Closure Properties and Complexity of Rational Sets of Regular Languages

Andreas Holzer[a,1,*], Christian Schallhart[b,2], Michael Tautschnig[c], Helmut Veith[a]

[a] *Institut für Informationssysteme 184/4, Vienna University of Technology*
Favoritenstrasse 9-11, 1040 Wien, Austria
[b] *Department of Computer Science, Oxford University*
Wolfson Building, Oxford OX1 3QD, UK
[c] *School of Electronic Engineering and Computer Science*
Queen Mary University of London, Mile End Road, London E1 4NS, UK

Abstract

The test specification language FQL describes relevant test goals as regular expressions over program locations, such that each matching test case has an execution path matching this expression. To specify not only test goals but entire suites, FQL describes families of related test goals by regular expressions over extended alphabets: Herein, each symbol corresponds to a regular expression over program locations, and thus, a word in an FQL expression corresponds to a regular expression describing a single test goal. In this paper we provide a systematic foundation for FQL test specifications, which are in fact rational sets of regular languages (RSRLs). To address practically relevant problems like query optimization, we tackle open questions about RSRLs: We settle closure properties of general and finite RSRLs under common set theoretic operations. We also prove complexity results for checking equivalence and inclusion of star-free RSRLs, and for deciding whether a regular language is a member of a general or star-free RSRL.

*Corresponding author.
Email addresses: aholzer@cs.toronto.edu (Andreas Holzer), christian.schallhart@cs.ox.ac.uk (Christian Schallhart), michael.tautschnig@qmul.ac.uk (Michael Tautschnig), veith@forsyte.at (Helmut Veith)
[1] Now at University of Toronto.
[2] Now at Google London.

Keywords: Rational Sets, Regular Languages, Test Specification in FQL, Closure Properties, Decision Problems

1. Introduction

Despite the success of model checking and theorem proving, software testing retains a dominant role in industrial practice. In fact, state-of-the-art development guidelines such as the avionic standard DO-178B [1] are heavily dependent on test coverage criteria. It is therefore quite surprising that the formal specification of coverage criteria has been a blind spot in the formal methods and software engineering communities for a long time.

In a recent thread of papers [2, 3, 4, 5, 6, 7], we have addressed this situation and introduced the FSHELL Query Language (FQL). FQL allows to specify and tailor coverage criteria, and has been implemented in FSHELL [6] and CPA/TIGER [7], tools to generate matching test suites for ANSI C programs. At the semantic core of FQL, test goals are described as regular expressions the alphabet of which are the edges of the program's control-flow graph (CFG). For example, to cover a particular CFG edge c, one can use the regular expression $\Sigma^\star\ c\ \Sigma^\star$. Importantly, however, a coverage criterion usually induces not just a single test goal, but a (possibly large) number of test goals, e.g., *all* basic blocks of a program. FQL therefore employs regular languages which can express sets of regular expressions.

To this end, the alphabet contains not only the CFG edges but also *postponed regular expressions* over these edges, delimited by quotes. For example, "$\Sigma^\star$" $(a + b + c + d)$ "$\Sigma^\star$" describes the language $\{$"$\Sigma^\star$" a "$\Sigma^\star$", "$\Sigma^\star$" b "$\Sigma^\star$", "$\Sigma^\star$" c "$\Sigma^\star$", "$\Sigma^\star$" d "$\Sigma^\star$"$\}$. Each of these words yields a regular expression, e.g., $\Sigma^\star\ a\ \Sigma^\star$, that will in turn serve as test goal. Following [8], we call such languages *rational sets of regular languages (RSRLs).*

The goal of this paper is to initiate a systematic theoretical study of RSRLs, considering closure properties and complexity of common set-theoretic operations. Thus, this paper is a first step towards a systematic foundation of FQL. In particular, a good understanding of set-theoretic operations is necessary for systematic algorithmic optimization and manipulation of test specifications. First results on query optimization for FQL have been obtained in [7].

Contributions and Organization. Our results on RSRLs encompass closure properties for set theoretic operations and variants thereof as well as com-

plexity results on decision problems, justifying the design of FQL, as detailed in Section 3. Our paper is organized as follows:

- We formally introduce RSRLs in Section 2, then, sketch FQL and clarify the questions leading to the presented research in Section 3, and, then, survey related work in Section 4.

- *Closure properties (Section 5).* We consider general and finite RSRLs together with the operators Kleene star, product, complement, union, intersection, set difference, and symmetric difference. We also consider the case of finite RSRLs with a fixed language substitution φ, as this case is of particular interest for testing applications (cf. Section 3).

- *Complexity results (Section 6).* We discuss the complexity to decide equivalence, inclusion, and membership for Kleene-star free RSRLs. To prove an upper bound on the complexity of the membership problem for general RSRLs, we expand the decidability proof in [8] and give a first complete and explicit algorithm for the problem.

- *Justification of FQL Design (Section 7).* We conclude in discussing how our results reflect back on the design of FQL. In particular, our theoretical results confirm the decision to allow only Kleene-star free RSRLs in FQL.

A preliminary version of this work has been published in [9]; however, this older version was lacking almost all proofs. In contrast, the current paper contains full proof details on all results presented in Sections 5 and 6.

2. Rational Sets of Regular Languages

One of the most fundamental concepts in this paper are regular language substitutions, which map symbols in one alphabet to regular languages over another alphabet.

Definition 1 (Regular Language Substitution)**.** *Given a finite alphabet Σ, let* $\textsc{Reg}(\Sigma)$ *denote the set of regular languages over Σ. Then, given alphabets Δ and Σ, a* regular language substitution $\varphi : \Delta \to \textsc{Reg}(\Sigma)$ *maps each symbol $\delta \in \Delta$ to a regular language $\varphi(\delta) \in \textsc{Reg}(\Sigma)$. We extend φ to words $w \in \Delta^+$ with $\varphi(\delta \cdot w) = \varphi(\delta) \cdot \varphi(w)$, and set $\varphi(L) = \bigcup_{w \in L} \varphi(w)$ for $L \subseteq \Delta^+$.*

Please note that the extension of a regular language substitution to words yields regular languages again. Before we define rational sets of regular languages, we define rational sets of a monoid in general.

Definition 2 (Rational Sets of a Monoid)**.** *The class of rational sets of a monoid $(M, \cdot, e)$ is the smallest subclass of M such that (i) $\emptyset$ is a rational set, (ii) each singleton set $\{m\}$ for $m \in M$ is a rational set, and if N_1 and N_2 are rational sets (iii) then $N_1 \cdot N_2$ is a rational set where $\cdot$ on rational sets is defined by the point-wise application of the monoid's $\cdot$ operation, (iv) $N_1 \cup N_2$ is a rational set, and (v) $N_1^\star$ is a rational set [10, 11].*

Definition 3 (Rational Sets of Regular Languages, RSRLs [8])**.** *Given a finite alphabet Σ, the* rational sets of regular languages *are the rational sets over the monoid $(\textsc{Reg}(\Sigma), \cdot, \{\varepsilon\})$, where ε denotes the empty word.*

Definition 4 (Representation of RSRLs [8])**.** *We represent an RSRL $\mathcal{R}$ as a tuple (K, φ), where $K \subseteq \Delta^+$ is a regular language over a finite alphabet Δ, and φ is a regular language substitution $\varphi : \Delta \to \textsc{Reg}(\Sigma)$, such that $\mathcal{R} = \{\varphi(w) \mid w \in K\}$. We say that the RSRL $\mathcal{R}$ is* Kleene-star free, *if $\mathcal{R}$ is finite. That means that there exists a tuple (K, φ) such that $(K, \varphi) = \mathcal{R}$ where K is finite (and hence Kleene-star free).*

Note that we require a regular language $K \subseteq \Delta^+$ to exclude the empty word as the extension of regular language substitutions from symbols to words is not defined for the empty word. In the next two lemmas, we show that each RSRL can be represented by a tuple (K, φ) and that each tuple (K, φ) represents an RSRL.

