$, so the time complexity becomes $O(|w| \cdot |\Sigma|^{|\mathcal{W}|} \cdot n \log^2 n)$. Hence, the only gain, that the former algorithm gives is lowering the exponent over Σ from $|\mathcal{W}|$ to $\min\{|w|, |\mathcal{W}|\}$ and this is because it takes into advantage a smaller class of mappings, confined by the length of the run of w on $\mathcal{A}$. In fact, this gain is very large if we speak of problem instances with a relatively small w and a huge e. □

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 standard 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 [17]). The lower bound proof is again by a generic reduction.

Theorem 6.
- *The problem* UNIVERSALITY$_\Box$ *is* PSPACE-*complete.*
- *The problem* UNIVERSALITY$_\diamond$ *is* EXPSPACE-*complete.*

Proof: We only need to show the second part. We begin with the upper bound. It is well known that there is an algorithm to decide whether the language of a standard regular expression e (that is, without variables) is universal, that requires polynomial space with respect to the size of the input expression e. Given a parameterized regular expression e', construct the regular expression $e = \bigcup\{\nu(e') \mid \nu \text{ is a valuation for } e'\}$ without variables. Clearly, $\mathcal{L}_\diamond(e') = \mathcal{L}(e)$. We can then decide universality of $\mathcal{L}_\diamond(e')$ by first computing the regular expression e, and then checking if $\mathcal{L}(e)$ is universal using the standard algorithm for regular expressions without variables. Since the expression e is of size exponential with respect to the original expression e' (the number of possible valuations for e' is $|\Sigma|^{|\mathcal{W}|}$, where $\mathcal{W}$ is the set of variables in e'), the above procedure runs in EXPSPACE.

For the lower bound we present a reduction from the complement of the acceptance problem of a Turing machine. Let L be a language that belongs to EXPSPACE, and let $\mathcal{M}$ be a Turing machine that decides L in EXPSPACE. Given an input $\bar{a}$ we construct in polynomial time with respect to $\mathcal{M}$ and $\bar{a}$ a parameterized regular expression $e_{\mathcal{M},\bar{a}}$ over some alphabet Δ such that $\mathcal{L}_\diamond(e_{\mathcal{M},\bar{a}})$ consists of all the strings over Δ if and only if $\mathcal{M}$ does not accept input $\bar{a}$.

Just as in section 4.1, we assume that $\mathcal{M} = (Q, \Gamma, q_0, \{q_m\}, \delta)$, where $Q = \{q_0, \ldots, q_m\}$ is the set of states, Γ is the tape alphabet (that contains the distinguished *blank* symbol B), the initial state is q_0, q_m is the unique final state, and $\delta : (Q \setminus \{q_m\}) \times \Gamma \to Q \times \Gamma \times \{L, R\}$ is the transition function. Notice that we assume without loss of generality that no transition is defined on the final state q_m. Since $\mathcal{M}$ decides L in EXPSPACE, there is a polynomial $S()$ such that, for every input $\bar{a}$ over Σ, $\mathcal{M}$ decides $\bar{a}$ using space of order $2^{S(|\bar{a}|)}$.

Let $\bar{a} = a_0 a_1 \cdots a_{k-1} \in \Sigma^*$ be an input for $\mathcal{M}$ (that is, each a_i, $0 \leq i \leq k-1$ is a symbol in Γ). For notational convenience we will assume from now on that $S(|\bar{a}|) = n$.

We also find it convenient to introduce the following notation. For an alphabet $\Sigma = \{b_1, \ldots, b_p\}$ of symbols, we abuse notation and denote by Σ the regular expression $(b_1 \mid \cdots \mid b_p)$. Thus, for example, assume that $\Gamma = \{b_1, \ldots, b_p\} \cup \{B\}$. Then, when we write $(\Gamma \cup (\Gamma \times Q))$ we represent the language given by $(b_1 \mid \cdots \mid b_p \mid B \mid (b_1, q_0) \mid \cdots \mid (B, q_m))$. Furthermore, we reuse the notation introduced in Section 4.1, and write the shorthand $[i]$ to denote the binary representation of the number i as a string of n characters (i.e., $[0]$ corresponds to the word 0^n, and $[2]$ corresponds to the word $0^{n-2}10$).

Our parameterized expression $e_{\mathcal{M},\bar{a}}$ uses the alphabet $\Delta = \{0, 1, \#, \&, \%\} \cup \Gamma \cup (\Gamma \times Q)$. The idea of the reduction is as follows. Using Δ, we represent a configuration of $\mathcal{M}$ by words in Δ^* of the form:

$$\begin{array}{l} \# \cdot [0] \cdot (\Gamma \cup (\Gamma \times Q)) \cdot \& \cdot \\ \qquad [1] \cdot (\Gamma \cup (\Gamma \times Q)) \cdot \& \cdot \\ \qquad\qquad [2] \cdot (\Gamma \cup (\Gamma \times Q)) \cdot \& \\ \qquad\qquad\qquad \vdots \\ \qquad\qquad\qquad [2^n - 1] \cdot (\Gamma \cup (\Gamma \times Q)) \cdot \& \cdot \% \end{array} \tag{3}$$

Intuitively, the strings $[0], [1], \ldots, [2^n - 1]$ indicate each one of the 2^n cells of $\mathcal{M}$, and the symbol following these strings represents either the content of the cell, or the content of the cell plus the state of $\mathcal{M}$, if $\mathcal{M}$ is pointing into that particular cell of the tape in the configuration that is being encoded.

Since each word of the form (3) represents a configuration of $\mathcal{M}$, we can represent a run of $\mathcal{M}$ on input $\bar{a}$ as a sequence of concatenations of words of the form (3), as long as each one of these configurations is consistent with the computation of $\mathcal{M}$. The idea of the reduction is to construct a parameterized regular expression $e_{\mathcal{M},\bar{a}}$ that represent all words w in Δ^* that are either *not* valid concatenations of subwords of the form (3), or, in case that they are, that the sequence of configurations represented by w is not a valid run of $\mathcal{M}$ on input $\bar{a}$. In other words, any word in Δ^* that does not belong to $\mathcal{L}_\diamond(e_{\mathcal{M},\bar{a}})$ is bound to represent a valid run of $\mathcal{M}$ over input $\bar{a}$, thus obtaining that $\mathcal{L}_\diamond(e) \neq \Delta^*$ if and only if $\mathcal{M}$ accepts on input $\bar{a}$.