Lemma 5 (Representation of RSRLs (part 1))**.** *Any RSRL $\mathcal{R}$ can be represented as a tuple (K, φ).*

Proof. Based on the structure of an RSRL $\mathcal{R}$, we define in Table 1 a representation of RSRLs as tuples of regular languages and regular language substitutions. In particular, given an alphabet Σ and an RSRL $\mathcal{R} \subseteq \textsc{Reg}(\Sigma)$, we define an alphabet $\Delta_\mathcal{R}$, a regular language $K_\mathcal{R} \subseteq \Delta_\mathcal{R}^+$, and a regular language substitution $\varphi_\mathcal{R} : \Delta_\mathcal{R} \to \textsc{Reg}(\Sigma)$ such that $\mathcal{R} = (K_\mathcal{R}, \varphi_\mathcal{R})$. We prove that $\mathcal{R} = (K_\mathcal{R}, \varphi_\mathcal{R})$ by induction over the structure of $\mathcal{R}$. The first two cases in Table 1 are immediate. In the third case, we have to deal with the potential overlap in the domains of $\varphi_{\mathcal{R}_1}$ and $\varphi_{\mathcal{R}_2}$. Since we cannot be sure that the overlapping symbols map to the same regular languages in both

RSRL $\mathcal{R} \subseteq \mathbf{Reg}(\Sigma)$	**Representation** $(K_\mathcal{R}, \varphi_\mathcal{R})$
$\emptyset$	$(\emptyset, [])$
$\{L\}$ where $L \in \textsc{Reg}(\Sigma)$	$(\{\delta\}, [\delta \mapsto L])$
$\mathcal{R}_1 \cdot \mathcal{R}_2$	(K', φ') where $K' = \{w_1 \cdot v_2 \mid w \in K_{\mathcal{R}_1}, v \in K_{\mathcal{R}_2}\}$, $\Delta' = \{\delta_1 \mid \delta \in \Delta_{\mathcal{R}_1}\} \cup \{\delta_2 \mid \delta \in \Delta_{\mathcal{R}_2}\}$, and $\varphi' = [\delta_1 \mapsto \varphi_{\mathcal{R}_1}(\delta)] \cup [\delta_2 \mapsto \varphi_{\mathcal{R}_2}(\delta)]$
$\mathcal{R}_1 \cup \mathcal{R}_2$	(K', φ') where $K' = \{w_1 \mid w \in K_{\mathcal{R}_1}\} \cup \{v_2 \mid v \in K_{\mathcal{R}_2}\}$, $\Delta' = \{\delta_1 \mid \delta \in \Delta_{\mathcal{R}_1}\} \cup \{\delta_2 \mid \delta \in \Delta_{\mathcal{R}_2}\}$, and $\varphi' = [\delta_1 \mapsto \varphi_{\mathcal{R}_1}(\delta)] \cup [\delta_2 \mapsto \varphi_{\mathcal{R}_2}(\delta)]$
$\mathcal{R}_1^*$	$(K', \varphi_{\mathcal{R}_1})$ where $K' = K^*_{\mathcal{R}_1}$

Table 1: Mapping of an RSRL $\mathcal{R}$ to a tuple $(K_\mathcal{R}, \varphi_\mathcal{R})$ (cf. Lemma 5)

language substitutions, we introduce a new alphabet Δ' and a new regular language substitution φ'. The alphabet Δ' contains for each symbol $\sigma \in \Delta_{\mathcal{R}_1}$ a new symbol σ_1 and for each symbol $\delta \in \Delta_{\mathcal{R}_2}$ a new symbol δ_2, i.e., a symbol $\sigma \in \Delta_{\mathcal{R}_1} \cap \Delta_{\mathcal{R}_2}$ results in distinct symbols σ_1 and σ_2. The language substitution φ' then maps a symbol σ_i to the regular language $\varphi_{\mathcal{R}_i}(\sigma)$, for $i \in \{1, 2\}$.

Let $L \in \varphi'(K')$. Then, by construction, there must be a word $w_1 v_2 \in K'$ such that $L = \varphi'(w_1 v_2) = \varphi_{\mathcal{R}_1}(w) \cdot \varphi_{\mathcal{R}_2}(v)$. From the induction hypothesis we have $\varphi_{\mathcal{R}_1}(w) \in \mathcal{R}_1$ and $\varphi_{\mathcal{R}_2}(v) \in \mathcal{R}_2$ and, hence, $\varphi'(w_1 v_2) \in \mathcal{R}_1 \cdot \mathcal{R}_2$. On the other hand, let $L \in \mathcal{R}_1 \cdot \mathcal{R}_2$. Then, there are languages $L_1 \in \mathcal{R}_1$ and $L_2 \in \mathcal{R}_2$ such that $L = L_1 \cdot L_2$. From the induction hypothesis we have that there are words w and v such that $\varphi_{\mathcal{R}_1}(w) = L_1$ and $\varphi_{\mathcal{R}_2}(v) = L_2$. Then, $w_1 v_2 \in K'$ and, furthermore, $\varphi'(w_1 v_2) = \varphi_{\mathcal{R}_1}(w) \cdot \varphi_{\mathcal{R}_2}(v) = L_1 \cdot L_2 = L$. The union case is analogous. Finally, let $L \in \mathcal{R}_1^*$, i.e., $L \in \bigcup_{i \geq 0} \mathcal{R}_1^i$. That means that there exists an $i \geq 0$ such that $L \in \mathcal{R}_1^i$. This, in turn, means that we can decompose L into a sequence of compositions: $L = L_0 \cdot L_1 \cdot \ldots L_{i-1}$ where $L_j \in \mathcal{R}_1$ for $0 \leq j < i$. By induction hypothesis, for each L_j there must be a word $w_j \in K_{\mathcal{R}_1}$ such that $L_j = \varphi_{\mathcal{R}_1}(w_j)$. Therefore, $w_0 w_1 \ldots w_{i-1} \in K^i_{\mathcal{R}_1}$. And since $L = L_0 \cdot L_1 \cdot \ldots \cdot L_{i-1} = \varphi_{\mathcal{R}_1}(w_0) \cdot \varphi_{\mathcal{R}_1}(w_1) \cdot \ldots \cdot \varphi_{\mathcal{R}_1}(w_{i-1})$ we have

Regular Language $K \subseteq \Delta^+$	**RSRL** $\mathcal{R}_{(K,\varphi)}$
$\emptyset$	$\emptyset$
$\{\delta\}$	$\{\varphi(\delta)\}$
$K_1 \cdot K_2$	$\mathcal{R}_{(K_1,\varphi)} \cdot \mathcal{R}_{(K_2,\varphi)}$
$K_1 \cup K_2$	$\mathcal{R}_{(K_1,\varphi)} \cup \mathcal{R}_{(K_2,\varphi)}$
K_1^*	$\mathcal{R}^*_{(K_1,\varphi)}$

Table 2: Mapping of a tuple (K, φ) to an RSRL $\mathcal{R}$ (cf. Lemma 6).

that $L = \varphi_{\mathcal{R}_1}(w_0 w_1 \dots w_{i-1})$. The other direction follows in an analogous way from the induction hypothesis. □

Lemma 6 (Representation of RSRLs (part 2))**.** *Any tuple (K, φ), where $K \subseteq \Delta^+$ is a regular language over Δ and $\varphi : \Delta \to \textsc{Reg}(\Sigma)$ is a regular language substitution, represents an RSRL.*

Proof. Let (K, φ) be a tuple of a regular language $K \subseteq \Delta^+$ and a regular language substitution $\varphi : \Delta \to \textsc{Reg}(\Sigma)$. In Table 2 we give the correspondence of (K, φ) to an RSRL $\mathcal{R}_{(K,\varphi)}$. The proof of this correspondence is by induction over the structure of L: The first two cases are immediate. The remaining three cases follow from the induction hypothesis and the definition of RSRLs. □

The following proposition summarizes the results of Lemma 5 and 6:

Proposition 7. *Any RSRL can be represented as a tuple (K, φ) and any tuple (K, φ) represents an RSRL.*

Depending on context, we refer to $\mathcal{R}$ as a set of languages or as a pair (K, φ), and we always write $L \in \mathcal{R}$ iff $\exists w \in K : L = \varphi(w)$. Reconsider the specification "$\Sigma^\star$" $(a + b + c + d)$ "$\Sigma^\star$" of Section 1 over base alphabet $\Sigma = \{a, b, c, d\}$. To represent this specification as RSRL $\mathcal{R} = (K, \varphi)$, we set $\Delta = \{\delta_{\Sigma^\star}\} \cup \Sigma$, containing a fresh symbol $\delta_{\Sigma^\star}$ for the quoted expression "$\Sigma^\star$". We set $K = L(\delta_{\Sigma^\star}\ (a + b + c + d)\ \delta_{\Sigma^\star})$ with $\varphi(\delta_{\Sigma^\star}) = \Sigma^\star$ and $\varphi(\sigma) = \{\sigma\}$ for $\sigma \in \Sigma$. Thus K contains the words $\delta_{\Sigma^\star}\ a\ \delta_{\Sigma^\star}, \dots$ with $\varphi(\delta_{\Sigma^\star}\ a\ \delta_{\Sigma^\star}) = L(\Sigma^\star\ a\ \Sigma^\star) \in \mathcal{R}$, as desired.

Note that the RSRL above is finite with exactly four elements. This is not atypical: in concrete testing applications, FQL generates finite sets of

test goals, since it relies on *Kleene-star free* RSRLs only. In this paper, we are therefore considering the general, finite, and Kleene-star free case. For future applications, however, it is well possible to consider infinite sets of test goals, e.g., for unbounded variables or path coverage criteria, which are either matched partially or by abstract executions.