We split the definition of $e_{\mathcal{M},\bar{a}}$ into five parts: $e_{\mathcal{M},\bar{a}} = e^1 \mid e^2 \mid e^3 \mid e^4 \mid e^5$, where:

- e^1 describes all the words that are not concatenations of subwords of form (3).
- e^2 describes all the words that, even if they are concatenations of subwords of form (3), these subwords do not represent valid configurations of $\mathcal{M}$.
- e^3 describes words that do not start with a subword of form (3) correctly describing the initial configuration of $\mathcal{M}$ over input $\bar{a}$.
- e^4 describes words whose final subword of form (3) does not contain any final states (and is therefore not a final configuration of $\mathcal{M}$).
- e^5 describes words that contains two consecutive subwords of form (3) that represent configurations α and β for $\mathcal{M}$ such that α and β do not agree on δ.

Next we show how to construct expressions e^1, e^2, e^3, e^4, e^5. We do not provide the precise details of e^1, since it is straightforward to define it from (3). Expression e^2 is defined as the union of the following two expressions, stating that:

- Between a symbol # and % there is no symbol in $(\Gamma \times Q)$ (in other words, the machine is pointing at none of the cells in that configuration):

$$e^2_1 = \Delta^* \cdot \# \cdot (\Delta \setminus (\{\%\} \cup \Gamma \times Q))^* \cdot \% \cdot \Delta^*$$

- Between a symbol # and % there is more than one symbol in $(\Gamma \times Q)$ (a configuration features two positions being read by the machine):

$$e^2_2 = \Delta^* \cdot \# \cdot (\Delta \setminus \{\%\})^* \cdot (\Gamma \times Q) \cdot (\Delta \setminus \{\%\})^* \cdot (\Gamma \times Q) \cdot (\Delta \setminus \{\%\})^* \cdot \% \cdot \Delta^*$$

Expression e^3 is the union of the following expressions, describing that:

- The first configuration does not contain the initial state in the first position of the tape, reading the first symbol of the output:

$$e^3_1 = \# \cdot [0] \cdot \Gamma \cup \big((\Gamma \times Q) \setminus \{(q_0, a_0)\}\big) \cdot \Delta^*$$

- The rest of the $k-1$ symbols of the tape do not agree with the input:

$$\begin{aligned}
e^3_2 &= \# \cdot [0] \cdot (\Gamma \cup (\Gamma \times Q)) \cdot \& \cdot [1] \cdot (\Gamma \cup (\Gamma \times Q) \setminus \{a_1\}) \cdot \Delta^* \\
\vdots &= \vdots \\
e^3_k &= \# \cdot [0] \cdot (\Gamma \cup (\Gamma \times Q)) \cdot \& \cdots [k-1] \cdot (\Gamma \cup (\Gamma \times Q) \setminus \{a_{k-1}\}) \cdot \Delta^*
\end{aligned}$$

- The rest of the symbols of the first configuration are not blank symbols:

$$e^3_{k+1} = \# \cdot [0] \cdot \Delta \cdot \& \cdots [k-1] \cdot \Delta \cdot \& \cdot (\Delta \setminus \{\%\})^* \cdot (0 \mid 1)^n \cdot (\Gamma \cup (\Gamma \times Q) \setminus \{B\}) \cdot \Delta^*$$

Expression e^4 describes words whose final configuration is not in a final state:

$$e^4 = \Delta^* \cdot \# \cdot (\Delta \setminus \{\%\})^* \cdot \big(\Gamma \times (Q \setminus \{q_m\})\big) \cdot (\Delta \setminus \{\%\})^* \cdot \%$$

Finally, we describe expression e^5. Intuitively, it describes words that feature two subsequent configurations that are not consistent with each other. More precisely, it is the union of the following expressions, stating that:

- A cell not pointed by the head changed its content from one configuration to the subsequent one:

$$e^5_1 = \bigcup_{a \in \Gamma} \Delta^* \cdot x_1 \cdots x_n \cdot a \cdot \& \cdot (\Delta \setminus \{\%\})^* \cdot \% \cdot \\ \# \cdot \big((0|1)^n \cdot (\Gamma \cup (\Gamma \times Q)) \cdot \&\big)^* \cdot x_1 \cdots x_n \cdot \big((\Gamma \setminus \{a\}) \cup ((\Gamma \setminus \{a\}) \times Q)\big) \cdot \Delta^*$$

- A configuration that is not final features a pair in $Q \times \Sigma$ for which no transition is defined (the symbol # states the configuration is not the final one):

$$e^5_2 = \bigcup_{\{(a,q) \ \mid \ \delta(q,a) \text{ is not defined}\}} \Delta^* \cdot (a,q) \cdot \Delta^* \cdot \# \cdot \Delta^*$$

- The change of state does not agree with δ:

$$e^5_3 = \bigcup_{\{(a,q) \ \mid \ \delta(q,a)=(q',a',\{L,R\})\}} \Delta^* \cdot (a,q) \cdot (\Delta \setminus \{\%\})^* \cdot \% \cdot \\ \# \cdot (\Delta \setminus \{\%\})^* \cdot (\Gamma \times (Q \setminus \{q'\})) \cdot \Delta^*$$

- The symbol written in a given step does not agree with δ:

$$e^5_4 = \bigcup_{\{(a,q) \ \mid \ \delta(q,a)=(q',a',\{L,R\})\}} \Delta^* \cdot y_1 \cdots y_n \cdot (a,q) \cdot (\Delta \setminus \{\%\})^* \cdot \% \cdot \\ (\Delta \setminus \{\%\})^* \cdot y_1 \cdots y_n \cdot (\Gamma \setminus \{a'\}) \cdot \Delta^*$$

- The movement of the head does not agree with δ:

$$e^5_5 = \bigcup_{\{(a,q) \ \mid \ \delta(q,a)=(q',a',R)\}} \Delta^* \cdot z_1 \cdots z_n \cdot (a,q) \cdot (\Delta \setminus \{\%\})^* \cdot \% \cdot \\ (\Delta \setminus \{\%\})^* \cdot z_1 \cdots z_n \cdot a' \cdot \& \cdot \big(\varepsilon \mid ((0|1)^n \cdot \Gamma \cdot (\Delta \setminus \{\%\}))^*\big) \cdot \% \cdot \Delta^*$$

$$e_6^5 = \bigcup_{(a,q)\ |\ \delta(q,a)=(q',a',L)} \Delta^* \cdot w_1 \cdots w_n \cdot (a,q) \cdot (\Delta \setminus \{\%\})^* \cdot \% \cdot$$
$$\# \cdot \big(\varepsilon \mid ((\Delta \setminus \{\%\})^* \cdot (0|1)^n \cdot \Gamma \cdot \&)\big) \cdot w_1 \cdots w_n \cdot \Delta^*$$

Having defined $e_{\mathcal{M},\bar{a}}$, it is now straightforward to show that $\mathcal{L}_\diamond(e_{\mathcal{M},\bar{a}}) = \Delta^*$ if and only if $\mathcal{M}$ does not accept on input $\bar{a}$. This finishes the proof of the Expspace lower bound. □

Similarly to the nonemptiness problem (studied in Section 4.1), the Expspace bound for Universality$_\diamond$ is quite resilient, as it holds even 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 7. *The problem* Universality$_\diamond$ *remains* Expspace*-hard over the class of simple parameterized regular expressions.*

Proof: We sketch how to adapt the reduction of Theorem 6 to hold for simple parameterized regular expressions (i.e. without repetitions of variables).