Example 8. *Consider the alphabets* $\Delta = \{\delta_1, \delta_2\}$ *and* $\Sigma = \{a, b\}$*. Then,*

(1) *with* $\varphi(\delta_1) = L(a^\star)$*,* $\varphi(\delta_2) = \{ab\}$*, and* $K = L(\delta_1\delta_2^\star\delta_1)$*, we obtain the rational set of regular languages* $\{L(a^\star(ab)^i a^\star) \mid i \in \mathbb{N}\}$*;*

(2) *with* $\varphi(\delta_1) = L(a^\star)$*,* $\varphi(\delta_2) = \{a\}$*, and* $K = L(\delta_1\delta_2^\star)$*, we obtain* $\varphi(w_1) \supset \varphi(w_2)$ *for all* $w_1, w_2 \in K$ *with* $|w_1| < |w_2|$*;*

(3) *with* $\varphi(\delta_1) = \{\varepsilon, a\}$*,* $\varphi(\delta_2) = \{aa\}$*, and* $K = L(\delta_1\delta_2^\star)$*, we have* $|\varphi(w)| = 2$ *and* $\varphi(w) \cap \varphi(w') = \emptyset$ *for all* $w \neq w' \in K$*.*

In the finite case we make an additional distinction for the subcase where the regular expressions in Δ, i.e., the set of postponed regular expressions, are fixed. This has practical relevance, because in the context of FQL, the results of the operations on RSRL will be better readable by engineers if Δ is unchanged.

3. FQL from an RSRL Point of View

To motivate our research results of Sections 5 and 6 we first provide a short introduction to the coverage specification language FQL from an RSRL point of view. The design and study of FQL leads to a number of research questions on the properties of RSRLs, since RSRLs underly FQL as theoretical framework.

FQL is rooted in the *query-driven program testing* paradigm [5] that provides test cases that follow an engineer's specification instead of randomly walking the code. FQL queries are declarative specifications of test suites which are then interpreted by suitable automatic test case generation backends like FShell[3] [12, 6] or CPA/Tiger [7][4]. For ease of presentation, we will focus on the core concepts of FQL and omit aspects of FQL irrelevant to this paper. For a comprehensive discussion of please see [2, 4].

[3] `http://www.forsyte.at/software/fshell/`
[4] `http://www.forsyte.at/software/cpatiger/`

```
int cmp(int x, int y) {
  int value = 0;
  if (x > y)
    value = 1;
  else if (x < y)
    value = −1;
  return value;
}
```

Listing 1: C Source Code of Function `cmp`

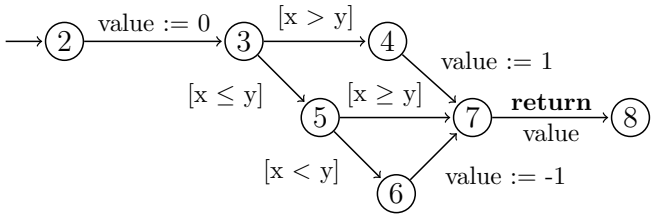

Figure 1: Control-Flow Automaton for Listing 1

A Short Introduction to FQL

In its simplest form, an FQL query can be directly mapped to an RSRL (K, φ) where $K \subseteq \Delta^+$ is a regular language over an alphabet Δ and $\varphi : \Delta \to \textsc{Reg}(\Sigma)$ is a regular language substitution that maps a symbol $\delta \in \Delta$ to a regular language $L_\delta \subseteq \Sigma^*$ over another alphabet Σ. We will start our discussion of FQL by introducing how FQL defines the alphabet Σ.

Programs as Control-Flow Automata. Syntactically, FQL uses control-flow automata (CFA) [13] to represent programs. For example, Figure 1 shows the CFA for the code in Listing 1. A node in the graph represents a program counter value and an edge is labeled with an operation which is either a skip statement, an assignment, an assumption (modeling conditional statements), a function call, or a function return. Furthermore, edges are also annotated with parsing information, for example, line numbers, file names, function names, code labels, etc. (for simplicity, we omit such parsing information in Figure 1). *The edges of a CFA form the alphabet Σ of an RSRL that results from an FQL query.* A regular language over Σ represents a set of program executions and FQL identifies *test goals* with such regular languages.

Regular Language Substitutions in FQL. Each symbol $\sigma \in \Sigma$ is also a symbol in the alphabet Δ and is mapped to the singleton set $\{\sigma\}$ via the language substitution φ, i.e., $\varphi(\sigma) = \{\sigma\}$. Furthermore, FQL features a *quotation* operator that turns a regular expression over Σ into a symbol in Δ. For example, FQL considers the quoted regular expression "$\sigma_1 + \sigma_2$" as a symbol in Δ and defines its language substitution as $\varphi(\text{“}\sigma_1 + \sigma_2\text{”}) = L(\sigma_1 + \sigma_2) = \{\sigma_1, \sigma_2\}$. In general, a *quoted regular expression* "e" is mapped to its regular language $L(e) \in \textsc{Reg}(\Sigma)$ via the language substitution φ, i.e., $\varphi(\text{“}e\text{”}) = L(e)$.

Filter Functions. CFAs and thus the alphabet Σ are an internal representation of the source code under investigation. FQL does not grant direct access to it but instead provides so called *filter functions* that identify sets of CFA edges. For example, the filter function `@BASICBLOCKENTRY` identifies the edges of a CFA that correspond to the respectively first statement in a basic block. As the CFA is finite these sets are also finite and therefore regular languages over Σ. Hence, filter functions serve as placeholders for regular expressions that are internally generated when source code is parsed. Filter functions provide the link to the individual programming language and the program under investigation. FQL also provides set operations like union, intersection, or complement, to further manipulate regular expressions that result from filter functions.

Question 9 (Fixed Language Substitution)**.** Language substitutions *directly map to evaluated filter functions. Any automatic/tool-internal manipulation of FQL queries whose result is presented to the user might* suffer from a lack of proper language substitutions, i.e., filter functions, to express the result. *We thus ask for operations on FQL query expressions that result in query expressions which are based on the same language substitution as the input.*

In order to express regular expressions over filter functions, FQL features the standard regular expression operators '`+`', '`*`', '`.`' for alternative, Kleene star and concatenation. However, in order to prevent an infinite number of test goals, no Kleene star is allowed outside of quotes.

Question 10 (Finite Case for Decision Problems and Closure Properties)**.** *FQL is* limited to the finite case *of RSRLs and we therefore ask which operations on FQL query expressions are closed under finite RSRLs.*

FQL Queries. FQL queries have the form `cover` (K, φ) `passing` L_P where (K, φ) is an RSRL constructed as described above and $L_P \in \textsc{Reg}(\Sigma)$. As mentioned earlier, each regular language $L \in \{\varphi(w) \mid w \in K\}$ constitutes a test goal and FQL requires a test-generation backend to provide for each such test goal L a program execution that traverses the CFA along edges that match a word in L. The *passing clause* L_P further restricts the test goals, i.e., for each test goal L, a program execution is required that matches a word in $L \cap L_P$. In the rest of this paper, we will refer to the intersection of all regular languages in an RSRL with another given regular language as *point-wise intersection*.

Question 11 (Closure Properties for Point-wise Set Operations)**.** *The passing clause helps writing concise queries, however, it remains to establish whether the passing clause is increasing the expressibility of FQL or whether it is merely syntactic sugar.*

Beyond closure properties, we would like to know the complexity of decision problems arising in the processing of FQL queries. For example, we would like to decide equivalence and inclusion over finite RSRLs to determine whether additional test cases need to be generated to cover the goals specified by a new FQL query (assuming we have test cases to cover the goals of an already given query). Inclusion and equivalence are also important to optimize FQL queries, e.g., to run on different test case generation backends. As another example, during the analysis of a program we might build a test suite by constructing a suitable FQL query iteratively and by generating a test suite from the query. If we want to include further test goals, we have to decide whether they are already covered by the query constructed so far or whether a query update is necessary, resulting in an additional run of the test case generator.

Question 12 (Complexity of Decision Problems)**.** *What is the complexity of equivalence, inclusion, and membership for FQL queries, i.e., on finite RSRLs.*

4. Related Work

Afonin et al. [8] introduced RSRLs and studied the decidability of whether a regular language is contained in an RSRL and the decidability of whether an RSRL is finite. Although Afonin et al. briefly discuss possible upper bounds for the membership decision problem, their analysis is incomplete due to gaps in their algorithmic presentation (see also a more detailed discussion in Section 6.5). Closely connected to the membership problem is the question, whether a regular language L is expressible via a combination of a given set of regular languages L_i. Motivated by query rewriting for graph databases, Calvanese et al. [14] show the complexity of determining the maximal rewriting of a regular language L with given regular languages L_i. In earlier work, Hashiguchi [15] shows that it is decidable whether a regular language L is expressible via a finite application of a subset of the regular operators concatenation, union, and star to regular languages L_i. Afonin et al. [8] realized that distance automata [16] enable a decision algorithm for the membership

problem for RSRL. Although their construction relies on distance automata, the properties analyzed by Krob [17] and Colcombet and Daviaud [18] are not applicable in our context. Kirsten [19, 20] generalizes distance automata to distance desert automata and uses these automata to show the first complexity result for determining whether a regular language is of a certain star height. Berstel [21] surveys closure properties of rational and recognizable subsets of monoids and thereby also the relationship between rational and recognizable subsets. Yet, most stated results do not apply to RSRLs, hence we investigate closure properties of RSRLs. Pin [22] introduced the term *extended automata* for RSRLs as an example of recognizable languages that can be characterized by constraint systems over symbols and substrings occurring in words of the language, but he did not further investigate any of their properties. In our own related work on FQL [2, 12, 3, 4, 5, 6, 7], we deal with practical issues arising in test case generation. Beyond RSRLs, FQL provides an additional language layer to extract suitable alphabets from the programs e.g., referring with a single symbol to all basic blocks of the program under scrutiny.