Recall that the previous reduction used the alphabet $\{0, 1, \%, \#, \&\} \cup \Gamma \cup (\Gamma \times Q)$. In this case, we need a slightly bigger alphabet. Let $\Delta = \{0, 1, \&, \#_{\text{even}}, \%_{\text{even}}, \#_{\text{odd}}, \%_{\text{odd}}\} \cup \Gamma \cup (\Gamma \times Q)$. The idea is to modify the way configurations are represented.

Previously, we had that runs of $\mathcal{M}$ were represented by words in the language:

$$\big(\# \cdot [0] \cdot (\Gamma \cup (\Gamma \times Q)) \cdot \& \cdots [2^n - 1] \cdot (\Gamma \cup (\Gamma \times Q)) \cdot \& \cdot \%\big)^*.$$

We modify the coding, so that configurations are represented in the following way:

$$\big(\#_{\text{even}} \cdot [0] \cdot (\Gamma \cup (\Gamma \times Q)) \cdot \& \cdots [2^n - 1] \cdot (\Gamma \cup (\Gamma \times Q)) \cdot \& \cdot \%_{\text{even}} \cdot$$
$$\#_{\text{odd}} \cdot [0] \cdot (\Gamma \cup (\Gamma \times Q)) \cdot \& \cdots [2^n - 1] \cdot (\Gamma \cup (\Gamma \times Q)) \cdot \& \cdot \%_{\text{odd}}\big)^*$$

The intuition is that configurations using $\#_{\text{even}}$ and $\%_{\text{even}}$ represent an even step of the computation of the Turing machine, whereas configurations using $\#_{\text{odd}}$ and $\%_{\text{odd}}$ represent an odd step. Notice that one can assume, without loss of generality, that the run of $\mathcal{M}$ over input $\bar{a}$ ends after an odd number of computations.

All that remain to do is to adapt the definition of the expression $e_{\mathcal{M},\bar{a}} = e^1 \mid \cdots \mid e^5$ so that it works under this modified coding, and such that $e_{\mathcal{M},\bar{a}}$ is simple. We omit most of the details, since most of the expressions in $e^1, \ldots, e^5$ do not use parameters, and thus are not difficult to modify.

To see how the expressions using parameters can be modified so that they are simple, we show how to adapt the expression e_1^5, that intuitively accepts all words describing two configurations in which a cell not pointed by the head changed its content. It was defined previously as

$$e_1^5 = \bigcup_{a \in \Gamma} \Delta^* \cdot x_1 \cdots x_n \cdot a \cdot \& \cdot (\Delta \setminus \{\%\})^* \cdot \% \cdot$$
$$\# \cdot \big((0|1)^n \cdot (\Gamma \cup (\Gamma \times Q)) \cdot \&\big)^* \cdot x_1 \cdots x_n \cdot \big((\Gamma \setminus \{a\}) \cup ((\Gamma \setminus \{a\}) \times Q)\big) \cdot \Delta^*$$

A straightforward idea is to explicitly state even - odd and odd - even cases, that is, redefine e_1^5 as $e_{1,e}^5 \mid e_{1,o}^5$, where

$$e_{1,e}^5 = \bigcup_{a\in\Gamma} \Delta^* \cdot x_1 \cdots x_n \cdot a \cdot \& \cdot (\Delta \setminus \{\%_{\text{even}}, \%_{\text{odd}}\})^* \cdot \%_{\text{even}} \cdot \\ \#_{\text{odd}} \cdot \big((0|1)^n \cdot (\Gamma \cup (\Gamma \times Q)) \cdot \&\big)^* \cdot x_1 \cdots x_n \cdot \big((\Gamma \setminus \{a\}) \cup ((\Gamma \setminus \{a\}) \times Q)\big) \cdot \Delta^*$$

$$e_{1,o}^5 = \bigcup_{a\in\Gamma} \Delta^* \cdot y_1 \cdots y_n \cdot a \cdot \& \cdot (\Delta \setminus \{\%_{\text{even}}, \%_{\text{odd}}\})^* \cdot \%_{\text{odd}} \cdot \\ \#_{\text{even}} \cdot \big((0|1)^n \cdot (\Gamma \cup (\Gamma \times Q)) \cdot \&\big)^* \cdot y_1 \cdots y_n \cdot \big((\Gamma \setminus \{a\}) \cup ((\Gamma \setminus \{a\}) \times Q)\big) \cdot \Delta^*$$

The problem is that these expressions are not simple: they reuse the variables $x_1, \ldots, x_n$ or $y_1, \ldots, y_n$. The solution is instead a bit more technical. We redefine e_1^5 as $e_{1,e}^6 \mid e_{1,o}^6$, where:

$$e_{1,e}^6 = \bigcup_{a\in\Gamma} \Delta^* \#_{\text{even}} \cdot (\Delta \setminus \{\%_{\text{even}}\})^* \cdot \\ \Bigg(x_1 \cdots x_n \cdot \Bigg(\big(a \cdot \& \cdot (\Delta \setminus \{\%_{\text{even}}\})^* \cdot \%_{\text{even}} \cdot \#_{\text{odd}} (\Delta \setminus \{\%_{\text{odd}}\})^* \cdot \&\big) \mid \\ \big(((\Gamma \setminus \{a\}) \cup ((\Gamma \setminus \{a\}) \times Q)) \cdot (\Delta \setminus \{\%_{\text{odd}}, \#_{\text{even}}, \%_{\text{even}}\})^*\big)\Bigg)\Bigg)^* \cdot \%_{\text{odd}} \cdot \Delta^*$$

$$e_{1,o}^6 = \bigcup_{a\in\Gamma} \Delta^* \#_{\text{odd}} \cdot (\Delta \setminus \{\%_{\text{odd}}\})^* \cdot \\ \Bigg(y_1 \cdots y_n \Bigg(\big(a \cdot \& \cdot (\Delta \setminus \{\%_{\text{odd}}\})^* \cdot \%_{\text{odd}} \cdot \#_{\text{even}} (\Delta \setminus \{\#_{\text{even}}\})^* \cdot \&\big) \mid \\ \big(((\Gamma \setminus \{a\}) \cup ((\Gamma \setminus \{a\}) \times Q)) \cdot (\Delta \setminus \{\%_{\text{even}}, \#_{\text{odd}}, \%_{\text{odd}}\})^*\big)\Bigg)\Bigg)^* \cdot \%_{\text{even}} \cdot \Delta^*$$