Let us finally discuss other work the terminology of which is similar to RSRLs without direct technical relation. Barceló et al. define *rational relations*, which are relations between words over a common alphabet, whereas we consider sets of regular languages [23]. Barceló et al. also investigate *parameterized regular languages* [24], where words are obtained by replacing variables in expressions with alphabet symbols. *Metaregular languages* deal with languages recognized by automata with a time-variant structure [25, 26]. Lattice Automata [27] only consider lattices that have a unique complement element, whereas RSRLs are not closed under complement (no RSRL has an RSRL as complement).

5. Closure Properties

We investigate the closure properties of RSRLs, considering standard set theoretic operators, such as union, intersection, and complement, and variants thereof, fitting RSRLs. In particular, we apply those operators also to pairs in the *Cartesian product* of RSRLs, and *point-wise* to each element in an RSRL and another given regular language.

Definition 13 (Operations on RSRL)**.** *Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be RSRLs and let R be a regular language. Then, we define the following operations on RSRLs:*

Operation	*Definition*		
Product	$\mathcal{R}_1 \cdot \mathcal{R}_2$	$=$	$\{L_1 \cdot L_2 \mid L_1 \in \mathcal{R}_1, L_2 \in \mathcal{R}_2\}$
Kleene Star	$\mathcal{R}_1^\star$	$=$	$\bigcup_{i \in \mathbb{N}} \mathcal{R}_1^i$
Point-wise	$\dot{\mathcal{R}}_1^\star$	$=$	$\{L^\star \mid L \in \mathcal{R}_1\}$
Complement	$\overline{\mathcal{R}_1}$	$=$	$\{L \subseteq 2^{\Sigma^\star} \mid L \notin \mathcal{R}_1\}$
Point-wise	$\dot{\overline{\mathcal{R}_1}}$	$=$	$\{\overline{L} \mid L \in \mathcal{R}_1\}$
Binary Operators	$\mathcal{R}_1 \cap \mathcal{R}_2,\ \mathcal{R}_1 \cup \mathcal{R}_2,\ \mathcal{R}_1 - \mathcal{R}_2$		*(standard def.)*
Point-wise	$\mathcal{R}_1 \cup / \cap / \dot{-}\, R$	$=$	$\{L \cup / \cap / - R \mid L \in \mathcal{R}_1\}$
Cartesian	$\mathcal{R}_1 \bowtie / \boxtimes / \stackrel{\times}{-} \mathcal{R}_2$	$=$	$\{L_1 \cup / \cap / - L_2 \mid L_1 \in \mathcal{R}_1, L_2 \in \mathcal{R}_2\}$
Symmetric Difference	$\mathcal{R}_1 \Delta \mathcal{R}_2$	$=$	$\{L \mid L \in ((\mathcal{R}_1 \cup \mathcal{R}_2) - (\mathcal{R}_1 \cap \mathcal{R}_2))\}$

We analyze *three different classes* of RSRLs for being closed under these operators:

(1) General RSRLs,

(2) finite RSRLs, and

(3) finite RSRLs with a fixed language substitution φ.

For closure properties, we do *not distinguish* between Kleene-star free and finite RSRLs, since every finite RSRL is expressible as Kleene-star free RSRL (however, given an RSRL with Kleene star, it is non-trivial to decide whether the given RSRL is finite or not [8]). Therefore, all closure properties for finite RSRLs apply to Kleene-star free RSRLs as well.

Observe that cases **(2-3)** correspond to FQL. As stated in Question 9, case **(3)** is relevant for usability in practice, allowing to apply the corresponding operators without constructing a new language substitution. This does not only significantly reduce the search space but also provides more intuitive results to users.

Theorem 14 (Closure Properties of RSRL)**.** *Table 3 summarizes the closure properties for RSRLs.*

We first state general observations on alphabet normalization and cardinality properties to subsequently exploit them in our proofs. Let $\mathcal{R}_1$ and

Operation		Closure Property		
		General	Finite RSRLs	
(+ closed - not closed ? unknown)			General	Fixed Subst.
Product	(Sec. 5.1)	+	+	+
Kleene Star	(Sec. 5.1)	+	-	-
Point-wise		-	+	-
Complement	(Sec. 5.2)	-	-	-
Point-wise		-	+	-
Union	(Sec. 5.3)	+	+	+
Point-wise		-	+	-
Cartesian		-	+	-
Intersection	(Sec. 5.4 and 5.6)	?	+	+
Point-wise		-	+	-
Cartesian		-	+	-
Difference	(Sec. 5.5 and 5.6)	?	+	+
Point-wise		-	+	-
Cartesian		-	+	-
Symmetric		?	+	+

Table 3: Closure Properties of RSRL

$\mathcal{R}_2$ be RSRLs over a common alphabet Σ with $\mathcal{R}_i = (K_i, \varphi_i)$, $K_i \subseteq \Delta_i^+$, and $\varphi_i : \Delta_i \to 2^{\Sigma^\star}$. Then we create a unified alphabet $\Delta = \{\langle i, \delta\rangle \mid \delta \in \Delta_i \text{ with } i = 1, 2\}$ and a unified language substitution $\varphi : \Delta \to 2^{\Sigma^\star}$ with $\varphi(\langle i, \delta\rangle) = \varphi_i(\delta)$. We obtain $\mathcal{R}_i = (K_i', \varphi)$ where K_i' is derived from K_i by substituting each symbol $\delta \in \Delta_i$ with $\langle i, \delta\rangle \in \Delta$. Hence without loss of generality, we **fix** the **alphabets** Δ and Σ with **language substitution** φ, allowing our RSRLs only to differ in the generating languages K_i. When we discuss binary operators, we freely refer to **RSRLs** $\mathcal{R}_i = (K_i, \varphi)$ for $i = 1, 2$, in case of unary operators to $\mathcal{R} = (K, \varphi)$, and in case of point-wise operators to the regular language $R \subseteq \Sigma^\star$.

Fact 15 (Finite Sets of Regular Languages are Rational)**.** *Every finite set of regular languages is rational.*

Proof. For a finite set of regular languages $\mathcal{R}$, we set $\varphi(\delta_L) = L$ for all $L \in \mathcal{R}$, taking fresh symbols δ_L. With $\Delta_\mathcal{R} = \{\delta_L \mid L \in \mathcal{R}\}$ we obtain

$\mathcal{R} = (\Delta_{\mathcal{R}}, \varphi)$. □

Fact 16 (Cardinality of RSRL)**.** *An RSRL contains at most countably many languages. In particular,* $2^{\Sigma^\star}$ *is not an RSRL.*

Proof. An RSRL $\mathcal{R} = (K, \varphi)$ is countable, as K contains countably many words, and $|K| \geq |\mathcal{R}|$ holds, because φ is a function. Since $2^{\Sigma^\star}$ is uncountable, it is not an RSRL. □

5.1. Product and Kleene Star

Proposition 17 (Closure of Product and Kleene Star)**. (1)** $\mathcal{R}_1 \cdot \mathcal{R}_2$ *is an RSRL, defined over the same language substitution* φ*. If* $\mathcal{R}_1$ *and* $\mathcal{R}_2$ *are finite, then* $\mathcal{R}_1 \cdot \mathcal{R}_2$ *is also finite.* **(2)** $\mathcal{R}^\star$ *is an RSRL. It is in general infinite even if* $\mathcal{R}$ *is finite.*

Proof. We prove each statement individually:

(1) We construct $\mathcal{R}' = (K', \varphi)$ with $K' = K_1 \cdot K_2$ and obtain $\mathcal{R}_1 \cdot \mathcal{R}_2 = \mathcal{R}'$.

(2) We construct $\mathcal{R}' = (K', \varphi')$ with $K' = K^\star - \{\varepsilon\} \cup \{\delta_\varepsilon\}$ setting $\varphi'(\delta_\varepsilon) = \{\varepsilon\}$ and $\varphi'(\delta) = \varphi(\delta)$ otherwise, and obtain $\mathcal{R}^\star = \mathcal{R}'$. Consider the finite RSRL $\mathcal{R} = \{\{a\}\}$, then, $\mathcal{R}^\star$ is the infinite RSRL $\{\{a^i\} \mid i \geq 0\}$. □

In the following we consider the set $S(L)$ of *shortest words* of a language L, disregarding ε, defined with $S(L) = \{w \in L \mid |w| = \mathsf{minlen}(L - \{\varepsilon\})\}$. We also refer to the shortest words $S(\mathcal{R})$ of an RSRL $\mathcal{R}$ with $S(\mathcal{R}) = \bigcup_{L \in \mathcal{R}} S(L)$.