Notice then that $e_{1,e}^6 \mid e_{1,o}^6$ is a simple parameterized regular expression. In order to see that the intended meaning of these expressions remains the same, notice that $\mathcal{L}_\diamond(e_{1,e}^5) \subseteq \mathcal{L}_\diamond(e_{1,e}^6)$ and $\mathcal{L}_\diamond(e_{1,o}^5) \subseteq \mathcal{L}_\diamond(e_{1,o}^6)$. Moreover, it is not difficult to check that none of the words that belong to $\mathcal{L}_\diamond(e_{1,e}^6)$ but not to $\mathcal{L}_\diamond(e_{1,e}^5)$ represent a valid run of $\mathcal{M}$, and neither does any word in $\mathcal{L}_\diamond(e_{1,o}^6)$ but not in $\mathcal{L}_\diamond(e_{1,o}^5)$. Thus, the words in $\mathcal{L}_\diamond(e_{1,e}^6)$ but not in $\mathcal{L}_\diamond(e_{1,e}^5)$ (respectively, in $\mathcal{L}_\diamond(e_{1,o}^6)$ but not in $\mathcal{L}_\diamond(e_{1,o}^5)$) are not harmful for our purposes, since these extra words already belong to the language of some other disjunction in $e_{\mathcal{M},\bar{a}}$.

With these observations, it is not difficult to modify the remainder of the reduction of the proof of Theorem 6 so that every expression is simple. The proof then follows along the same lines as the proof of Theorem 6. □

4.4. Containment

Recall that the CONTAINMENT problem asks, given parameterized regular expressions e_1 and e_2, whether $\mathcal{L}_\Box(e_1) \subseteq \mathcal{L}_\Box(e_2)$ or $\mathcal{L}_\Diamond(e_1) \subseteq \mathcal{L}_\Diamond(e_2)$ holds. 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 8. *Both* CONTAINMENT$_\Box$ *and* CONTAINMENT$_\Diamond$ *are* EXPSPACE*-complete.*

Proof: Since $\Sigma^* \subseteq \mathcal{L}_\Diamond(e)$ iff UNIVERSALITY$_\Diamond(e)$ is true, and $\mathcal{L}_\Box(e) \subseteq \emptyset$ iff NONEMPTINESS$_\Box(e)$ is false, we get 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 EXPSPACE, since one can construct exponentially many automata for $\mathcal{L}(\nu(e_1))$ in 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 EXPSPACE. Since this needs to be done for exponentially many valuations ν', the overall 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: CONTAINMENT$_*(e_1, \cdot)$ asks, for a parameterized regular expression e_2, whether $\mathcal{L}_*(e_1) \subseteq \mathcal{L}_*(e_2)$; and CONTAINMENT$_*(\cdot, e_2)$ is defined similarly. The reductions proving Theorem 8 show that CONTAINMENT$_\Box(\cdot, e_2)$ and CONTAINMENT$_\Diamond(e_1, \cdot)$ remain 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 9.
- CONTAINMENT$_\Box(e_1, \cdot)$ *is* PSPACE*-complete.*
- CONTAINMENT$_\Diamond(\cdot, e_2)$ *is* coNP*-complete.*

Proof: (Part 1) It is well known that CONTAINMENT$_\Box(e_1, \cdot)$ is PSPACE-hard even for standard regular expressions. For the upper bound, let e_1' be an expression such that $\mathcal{L}(e_1') = \mathcal{L}_\Box(e_1)$. Since e_1 is fixed, the expression e_1' can be computed in constant time. Then, it suffices to guess a valuation ν and a word w such that $w \in \mathcal{L}(e_1')$, but $w \notin \mathcal{L}(\nu(e_2))$, which can clearly be done in PSPACE.

(Part 2) We begin with the upper bound for the problem CONTAINMENT$_\Diamond(\cdot, e_2)$. Assume that the input is a parameterized regular expression e_1 over Σ, and that $\mathcal{W} \subset \mathcal{V}$ is the set of variables mentioned in e_1. The following coNP algorithm solves the problem CONTAINMENT$_\Diamond(\cdot, e_2)$. First, construct a DFA $\mathcal{A}_{e_2}$ such that $\mathcal{L}(\mathcal{A}_{e_2}) = \mathcal{L}_\Diamond(e_2)$, and then construct $\mathcal{A}_{e_2}^C$, the DFA that accepts the complement of $\mathcal{L}(\mathcal{A}_{e_2})$. Since e_2 is fixed, $\mathcal{A}_{e_2}^C$ can be constructed in constant time. Next, guess a valuation $\nu : \mathcal{W} \to \Sigma$, and, from $\nu(e_1)$, construct the NFA $\mathcal{A}_{\nu(e_1)}$ that accepts $\mathcal{L}(\nu(e_1))$. It is well-known that this automaton can be constructed in polynomial time from $\nu(e_1)$. Finally, check that $\mathcal{A}_{\nu(e_1)} \cap \mathcal{A}_{e_2}^C \neq \emptyset$, which can be performed in polynomial time using a standard reachability test over the product of $\mathcal{A}_{\nu(e_1)}$ and $\mathcal{A}_{e_2}^C$. It is not hard to see that this algorithm is sound and complete for the problem. In fact, if $\mathcal{A}_{\nu(e_1)} \cap \mathcal{A}_{e_2}^C \neq \emptyset$, then there is a word $w \in \mathcal{L}(\nu(e_1))$, and hence in $\mathcal{L}_\diamond(e_1)$, that does not belong to $\mathcal{L}_\diamond(e_2)$. This implies that $\mathcal{L}_\diamond(e_1)$ is not contained in $\mathcal{L}_\diamond(e_2)$. On the other hand, it is clear that if $\mathcal{A}_{\nu(e_1)} \cap \mathcal{A}_{e_2}^C = \emptyset$ for all possible valuations ν from $\mathcal{W}$ to Σ, then $\mathcal{L}_\diamond(e_1)$ is contained in $\mathcal{L}_\diamond(e_2)$.

The lower bound is established via a reduction from 3-SAT to the complement of CONTAINMENT$_\square(\cdot, e$ where e_2 is the following regular expression over the alphabet $\Sigma = \{0, 1, \#\}$:

$$e_2 \ := \ \big((10 \mid 01)^*\#((0 \mid 1)^3)^*000((0 \mid 1)^3)^*\big) \mid \big(((0 \mid 1)^2)^*(00 \mid 11)\Sigma^*\big) \mid \big(\Sigma^*\#\Sigma^*\#\Sigma^*\big).$$

Notice that e_2 mentions no variables, and hence $\mathcal{L}_\diamond(e_2) = \mathcal{L}(e_2)$.

Let $\varphi = \bigwedge_{1 \leq i \leq n} (\ell_i^1 \vee \ell_i^2 \vee \ell_i^3)$ be a propositional formula in 3-CNF over variables $\{p_1, \dots, p_m\}$. That is, each literal ℓ_i^j, for $1 \leq i \leq n$ and $1 \leq j \leq 3$, is either p_k or $\neg p_k$, for $1 \leq k \leq m$. Next we show how to construct in polynomial time from φ a parameterized regular expression e_1 over the alphabet $\Sigma = \{0, 1, \#\}$ such that φ is satisfiable if and only if $\mathcal{L}_\diamond(e_1) \not\subseteq \mathcal{L}(e_2)$.

Let $\mathcal{W} = \{x_i, \hat{x}_i \mid 1 \leq i \leq m\}$. Intuitively, each x_i represents the value assigned to p_i, and $\hat{x}_i$ represents the value of $\neg p_i$. Moreover, assume that h is a mapping from the literals ℓ_i^j ($1 \leq i \leq n$ and $1 \leq j \leq 3$) to $\mathcal{W}$, defined as expected: $h(\ell_i^j) = x_k$ if ℓ_i^j is p_k, for some $1 \leq k \leq m$, and $h(\ell_i^j) = \hat{x}_k$ if ℓ_i^j is $\neg p_k$.

Define e_1 as follows:

$$e_1 = x_1\hat{x}_1 \cdots x_m\hat{x}_m \# h(\ell_1^1)h(\ell_1^2)h(\ell_1^3) \cdots h(\ell_n^1)h(\ell_n^2)h(\ell_n^3).$$

We show that φ is satisfiable if and only if $\mathcal{L}_\diamond(e_1) \not\subseteq \mathcal{L}(e_2)$.

($\Rightarrow$): Assume that φ is satisfiable by valuation σ. Let ν be a valuation from $\mathcal{W}$ to Σ, defined as follows:

- For each $1 \leq k \leq m$, $\nu(x_k) = 1$ if $\sigma(p_k) = 1$, and $\nu(x_k) = 0$ otherwise.
- For each $1 \leq k \leq m$, $\nu(\hat{x}_k) = 0$ if $\sigma(p_k) = 1$, and $\nu(\hat{x}_k) = 1$ otherwise.

Notice that $L(\nu(e_1))$ consists of the single word:

$$\nu(x_1)\nu(\hat{x}_1) \cdots \nu(x_m)\nu(\hat{x}_m)\#\nu(h(\ell_1^1))\nu(h(\ell_1^2))\nu(h(\ell_1^3)) \cdots \nu(h(\ell_n^1))\nu(h(\ell_n^2))\nu(h(\ell_n^3)).$$

We shall abuse notation and denote by $\nu(e_1)$ both this word and the aforementioned expression. It is clear that $\nu(e_1)$ contains a single symbol $\#$, and starts with a prefix in $(01 \mid 10)^*\#$. Thus, if $\mathcal{L}_\diamond(e_1) \subseteq \mathcal{L}(e_2)$ it must be that $\nu(e_1)$ is defined by the expression $(10 \mid 01)^*\#((0 \mid 1)^3)^*000((0 \mid 1)^3)^*$. But this implies that there are literals ℓ_i^1, ℓ_i^2 and ℓ_i^3, for some $1 \leq i \leq n$, such that $\nu(h(\ell_i^1))\nu(h(\ell_i^2))\nu(h(\ell_i^3)) = 000$. By construction of ν, it must be the case that σ falsifies the i-th clause of φ, which contradicts the fact that σ is a satisfying assignment.

($\Leftarrow$): Assume on the other hand that $\mathcal{L}_\diamond(e_1) \not\subseteq \mathcal{L}(e_2)$. From the definition of the $\diamond$-semantics, there is at least one valuation ν from $\mathcal{W}$ to Σ such that $\mathcal{L}(\nu(e_1)) \not\subseteq \mathcal{L}(e_2)$. Notice again that, by construction of e_1, $\nu(e_1)$ consists of the single word:

$$\nu(x_1)\nu(\hat{x}_1) \cdots \nu(x_m)\nu(\hat{x}_m)\#\nu(h(\ell_1^1))\nu(h(\ell_1^2))\nu(h(\ell_1^3)) \cdots \nu(h(\ell_n^1))\nu(h(\ell_n^2))\nu(h(\ell_n^3)).$$

Again, we shall denote this word also by $\nu(e_1)$. Then if $\mathcal{L}(\nu(e_1)) \not\subseteq \mathcal{L}(e_2)$ it must be the case that $\nu(e_1)$ is not in $\mathcal{L}(e_2)$. This immediately entails that $\nu(e_1)$ cannot have two or more copies of the symbol #, and thus we conclude that ν assigns to each variable $\mathcal{W}$ a symbol in $\{0, 1\}$. From this it follows that the following valuation σ for the variables in φ is well defined:

- $\sigma(p_i) = 1$ if $\nu(x_i) = 1$, and $\sigma(p_i) = 0$ if $\nu(x_i) = 0$

Next we show that for each $1 \leq i \leq m$, it is the case that $\nu(x_i) \neq \nu(\hat{x}_i)$. Assume for the sake of contradiction that for some $i \leq i \leq m$ we have $\nu(x_i) = \nu(\hat{x}_i)$. From the construction of e_1 it must be the case that $\nu(e_1)$ is denoted by the expression $((0 \mid 1)^2)^*(00 \mid 11)\Sigma^*$, which contradicts the fact that $\nu(e_1)$ is not in $\mathcal{L}(e_2)$. Finally, we claim that φ is satisfiable by the valuation σ. Assume the contrary. Then there is a clause $(\ell_i^1 \vee \ell_i^2 \vee \ell_i^3)$, for $1 \leq i \leq n$, such that, for each $1 \leq j \leq 3$, if ℓ_i^j is the literal p_k, for some $1 \leq k \leq m$, then σ assigns the value 0 to p_k, and if ℓ_i^j is the literal $\neg p_k$, for some $1 \leq k \leq m$, then σ assigns the value 1 to p_k. It is now straightforward to conclude that this fact contradicts the assumption that $\nu(e_1)$ is not in $\mathcal{L}(e_2)$, by studying all of the 8 possible cases. □

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 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 1. 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 3. Summing up, we have:

Corollary 2.
- *The problem* NonemptyIntReg$_\Box$ *is* Expspace-*complete.*
- *The problem* 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 ConstructNFA$_*$: Given a parameterized regular expression e, construct an NFA $\mathcal{A}$ over Σ such that $\mathcal{L}(\mathcal{A}) = \mathcal{L}_*(e)$.

Proposition 10. *The problem* ConstructNFA$_\Diamond$ *can be solved in single-exponential time, and the problem* 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 11. *The sizes of minimal NFAs are necessarily double-exponential for $\mathcal{L}_\Box$, and necessarily exponential for $\mathcal{L}_\diamond$.*

Proof: 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 $x_1, \dots, x_{n+1}$:

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

Notice that each e_n uses $n+1$ variables, and is of linear size in n. We first show a technical lemma:

Lemma 3. *Let $u \in \{0,1\}^{n+1}$ be a word of size $n+1$. Then u is a subword of every word $w \in \mathcal{L}_\Box(e_n)$. Moreover, there is a match for u in w that starts in a position j of w ($1 \leq j \leq |w|$) such that $j = 1 \mod n+1$.*

Proof: Consider an arbitrary word $u = u_1, \dots, u_{n+1} \in \{0,1\}^{n+1}$, and let ν be the valuation for e_n such that $\nu(x_i) = u_i$, for $1 \leq i \leq (n+1)$. Then $\nu(e_n) = ((0 \mid 1)^{n+1})^* \cdot u \cdot ((0 \mid 1)^{n+1})^*$, and thus all words w in $\mathcal{L}(\nu(e_n))$ contain u as a subword, matching in a position $j = 1 \mod n+1$ of w. The lemma follows since by definition $\mathcal{L}_\Box(e_n) \subseteq \mathcal{L}(\nu(e_n))$. □

In order to show that every NFA deciding $\mathcal{L}_\Box(e_n)$ has 2^{2^n} states, we use the following result:

Theorem 12. *[25] 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 $P_n = \{(w_S, w_{\bar{S},n}) \mid S \subset \{0,1\}^{n+1} \text{ and } |S| = 2^n\}$. 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. Next, we show that $\mathcal{L}_\Box(e_n)$ and P_n satisfy properties (1) and (2) in Theorem 12, which proves the double exponential lower bound.

1. We need to show that for every set $S \subset \{0,1\}^{n+1}$ of size 2^n, the word $w_S \cdot w_{\bar{S},n}$ belongs to $\mathcal{L}(\nu(e_n))$, for every possible valuation $\nu : \Sigma \to \{x_1, \ldots x_{n+1}\}$. Let then S be an arbitrary subset of $\{0,1\}^{n+1}$ of size 2^n, and let ν be an arbitrary valuation from Σ to $\{x_1, \ldots, x_{n+1}\}$. Define $u = \nu(x_1) \cdots \nu(x_{n+1})$. Then u is a substring of either w_S or $w_{\bar{S},n}$. Assume the former is true (the other case is analogous). Then the word $w_S \cdot w_{\bar{S},n}$ can be written in the form $v \cdot u \cdot v' \cdot w_{\bar{S},n}$, with $v, v' \in \mathcal{L}((0 \mid 1)^{n+1})$. This shows that $w_S \cdot w_{\bar{S},n}$ belongs to $\mathcal{L}(\nu(e_n))$.
2. Assume for the sake of contradiction that there are distinct subsets S_1, S_2 of $\{0,1\}^{n+1}$ of size 2^n such that $w_{S_1} \cdot w_{\bar{S}_2,n}$ belongs to $\mathcal{L}_\Box(e_n)$. Since S_1 and S_2 are distinct, proper subsets of $\{0,1\}^{n+1}$ (they are of size 2^n), there must be a word in $\{0,1\}^{n+1}$ that belongs to S_2 but not to S_1. Let s be such word. Given that the word $w_{S_1} \cdot w_{\bar{S}_2,n}$ belongs to $\mathcal{L}_\Box(e_n)$, by Lemma 3 we have that s is a subword of $w_{S_1} \cdot w_{\bar{S}_2,n}$ that matches $w_{S_1} \cdot w_{\bar{S}_2,n}$ in a position j such that $j = 1 \mod n+1$. There are two possibilities. First, it could be that $j < |w_{S_1}|$. But since $j = 1 \mod n+1$, this means that s corresponds to one of the words in S_1, that gives form to w_{S_1}, which is a contradiction. On the other hand, if $j \geq |w_{S_1}|$, using essentially the same argument we conclude that s does not belong to S_2, which is also a contradiction.

We use essentially the same technique to address the $\Diamond$-semantics. 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) in Theorem 12.

1. From the fact that $\mathcal{L}_\Diamond(e_n) = \bigcup_{w \in \{0,1\}^n} w^*$, we have that for each $u \in \{0,1\}^n$ the word uu belongs to $\mathcal{L}_\Diamond(e_n)$.
2. The same fact shows that for every $u, v \in \{0,1\}^n$, if $u \neq v$, then $uv \notin \bigcup_{w \in \{0,1\}^n} w^*$, and thus $uv \notin \mathcal{L}_\Diamond(e_n)$.

This finishes the proof of the theorem. $\Box$

Note that the bounds of Theorem 11 apply to simple regular expressions.

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 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

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

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 of 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 13. *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 14. *For each parameterized regular expression e using variables $x_1, \ldots, x_n$ and for each 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). The automaton A'_e represents a finite set U of NFAs, obtained by applying valuations to each of the transitions of A'_e. Note that the set U is finite because the codomain of the valuations is now finite, as it is the union of all finite L_i's. It is now possible to show that these automata satisfy the required properties.