Lemma 18. *Let* $\varepsilon \in \varphi(\delta)$ *hold for all* $\delta \in \Delta$*. Then, for each* $w \in \Delta^+$ *and shortest word* $v \in S(\varphi(w))$*, there exists a* $\delta \in \Delta$ *such that* $v \in S(\varphi(\delta))$*.*

Proof. We start with a corollary: Because of $\varepsilon \in \varphi(\delta)$ for all $\delta \in \Delta$, we have $\varphi(\delta_i) \subseteq \varphi(w)$ for $w = \delta_1 \dots \delta_k$ and all $1 \leq i \leq k$.

Assume $v \in S(\varphi(w))$ with $v \notin \varphi(\delta)$ for all $\delta \in \Delta$. Then $v = v_1 \dots v_k$ with $v_i \in \varphi(\delta_i)$, and since $v \neq \varepsilon$, $v_p \neq \varepsilon$ for some $1 \leq p \leq k$. We fix such a p. From the corollary above, we get $v_p \in \varphi(\delta_p) \subseteq \varphi(w)$, leading to a contradiction: If $v \neq v_p$, then v is not a shortest word in $\varphi(w) - \{\varepsilon\}$, as v_p is shorter. If $v = v_p$, we contradict our assumption with $v = v_p \in \varphi(\delta_p)$.

Thus, we have shown that there exists a δ with $v \in \varphi(\delta)$. It remains to show $v \in S(\varphi(\delta))$. Assuming that $v' \in \varphi(\delta) - \{\varepsilon\}$ is shorter than v, we quickly arrive at a contradiction: $v' \in \varphi(\delta) \subseteq \varphi(w)$ from the corollary above, implies that v would not be a shortest word in $\varphi(w) - \{\varepsilon\}$ in the first place, i.e., $v \notin S(\varphi(w))$. □

Corollary 19. *Let $\varepsilon \in \varphi(\delta)$ hold for all $\delta \in \Delta$. Then the set of shortest words $S(\mathcal{R})$ is finite.*

Proof. Lemma 18 states for each word $v \in S(\mathcal{R})$, we have $v \in S(\varphi(\delta))$ for some $\delta \in \Delta$. But there are only finitely many symbols $\delta \in \Delta$, each generating only finitely many shortest words in $\varphi(\delta) - \{\varepsilon\}$. Hence $S(\mathcal{R})$ must be finite. □

Proposition 20 (Closure of Point-wise Kleene Star)**. (1)** *In general, $\dot{\mathcal{R}}^\star$ is not an RSRL.* **(2)** *If $\mathcal{R}$ is finite, $\dot{\mathcal{R}}^\star$ is a finite RSRL.* **(3)** *In the latter case, expressing $\dot{\mathcal{R}}^\star$ requires a new language substitution φ.*

Proof. We prove each statement individually:

(1) Consider the RSRL $\mathcal{R} = \{\{a^i\} \mid i \geq 1\}$ with $\dot{\mathcal{R}}^\star = \{L_i \mid i \geq 1\}$ with $L_i = \{a^{j \cdot i} \mid j \geq 0\}$. Every language $L_i \in \dot{\mathcal{R}}^\star$ contains the empty word $\varepsilon = a^{0 \cdot i}$, and hence, $\varepsilon \in \varphi(\delta)$ for all $\delta \in \Delta$ (disregarding symbols δ not occurring in K). Thus, Corollary 19 applies, requiring that the set of shortest words $S(\dot{\mathcal{R}}^\star)$ is finite. This leads to a contradiction, since $S(\dot{\mathcal{R}}^\star) = \{a^i \mid i \geq 1\}$ is infinite.

(2) Since $\mathcal{R}$ is finite, also $\dot{\mathcal{R}}^\star$ has to be finite and the statement follows from Fact 15.

(3) Consider the RSRL $\mathcal{R} = \{\{a\}\}$, produced from (K, φ) with $K = \{\delta_a\}$ and $\varphi(\delta_a) = a$. Then, $\dot{\mathcal{R}}^\star = \{\{a^i \mid i \geq 0\}\}$, and since $\{a\} \neq \{a^i \mid i \geq 0\}$ we have to introduce a new symbol. □

5.2. Complement

Proposition 21 (Non-closure under Complement)**.** *Let $\mathcal{R}$ be a rational set of regular languages. Then $\overline{\mathcal{R}}$ is not a rational set of regular languages.*

Proof. Fact 16 states that $\mathcal{R}$ is countable while $2^{\Sigma^\star}$ is uncountable. Hence, $2^{\Sigma^\star} - \mathcal{R}$ is uncountable and is therefore inexpressible as RSRL. □

Proposition 22 (Closure of Point-wise Complement)**. (1)** $\dot{\overline{\mathcal{R}}}$ *is in general not an RSRL.* **(2)** *If $\mathcal{R}$ is finite, $\dot{\overline{\mathcal{R}}}$ is a finite RSRL as well,* **(3)** *requiring, in general, a modified language substitution.*

Proof. We prove each statement individually:

(1) Consider the RSRL $\mathcal{R} = (K, \varphi)$ with $K = L(\delta\delta^\star)$ and $\varphi(\delta) = \{a, b\} = \Sigma$. Then we have $\mathcal{R} = \{\Sigma^i \mid i \geq 1\}$. For $i \neq j$, we have $\overline{\Sigma^i} \not\subseteq \overline{\Sigma^j}$ and $\overline{\Sigma^i} \not\supseteq \overline{\Sigma^j}$ since $\Sigma^j \subseteq \overline{\Sigma^i}$ and $\Sigma^i \subseteq \overline{\Sigma^j}$. Furthermore, observe $\varepsilon \in \overline{\Sigma^i}$ for each $i \geq 1$. Assume $\dot{\overline{\mathcal{R}}}$ is an RSRL. Then, there are K' and φ' such that $\dot{\overline{\mathcal{R}}} = (K', \varphi')$. Since $\dot{\overline{\mathcal{R}}}$ is infinite and K' is regular, there exists a word $w \in K'$ with $w = uvz$ and $\varphi(v) \neq \{\varepsilon\}$ and $uv^iz \in K'$ for all $i \geq 1$. Because of $\varepsilon \in \overline{\Sigma^p} = \varphi(uvz)$ for some p, we obtain $\varepsilon \in \varphi(v)$ as well. But then, for all $i \geq 1$, $\varphi(uvz) \subseteq \varphi(uv^iz)$, i.e., $\varphi(uvz) = \overline{\Sigma^p} \subseteq \overline{\Sigma^q} = \varphi(uv^iz)$. This contradicts the observation that $\overline{\Sigma^p} \not\subseteq \overline{\Sigma^q}$.

(2) By Fact 15.

(3) Let $\mathcal{R} = (\{\delta_a\}, \varphi)$ with $\varphi(\delta_a) = \{a\}$. Then, $\dot{\overline{\mathcal{R}}} = \{\Sigma^\star - \{a\}\}$. But, $\{a\} \neq \Sigma^\star - \{a\}$. Therefore, we need a new symbol to represent $\Sigma^\star - \{a\}$.

□

In contrast to complementation, some RSRLs have a point-wise complement that is an RSRL as well; first, this is true for all finite RSRLs, as shown above, but there are also some infinite RSRLs which have point-wise complement.

Example 23. *The RSRL $\mathcal{R} = (L(\delta\delta^\star), \varphi)$ with $\varphi(\delta) = \{a, b, \varepsilon\}$ has the point-wise complement $\dot{\overline{\mathcal{R}}} = (L(\delta_1\delta_1\delta_1^\star\delta_2), \varphi')$ with $\varphi'(\delta_1) = \{a, b\}$ and $\varphi'(\delta_2) = L((a + b)^\star)$.*

5.3. Union

Proposition 24 (Closure of Union)**.** *The set $\mathcal{R}_1 \cup \mathcal{R}_2$ is a rational set of regular languages, expressible as $(K_1 \cup K_2, \varphi)$ without changing the substitution φ.*

Proof. Regular languages are closed under union, hence the claim follows. □

The following set of regular languages is not an RSRL and we use it to prove the non-closure of RSRLs under the point-wise union operator.

Example 25. *Consider the set $\mathcal{M} = \{\{b\} \cup \{a^i \mid 1 \leq i \leq n + 1\} \mid n \in \mathbb{N}\} \subseteq 2^{\{a,b\}^\star}$. $\mathcal{M}$ contains infinitely many languages, therefore, any RSRL $\mathcal{R} = (K, \varphi)$, with $\mathcal{M} = \mathcal{R}$, requires a regular language K containing infinitely many words. By L_n we denote the set $\{b\} \cup \{a^i \mid 1 \leq i \leq n+1\}$. Then, $L_0 \subsetneq$*

$L_1 \subsetneq \ldots L_{i-1} \subsetneq L_i \subsetneq L_{i+1} \subsetneq \ldots$. *There must be a word* $w = uvz \in K$ *such that* $uv^iz \in K$, *for all* $i \geq 1$ *(cf. pumping lemma for regular languages [28]). Furthermore, there must be such a word* $w = uvz$ *such that* $\varphi(u) \neq \emptyset$, $\varphi(v) \neq \emptyset$, $\varphi(v) \neq \{\varepsilon\}$, *and* $\varphi(z) \neq \emptyset$. *This is due to the fact that we have to generate arbitrary long words* a^i. *We can assume that* $b \notin \varphi(v)$ *because otherwise* $b^i \in \varphi(v^i)$, *for all* $i \geq 1$. *Therefore,* $a^k \in \varphi(v)$ *for some* $k \geq 1$. *Since* $b \in \varphi(uvz)$ *has to be true, we can assume w.l.o.g. that* $b \in \varphi(u)$. *But, then* $ba^k \ldots \in \varphi(uvz)$. *This is a contradiction to the fact that, for all* $n \geq 1$, $ba^k \ldots \notin L_n$.

Proposition 26 (Closure of Point-wise Union)**. (1)** *The set* $\mathcal{R}_1 \cup R$ *is, in general, not an RSRL.* **(2)** *The set* $\mathcal{R} \cup R$ *is an RSRL for finite* $\mathcal{R}$. **(3)** *In the latter case, the resulting RSRL requires in general a different language substitution.*

Proof. We prove each statement individually:

(1) Let $\mathcal{R} = (L(\delta_1\delta_2^\star), \varphi)$ with $\varphi(\delta_1) = \{a\}$ and $\varphi(\delta_2) = L(a + \varepsilon)$ and let $R = \{b\}$. Then, $\mathcal{R} \cup R = \{\{b\} \cup \{a^i \mid 1 \leq i \leq n+1\} \mid n \in \mathbb{N}\}$ which is not an RSRL, as shown in Example 25.

(2) By Fact 15.

(3) Let $\mathcal{R} = (\{\delta\}, \varphi)$ with $\Delta = \{\delta\}$, $\Sigma = \{a, b\}$, $\varphi(\delta) = \{a\}$ and let $R = \{b\}$. Then, $\mathcal{R} \cup R = \{\{a, b\}\}$, which is inexpressible with φ. □

5.4. Intersection

Proposition 27 (Closure of Intersection)**.** *Let* $\mathcal{R}_1$ *and* $\mathcal{R}_2$ *be two finite RSRLs using the same language substitution* φ. *Then,* $\mathcal{R}_1 \cap \mathcal{R}_2$ *is a finite RSRL which can be expressed using the language substitution* φ.

Proof. We can enumerate each word $w_1 \in K_1$ and check whether there is a word $w_2 \in K_2$ such that $\varphi(w_1) = \varphi(w_2)$. If so, we keep w_1 in a new set $K_3 = \{w_1 \in K_1 \mid \exists w_2 \in K_2.\varphi(w_1) = \varphi(w_2)\}$ and $(K_3, \varphi) = \mathcal{R}_1 \cap \mathcal{R}_2$. □

In general, RSRLs are not closed under point-wise intersection but they are closed under point-wise intersection when restricting to finite RSRLs.

Proposition 28 (Closure of Point-wise Intersection)**. (1)** *RSRL are not closed under point-wise intersection.* **(2)** *For finite* $\mathcal{R}$ $\mathcal{R} \cap R$ *is a finite RSRL,* **(3)** *in general requiring a different language substitution.*

Proof. We prove each statement individually:

(1) Let $\mathcal{R} = (K, \varphi)$ with $K = L(\delta\delta^\star)$ and $\varphi(\delta) = L(a + b^\star)$, and set $R = L(a^\star + b)$. Then $\mathcal{R} \curlywedge R = \{\{b\} \cup \{a^i \mid 1 \leq i \leq n+1\} \mid n \in \mathbb{N}\}$. In Example 25, we showed that $\mathcal{R} \curlywedge R$ is not an RSRL.

(2) By Fact 15.

(3) Let $\mathcal{R} = (K, \varphi)$ with $K = \{\delta\}$ and $\varphi(\delta) = L(a + b^\star)$, and set $R = L(a^\star + b)$. Then, $\mathcal{R} \curlywedge R = \{L(a + b)\}$ which in inexpressible via φ. □

5.5. Set Difference

Proposition 29 (Closure of Difference)**.** *For finite $\mathcal{R}_1$ and $\mathcal{R}_2$, $\mathcal{R}_1 - \mathcal{R}_2$ is a finite RSRL, expressible as (K_3, φ), for some $K_3 \subseteq K_1$.*

Proof. Set $K_3 = \{w \in K_1 \mid \varphi(w) \in \mathcal{R}_2\}$ and the claim follows. □

Proposition 30 (Closure of Point-wise Difference)**. (1)** *In general, $\mathcal{R} \dot{-} R$ is not an RSRL.* **(2)** *$\mathcal{R} \dot{-} R$ is a finite RSRL for finite $\mathcal{R}$,* **(3)** *requiring in general a different language substitution.*

Proof. We prove each statement individually:

(1) Let $\mathcal{R} = (L(\delta_1\delta_2^\star), \varphi)$ with $\varphi(\delta_1) = L(a + b)$ and $\varphi(\delta_2) = L(a + b + \varepsilon)$. Let $R = L(bbb^\star + (a + b)^\star ab(a + b)^\star + (a + b)^\star ba(a + b)^\star)$. Then, $\mathcal{R} \dot{-} R = \{\{b\} \cup \{a^i \mid 1 \leq i \leq n+1\} \mid n \in \mathbb{N}\}$, which is not an RSRL (see Example 25).

(2) By Fact 15.

(3) Let $\mathcal{R} = (\{\delta_a\}, \varphi)$ with $\varphi(\delta_a) = \{a\}$ and let $R = \{a\}$. Then, $\mathcal{R} \dot{-} R = \{\emptyset\}$, requiring a new symbol. □

Proposition 31 (Closure of Symmetric Difference)**.** *Let $\mathcal{R}_1$ and $\mathcal{R}_2$ be finite RSRLs using the same language substitution φ. Then, $\mathcal{R}_1 \Delta \mathcal{R}_2$ is a finite RSRL and can be expressed using the language substitution φ.*

Proof. The proof follows immediately from the closure properties of union, intersection, and difference. □

5.6. Cartesian Binary Operators

We deal with Cartesian binary operators generically, by reducing the point-wise operators to the Cartesian one.

Lemma 32 (Reducing Point-Wise to Cartesian Operators)**.** *Let* $\circ$ *be an arbitrary binary operator over sets, let* $\odot \in \{\cup, \cap, \dot{-}\}$*, and let* $\otimes \in \{\bowtie, \between, \overset{\times}{-}\}$*.* **(1)** *If* $\mathcal{R}_1 \odot R$ *is not closed under rational sets of regular languages, then the corresponding* $\mathcal{R}_1 \otimes \mathcal{R}_2$ *is not closed.* **(2)** *If* $\mathcal{R}_1 \odot R$ *is not closed under finite rational sets of regular languages with constant language substitution, even in presence of a symbol* δ_R *with* $\varphi(\delta_R) = R$*, then the corresponding* $\mathcal{R}_1 \otimes \mathcal{R}_2$ *is also not closed.*

Proof. We prove each statement individually:

(1) If $\mathcal{R}_1 \odot R$ is not closed, we fix a violating pair $\mathcal{R}_1$ and R. Then we obtain $\mathcal{R}_1 \otimes \mathcal{R}_2 = \mathcal{R}_1 \odot R$ for $\mathcal{R}_2 = (\{\delta_R\}, \varphi)$ and $\varphi(\delta_R) = R$. Since $\mathcal{R}_1 \odot R$ is not an RSRL, neither is $\mathcal{R}_1 \otimes \mathcal{R}_2$, and the claim follows.

(2) If $\mathcal{R}_1 \odot R$ is inexpressible as an RSRL without introducing new symbols in φ, even in presence of δ_R, then $\mathcal{R}_1 \otimes \mathcal{R}_2$ is also inexpressible without changing φ. □

Given Lemma 32, it is not surprising that point-wise and Cartesian operators behave identically for all discussed underlying binary operators, as shown in Theorem 14.

Corollary 33 (Closure of Cartesian Binary Operators)**.** *Let* $\otimes \in \{\bowtie, \between, \overset{\times}{-}\}$*.* **(1)** *The set* $\mathcal{R}_1 \otimes \mathcal{R}_2$ *is, in general, not a rational set of regular languages.* **(2)** *The set* $\mathcal{R}_1 \otimes \mathcal{R}_2$ *is a rational set of regular languages if* $\mathcal{R}_1$ *and* $\mathcal{R}_2$ *are finite,* **(3)** *requiring in general a new language substitution.*

Proof. We prove each statement individually:

(1) By Lemma 32 we reduce the point-wise case to the Cartesian case, covered by Propositions 26, 28, and 30 for union, intersection, and set difference, respectively. The claim follows.

(2) Since all considered operators are closed for regular languages, the claim follows from Fact 15.