It will be more convenient for us to work with automata than with regular expressions. We deal with NFAs with extended transitions, which can be not just of the form (q, a, q'), where q and q' are states, and $a \in \Sigma$, but also (q, w, q'), where $w \in \Sigma^*$. Such an automaton accepts a word $s \in \Sigma^*$ in the standard way: in a run, in state q, if the subword starting in the current position is w, it can skip w and move to q' if there is a transition (q, w, q'). Note that such automata are a mere syntactic convenience (they will appear as the results of applying valuations), as any such automaton $\mathcal{A}$ can be transformed, in polynomial time, into a usual NFA $\mathcal{A}'$ so that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(A')$. Indeed, for each transition $t = (q, w, q')$ with $w = a_1 \ldots a_m$, introduce new states $q_t^1, \ldots, q_t^{m-1}$ and add transitions $(q, a_1, q_t^1), (q_t^1, a_2, q_t^2), \ldots, (q_t^{m-1}, a_n, q')$ to $\mathcal{A}'$. Thus, we shall work with automata with extended transitions.

Let e be a parameterized regular expression with variables $x_1, \ldots, x_n$, whose domains are regular languages $L_1, \ldots, L_n$. Let $\mathcal{A}_e$ be an NFA equivalent to e, over the alphabet $\Sigma \cup \{x_1, \ldots, x_n\}$. If we have a valuation ν so that $\nu(x_i) \in L_i$ for each $i \leq n$, then $\nu(A_e)$ is an automaton with extended transitions: in it, each transition (q, x_i, q') is replaced by $(q, \nu(x_i), q')$. It is then immediate from the construction that $\mathcal{L}(\nu(e)) = \mathcal{L}(\nu(\mathcal{A}_e))$ and thus $\mathcal{L}_\Box(e) = \bigcap_\nu \mathcal{L}(\nu(\mathcal{A}_e))$.

Next, consider *finitary* valuations ν, which are partial functions defined on variables x_i such that L_i is a finite language; of course $\nu(x_i) \in L_i$. On variables x_j with infinite L_j such valuations are undefined. By $\nu(\mathcal{A}_e)$ we mean the automaton (with extended transitions) resulting from $\mathcal{A}_e$ as follows. First, all transitions (q, a, q'), where a is a letter, are kept. Second, if (q, x_i, q') is a transition, then $\nu(\mathcal{A}_e)$ contains $(q, \nu(x_i), q')$ only if $\nu(x_i)$ is defined. In other words, transitions using variables whose domains are infinite, are dropped.

Let $\nu_1, \ldots, \nu_M$ enumerate all the finitary valuations (clearly there are finitely many of them). Let $\mathcal{A}_i = \nu_i(\mathcal{A}_e)$, for $i \leq M$. We now show that $\mathcal{L}_\Box(\mathcal{A}_e) = \bigcap_{i \leq M} \mathcal{L}(\mathcal{A}_i)$.

First, if ν_i is a finitary valuation and ν is any extension of ν_i to a valuation on all the variables $x_1, \ldots, x_n$, then clearly $\mathcal{L}(\nu_i(\mathcal{A}_e)) \subseteq \mathcal{L}(\nu(\mathcal{A}_e))$. Moreover, let V_i be the set of all valuations that are extensions of ν_i. Then we have that $\mathcal{L}(\nu_i(\mathcal{A}_e)) \subseteq \bigcap_{z \in V_i} \mathcal{L}(z(\mathcal{A}_e))$. But note that every valuation ν is an extension of some finitary valuation ν_i, and thus $\mathcal{L}_\Box(\mathcal{A}_e) = \bigcap_{\text{all valuations } \nu} \mathcal{L}(\nu(\mathcal{A}_e)) \supseteq \bigcap_{i \leq M} \mathcal{L}(\nu_i(\mathcal{A}_e))$. For the reverse inclusion, let $w \in \mathcal{L}_\Box(\mathcal{A}_e)$; in particular, $w \in \mathcal{L}(\nu(A_e))$ for every valuation ν. Take an arbitrary finitary valuation ν_i and let V_i be the (infinite) set of all the valuations ν that extend ν_i. Let $V_i(w)$ be the subset of V_i that contains valuations ν with the property that for each variable x_j with an infinite domain L_j, we have $|\nu(x_j)| > |w|$; clearly $V_i(w)$ is an infinite set as well. Take any $\nu \in V_i(w)$; we know from $w \in \mathcal{L}_\Box(\mathcal{A}_e)$ that $w \in \mathcal{L}(\nu(\mathcal{A}_e))$. In particular, there is an accepting run of $\nu(\mathcal{A}_e)$ that never uses any transition $(q, \nu(x_j), q')$ with L_j infinite, since $|\nu(x_j)| > |w|$. Thus, such an accepting run may only use transitions resulting from valuations of variables with finite domains, and hence it is also an accepting run of $\nu_i(\mathcal{A}_e)$. This shows that $w \in \mathcal{L}(\nu_i(\mathcal{A}_e))$; since ν_i was chosen arbitrarily, it means that $w \in \bigcap_{i \leq M} \mathcal{L}(\nu_i(\mathcal{A}_e))$, and thus proves $\mathcal{L}_\Box(\mathcal{A}_e) = \bigcap_{i \leq M} \mathcal{L}(\nu_i(\mathcal{A}_e)) = \bigcap_{i \leq M} \mathcal{L}(\mathcal{A}_i)$.

This immediately shows that $\mathcal{L}_\Box(e) = \mathcal{L}_\Box(\mathcal{A}_e)$ is regular, as a finite intersection of regular languages. Lower bounds on complexity apply immediately as they were all established for the case when each $L_i = \Sigma$. So we need to prove upper bounds. To do so, one can see, by analyzing the proofs for the case when all domains are Σ, that it suffices to establish the following facts on the set of automata $\mathcal{A}_i$, for $i < M$:

- M is at most exponential in the size of the input;
- checking whether a given automaton $\mathcal{A}$ is one of the $\mathcal{A}_i$'s can be done in time polynomial in the size of $\mathcal{A}$; and
- for each $\mathcal{A}_i$, for $i < M$, its size it at most polynomial in the size of the input.

(To give a couple of examples, to see the EXPSPACE-bound on NONEMPTINESS$_\Box$, we construct exponentially many automata of polynomial size and check nonemptiness of their intersection. To see the NP upper bound on MEMBERSHIP$_\Diamond$ for finite valuations, one guesses a polynomial-size $\mathcal{A}_i$, checks in polynomial time that it is indeed a correct automaton, and then checks again in polynomial time whether a given word is accepted by it.)