(3) Again, with Lemma 32 we reduce the point-wise case to the Cartesian case. The lemma is applicable, as the examples in the proofs of Propositions 26, 28, and 30 are not jeopardized by a symbol δ_R with $\varphi(\delta_R) = R$. Hence the claim follows. □

5.7. Discussion of Open Closure Properties

As Table 3 shows, our investigation of closure properties of RSRLs does not answer closure under the operations *intersection*, *difference*, and *symmetric difference*. Would RSRLs be closed under complement, then, these questions could be answered immediately since $\mathcal{R}_1 \cap \mathcal{R}_2 = \overline{\overline{\mathcal{R}_1} \cup \overline{\mathcal{R}_2}}$, $\mathcal{R}_1 - \mathcal{R}_2 = \mathcal{R}_1 \cap \overline{\mathcal{R}_2}$, and $\mathcal{R}_1 \Delta \mathcal{R}_2 = (\mathcal{R}_1 \cup \mathcal{R}_2) - (\mathcal{R}_1 \cap \mathcal{R}_2)$. However, the fact that RSRLs are not closed under complement does not necessarily imply that RSRLs are also not closed under any of the above mentioned operations. In particular, $2^{\Sigma^\star}$ is not an RSRL, which would otherwise imply that RSRLs cannot be closed under difference as $\overline{\mathcal{R}} = 2^{\Sigma^\star} - \mathcal{R}$. Closure under difference seems to be central, as closure under difference also implies closure under intersection ($\mathcal{R}_1 \cap \mathcal{R}_2 = \mathcal{R}_1 - (\mathcal{R}_1 - \mathcal{R}_2)$) and closure under symmetric difference (which is defined via union, intersection, and difference). On the other hand, if RSRLs are not closed under difference, then, we know that RSRLs are also either not closed under intersection or under symmetric difference (or both) as $\mathcal{R}_1 - \mathcal{R}_2 = (\mathcal{R}_1 \Delta \mathcal{R}_2) \cap \mathcal{R}_1$.

To answer the above questions, we do not have to consider arbitrary RSRLs but one can limit the investigation to special cases based on the fact that RSRLs are closed under union. For the case of intersection, it is sufficient to consider RSRLs $\mathcal{R}_1$ and $\mathcal{R}_2$ such that both sets are infinite and such that there are tuples (K_1, φ_1) and (K_2, φ_2) where K_1 and K_2 are both union-free. The general case follows immediately from closure under union. In case that $\mathcal{R}_1 \cap \mathcal{R}_2$ is finite (as it is the case if $\mathcal{R}_1$ or $\mathcal{R}_2$ are finite), then, $\mathcal{R}_1 \cap \mathcal{R}_2$ is an RSRL. Similarly, if $\mathcal{R}_1 - \mathcal{R}_2$ is finite (as it is the case if $\mathcal{R}_1$ is finite), then, $\mathcal{R}_1 - \mathcal{R}_2$ is an RSRL. Similarly, if $\mathcal{R}_1 \Delta \mathcal{R}_2$ is finite, then, it is also an RSRL.

6. Decision Problems

Given a regular language $R \subseteq \Sigma^\star$ and an RSRL $\mathcal{R} = (K, \varphi)$ over the alphabets Δ and Σ, the *membership problem* is to decide whether $R \in \mathcal{R}$ holds. Given another $\mathcal{R}' = (K', \varphi')$ over the alphabets Δ' and Σ, the *inclusion problem* asks whether $\mathcal{R} \subseteq \mathcal{R}'$ holds, and the *equivalence problem,* whether $\mathcal{R} = \mathcal{R}'$ holds.

Theorem 34 (Equivalence, Inclusion, and Membership for Kleene-star free RSRLs)**.** *Membership, inclusion, and equivalence are* PSPACE*-complete for Kleene-star free represented RSRLs.*

This holds true, since in case of Kleene-star free represented RSRLs (given explicitly as (K, φ) with K finite), we can enumerate the regular expressions defining all member languages in PSPACE.

Proof of Theorem 34. PSPACE-*Membership.* We exploit for the PSPACE-membership of all three considered problems the same observations: **(1)** Given Kleene-star free languages K, we can enumerate in PSPACE all words $w \in K$, and **(2)** we can check whether $L(R) = L(\varphi(w))$ holds, in PSPACE [29].

Thus, to check *membership* of R in (K, φ), we enumerate all $w \in K$ and check whether $L(R) = L(\varphi(w))$ holds for some w – if so, $R \in \mathcal{R}$ is true. For checking the *inclusion* $\mathcal{R}' \subseteq \mathcal{R}$, we enumerate all $w' \in K'$ and search in a nested loop for a $w \in K$ with $L(\varphi(w)) = L(\varphi(w'))$. If such a w exists for all w', we have established $(K', \varphi') \subseteq (K, \varphi)$. We obtain PSPACE-membership for *equivalence* $(K', \varphi') = (K, \varphi)$ by checking both, $(K', \varphi') \subseteq (K, \varphi)$ and $(K, \varphi) \subseteq (K', \varphi')$.

Hardness. For hardness we reduce the PSPACE-complete problem whether a given regular expression $X \subseteq \Sigma^\star$ is equivalent to $\Sigma^\star$ [29] to all three considered problems: Given an arbitrary regular expressions X, we set $K = \{a\}$, $\varphi(a) = X$, $K' = \{b\}$, $\varphi'(b) = \Sigma^\star$, and $R = \Sigma^\star$. This gives us $X = \Sigma^\star$ iff $(K, \varphi) = (K', \varphi')$ (equivalence) iff $(K, \varphi) \subseteq (K', \varphi')$ (inclusion) iff $R \in (K, \varphi)$ (membership). □

This approach does *not* immediately generalize to finite RSRLs, since finite RSRLs $\mathcal{R} = \{\varphi(w) \mid w \in K\}$ may be generated from an infinite K with Kleene stars. And in the general case, the situation is quite different: Previous work shows that the membership problem is decidable [8], but without providing a concrete algorithm or determining an upper complexity bound. Taking this work as starting point, in the remainder of this section, we give an 2EXPSPACE upper bound on the complexity of the problem and discuss the relationship with [8] at the end of the section. The decidability of inclusion and equivalence remains open.

6.1. Membership for General RSRLs

By definition, the membership problem is equivalent to asking whether there exists a $w \in K$ with $\varphi(w) = R$. For checking the existence of such a w, we have to check possibly infinitely many words in K efficiently. To render this search feasible, we **(A)** rule out irrelevant parts of K, and **(B)** treat subsets of K at once. This leads to the procedure $\mathsf{membership}(K, R, \varphi)$ shown in

Algorithm 1: membership(R, K, φ)

input : regular languages $R \subseteq \Sigma^\star$, $K \subseteq \Delta^\star$,
regular language substitution φ with $\varphi(\delta) \subseteq \Sigma^\star$ for all $\delta \in \Delta$

returns : **true** iff $\exists w \in K : \varphi(w) = R$ (i.e., iff $R \in (K, \varphi)$)

1 **foreach** $M' \in$ enumerate(R, K, φ) **do**

2 **if** basiccheck(R, M', φ) **then return true** ;

3 **return false**;

Algorithm 1, which first enumerates with $M' \in \mathsf{enumerate}(K, R, \varphi)$ a sufficient set of sublanguages (Line 1), and then checks each of those sublanguages individually (Line 2). More specifically, we employ the following optimizations: We rule out **(A.1)** all words w with $\varphi(w) \not\subseteq R$, and **(A.2)** all words w the language $\varphi(w)$ of which differs from R in the *length of its shortest word.* We subdivide the remaining search space **(B)** into finitely many suitable languages M' and check the existence of a $w \in M'$ with $\varphi(w) = R$ in a single step.

Below, we discuss a mutually fitting design of these steps and consider the resulting complexity.

(A.1) *Maximal Rewriting.* To rule out all w with $\varphi(w) \not\subseteq R$, we rely on the notion of a *maximal φ-rewriting $M_\varphi(R)$ of R*, taken from [14]. $M_\varphi(R)$ consists of the words w with $\varphi(w) \subseteq R$, i.e., we set $M_\varphi(R) = \{w \in \Delta^+ \mid \varphi(w) \subseteq R\}$. Furthermore, all subsets $M \subseteq M_\varphi(R)$ are called *rewritings* of R, and if $\varphi(M) = R$ holds, M is called *exact* rewriting.

Proposition 35 (Regularity of maximal rewritings [14][5])**.** *Let $\varphi : \Delta \to 2^{\Sigma^\star}$ be a regular language substitution. Then the* maximal φ-rewriting $M_\varphi(R)$ *of a regular language $R \subseteq \Sigma^\star$ is a regular language over Δ.*

As all words w with $\varphi(w) = R$ must be element of $M_\varphi(R)$, we restrict our search to $M = M_\varphi(R) \cap K$.

[5]This proposition is not trivial, as φ is not a homomorphism mapping each word to a single word, but a substitution mapping each word w to a language $\varphi(w)$; if $\varphi(w)$ would yield only words, we would immediately obtain $M_\varphi(R) = \overline{\varphi^{-1}(\overline{R})}$ for $\varphi^{-1}(L) = \{w \mid \varphi(w) \cap L \neq \emptyset\}$.

(A.2) *Minimal Word Length.* We restrict the search space further by checking the *minimal word length,* i.e., we compare the length of the respectively shortest word in R and $\varphi(w)$. If R and $\varphi(w)$ have different minimal word lengths, $R \neq \varphi(w)$ holds, and hence, we rule out w. We define the minimal word length $\mathsf{minlen}(L)$ of a language L with $\mathsf{minlen}(L) = \min\{|w| \mid w \in L\}$, leading to the definition of language strata.