Recall that the input to the problem we are considering is $(e; \bar{L})$, or $(\mathcal{A}_e; \bar{L})$, and we can assume that each L_i is given by an NFA $\mathcal{B}_i$ (if part of the input is a regular expression, we can convert it into an NFA in $O(n \log^2 n)$ time [24]).

To show the bounds, assume without loss of generality that from each $\mathcal{B}_i$ all nonreachable states, and states from which final states cannot be reached, are removed (this can be done in polynomial time). Then $\mathcal{L}(\mathcal{B}_i)$ is finite iff $\mathcal{B}_i$ does not have cycles. Thus, if n_i is the number of states of $\mathcal{B}_i$, then the longest word accepted by $\mathcal{B}_i$ is of length n_i, and hence the size of each finite $L_i = \mathcal{L}(\mathcal{B}_i)$ is at most $|\Sigma|^{n_i+1}$. Hence, the total number of all the words in the finite languages L_i's is less than $|\Sigma|^N$, where $N = n + \sum n_i$, with the sum taken over indexes i such that L_i is finite. This means that in turn the number of finitary valuations M, i.e. mappings from some of the variables x_i's into words in these finite languages is at most $|\Sigma|^{Nn}$, which is thus exponential in the size of the input.

The remaining two properties are easy. Since the length of each word accepted by one of the $\mathcal{B}_i$'s is at most the number of states in $\mathcal{B}_i$, the size of all the automata $\nu_i(\mathcal{A}_e)$ is bounded by a polynomial in the size of the input; changing extended transitions in those to the usual NFA transitions involves only a linear increase of size. To check whether an automaton $\mathcal{A}$ is one of the $\mathcal{A}_i$'s, we check whether all its transitions involving both states from $\mathcal{A}_e$ come from $\mathcal{A}_e$ or from a single-letter valuation. Every other transition must be on a path between two states from $\mathcal{A}_e$. One reads words on these paths, and checks if they form a finitary valuation. This can easily be done in polynomial time. $\square$

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 [7]. 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] R. Angles, C. Gutierrez, Survey of graph database models, ACM Computing Surveys 40 (2008).

[2] P. Barceló, C. Hurtado, L. Libkin, P. Wood, Expressive languages for path queries over graph-structured data, in: 29th ACM Symposium on Principles of Database Systems (PODS), pp. 3–14.

[3] D. Calvanese, G. De Giacomo, M. Lenzerini, M. Vardi, Containment of conjunctive regular path queries with inverse, in: 7th International Conference on Principles of Knowledge Representation and Reasoning (KR), pp. 176–185.

[4] D. Calvanese, G. De Giacomo, M. Lenzerini, M. Vardi, Rewriting of regular expressions and regular path queries, Journal of Computer and System Sciences 64 (2002) 443–465.

[5] M. Consens, A. Mendelzon, Graphlog: A visual formalism for real life recursion, in: 9th ACM Symposium on Principles of Database Systems (PODS), pp. 404–416.

[6] S. Abiteboul, S. Cluet, T. Milo, Correspondence and translation for heterogeneous data, Theor. Comput. Sci. 275 (2002) 179–213.

[7] P. Barceló, L. Libkin, J. L. Reutter, Querying graph patterns, in: 30th ACM Symposium on Principles of Database Systems (PODS), pp. 199–210.

[8] D. Calvanese, G. De Giacomo, M. Lenzerini, M. Vardi, Simplifying schema mappings, in: 14th International Conference on Database Theory (ICDT), pp. 114–125.

[9] T. Imielinski, W. Lipski, Incomplete information in relational databases, Journal of the ACM 31 (1984) 761–791.

[10] Y. A. Liu, T. Rothamel, F. Yu, S. D. Stoller, N. Hu, Parametric regular path queries, in: Proceedings of the ACM Conference on Programming Language Design and Implementation (PLDI), pp. 219–230.

[11] Y. A. Liu, S. D. Stoller, Querying complex graphs, in: Proceedings of 8th International Symposium on the Practical Aspects of Declarative Languages (PADL), pp. 199–214.

[12] O. de Moor, D. Lacey, E. V. Wyk, Universal regular path queries, Higher-Order and Symbolic Computation 16 (2003) 15–35.

[13] X. Shen, Y. Zhong, C. Ding, Predicting locality phases for dynamic memory optimization, J. Parallel Distrib. Comput. 67 (2007) 783–796.

[14] J. Bhadra, A. K. Martin, J. A. Abraham, A formal framework for verification of embedded custom memories of the motorola mpc7450 microprocessor, Formal Methods in System Design 27 (2005) 67–112.

[15] G. Pesant, A regular language membership constraint for finite sequences of variables, in: Proceedings of the 10th International Conference on the Principles and Practice of Constraint Programming (CP), pp. 482–495.

[16] D. D. Freydenberger, Extended regular expressions: Succinctness and decidability, in: 28th International Symposium on Theoretical Aspects of Computer Science (STACS), pp. 507–518.

[17] O. Grumberg, O. Kupferman, S. Sheinvald, Variable automata over infinite alphabets, in: Proceedings of the 4th International Conference on Language and Automata Theory and Applications (LATA), pp. 561–572.

[18] M. Kaminski, D. Zeitlin, Finite-memory automata with non-deterministic reassignment, Int. J. Found. Comput. Sci. 21 (2010) 741–760.

[19] A. R. Meyer, L. J. Stockmeyer, The equivalence problem for regular expressions with squaring requires exponential space, in: 13th Annual Symposium on Switching and Automata Theory (SWAT/FOCS), pp. 125–129.

[20] W. Gelade, F. Neven, Succinctness of the complement and intersection of regular expressions, ACM Trans. Comput. Log. 13 (2012) 4.

[21] L. Kari, A. Mateescu, G. Paun, A. Salomaa, Multi-pattern languages, Theor. Comput. Sci. 141 (1995) 253–268.

[22] A. V. Aho, Algorithms for finding patterns in strings, in: Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity (A), 1990, pp. 255–300.

[23] D. Kozen, Lower bounds for natural proof systems, in: Proceedings of the 18th Annual Symposium on Foundations of Computer Science (FOCS), pp. 254–266.

[24] C. Hagenah, A. Muscholl, Computing epsilon-free nfa from regular expressions in o(n $\log^2$(n)) time, in: Proceedings of the 23rd International Symposium on the Mathematical Foundations of Computer Science (MFCS), pp. 277–285.

[25] I. Glaister, J. Shallit, A lower bound technique for the size of nondeterministic finite automata, Information Processing Letters 59 (1996) 75–77.