Definition 36 (Language Stratum)**.** *Let L be a language over Δ, and $\varphi : \Delta \to 2^{\Sigma^\star}$ be a regular language substitution, then the* B-stratum of L via φ, *denoted as $L[B, \varphi]$, is the set of words in L which generate via φ languages of minimal word length B, i.e., $L[B, \varphi] = \{w \in L \mid \mathsf{minlen}(\varphi(w)) = B\}$.*

Starting with $M = M_\varphi(R) \cap K$, we rule out words w with $\mathsf{minlen}(\varphi(w)) \neq B$ and thus restrict our search further to $M[\mathsf{minlen}(R), \varphi]$.

(B) *1-Word Summaries.* It remains to subdivide $M[\mathsf{minlen}(R), \varphi]$ into finitely many subsets M', which are then checked efficiently without enumerating their words $w \in M'$. Here, we only discuss the property of these subsets M' which enables such an efficient check, and later we will describe an enumeration of those subsets M'. When we check a subset M', we do not search for a single word $w \in M'$ with $\varphi(w) = R$ but for a finite set $F \subseteq M'$ with $\varphi(F) = R$. The soundness of this approach will be guaranteed by the existence of *1-word summaries:* A language $M' \subseteq \Delta^\star$ has 1-word summaries, if for all finite subsets $F \subseteq M'$ there exists a summary word $w \in M'$ with $\varphi(F) \subseteq \varphi(w)$. The property we exploit is given by the following proposition.

Proposition 37 (Membership Condition for Summarizable Languages, adapting [8])**.** *Let $M' \subseteq \Delta^\star$ be a regular language with 1-word summaries and $\varphi(M') \subseteq R$. Then there exists a $w \in M'$ with $\varphi(w) = R$ iff there exists a finite subset $F \subseteq M'$ with $\varphi(F) = \varphi(M') = R$.*

Proof. We show the two directions of the proposition statement separately. ($\Rightarrow$) With $w \in M'$ and $\varphi(w) = R$, taking $F = \{w\} \subseteq M'$, we obtain $R = \varphi(w) = \varphi(F) \subseteq \varphi(M') \subseteq R$, as required. ($\Leftarrow$) M' has 1-word summaries, hence there exists a $w \in M'$ with $\varphi(F) \subseteq \varphi(w)$, leading to $R = \varphi(F) \subseteq \varphi(w) \subseteq \varphi(M') \subseteq R$, as required. □

Putting it together. First, combining **(A.2)** and **(B)**, we obtain Lemma 39, to subdivide the search space $M[B, \varphi]$ into a set $\mathsf{rep}(M, B, \varphi)$ of languages

M' with 1-word summaries, as described in Definition 38. Second, in Theorem 40, building upon **(A.1)** and fixing $B = \mathsf{minlen}(R)$, we iterate through these languages M' using Lemma 39. We check each of them at once with our membership condition from Proposition 37. Mapping this approach to Algorithm 1, Lemma 39 provides the foundation for $\mathsf{enumerate}(K, R, \varphi)$ and Proposition 37 underlies $\mathsf{basiccheck}(R, M', \varphi)$ which returns true if there exists a finite $F \subseteq M'$ with $\varphi(F) = \varphi(M') = R$

Definition 38 (Summarizable Language Representation, adapting [8])**.** *For an integer $B \geq 0$, the* summarisable language representation $\mathsf{rep}(M, B, \varphi)$ *of $M \subseteq \Delta^\star$ with $\varphi : \Delta \to 2^{\Sigma^\star}$ is a family of union-free regular languages $M' \in \mathsf{rep}(M, B, \varphi)$ with 1-word summaries, such that $M[B, \varphi] \subseteq \bigcup_{M' \in \mathsf{rep}(M,B,\varphi)} M' \subseteq M$ holds.*

Lemma 39 (Existence of Summarizable Language Representations, adapting [8])**.** *For all regular languages $M \subseteq \Delta^\star$, regular language substitutions $\varphi : \Delta \to 2^{\Sigma^\star}$, and $B \geq 0$, there exists a summarizable language representation $\mathsf{rep}(M, B, \varphi)$.*

We show Lemma 39 as a consequence of Algorithm 4 in Section 6.3.

Theorem 40 (Membership Condition, following [8])**.** *Let $\mathcal{R} = (K, \varphi)$ be an RSRL and $\varphi : \Delta \to 2^{\Sigma^\star}$ be a regular language substitution. Then, for a regular language $R \subseteq \Sigma^\star$, we have $R \in \mathcal{R}$, iff there exists an $M' \in \mathsf{rep}(M_\varphi(R) \cap K, \mathsf{minlen}(R), \varphi)$ with a finite subset $F \subseteq M'$ with $\varphi(F) = \varphi(M') = R$.*

Proof. Most of the work for this proof is already achieved by the representation $\mathsf{rep}(M, \mathsf{minlen}(R), \varphi)$ of Lemma 39: The languages $M' \in \mathsf{rep}(M, \mathsf{minlen}(R), \varphi)$ are constructed to have 1-word summaries, allowing to check whether there exists $w \in M'$ with $\varphi(w) = R$ relatively easy – this is the case iff there exists a finite subset $F \subseteq M'$ with $\varphi(F) = \varphi(M') = R$. We show both directions of the theorem statement individually.

($\Rightarrow$) Assume $R \in \mathcal{R}$: By Definition 3, there exists $w \in K$ with $R = \varphi(w)$, by Definition 35, we get $w \in M_\varphi(R)$, and hence $w \in M_\varphi(R) \cap K = M$. From $R = \varphi(w)$ and $\mathsf{minlen}(R) = \mathsf{minlen}(\varphi(w))$, we get $w \in M[\mathsf{minlen}(R), \varphi]$. Since the maximal rewriting $M_\varphi(R)$ of a regular language R is regular as well [14], and since regular languages are closed under intersection, we obtain the regularity of M, and hence, Lemma 39 applies. Thus, there exists an $M' \in \mathsf{rep}(M, \mathsf{minlen}(R), \varphi)$ with $w \in M'$, and via Proposition 37, we obtain for $F = \{w\} \subseteq M'$, $R = \varphi(F) = \varphi(M')$, as required.

($\Leftarrow$) Assume that there exists an $M' \in \mathsf{rep}(M, \mathsf{minlen}(R), \varphi)$ with a finite subset $F \subseteq M'$ with $\varphi(F) = \varphi(M') = R$. Then, via Proposition 37, we take the summary word $w \in M'$ for F, yielding $R \in \mathcal{R}$, as required. □

We obtain the space complexity of membership, depending on the *size of the expressions* that represent the involved languages. More specifically, the complexity depends on the expression sizes $||R||$ and $||K||$ and the summed size $||\varphi|| = \Sigma_{\delta \in \Delta} ||\varphi(\delta)||$ of the expressions in the co-domain of φ.

Theorem 41 ($\mathsf{membership}(R, K, \varphi)$ runs in 2ExpSpace)**.** *More precisely, it runs in* DSpace $\left(||K||^r 2^{2^{(||R||+||\varphi||)^s}} \right)$ *for some constants* r *and* s*.*

We prove Theorem 41 in Section 6.4, relying on algorithms presented in Sections 6.2 and 6.3.

6.2. Implementing $\mathsf{basiccheck}(R, M', \varphi)$

Since Lemma 39 guarantees that union-free languages M' with 1-word summaries exist, we restrict our implementation to such languages and exploit these restrictions subsequently. We assume M' to be given in the form $M' = N_1 S_1^\star N_2 \dots N_m S_m^\star N_{m+1}$ with words $N_h \in \Delta^\star$ and union-free languages $S_h \subseteq \Delta^\star$. So, given such a language M' over Δ, and a regular language substitution $\varphi : \Delta \to 2^{\Sigma^\star}$, we need to check whether there exists a finite $F \subseteq M'$ with $\varphi(F) = \varphi(M') = R$. We implement this check with the procedure $\mathsf{basiccheck}(R, M', \varphi)$, splitting the condition of Proposition 37 into two parts, namely **(1)** whether there exists a finite $F \subseteq M'$ with $\varphi(F) = \varphi(M')$, and **(2)** whether $\varphi(M') = R$ holds. While the latter condition amounts to regular language equivalence, the former requires *distance automata* as additional machinery. Before introducing them formally below, we formalise the notation of a *path* of an automaton. To this effect, we denote a non-deterministic finite-state automaton (NFA) as a five tuple $(\Delta, Q, \rho, q_0, A)$, where Δ is the alphabet, Q is a finite set of states, $\rho \subseteq Q \times \Delta \times Q$ is the transition relation, $q_0 \in Q$ is the initial state, and $A \subseteq Q$ is the set of accepting states.

Definition 42 (Path)**.** *Given an NFA* $(\Delta, Q, \rho, q_0, A)$*, a* path π *is a sequence* $r_0 r_1 \dots r_k = (q_0, \delta_1, q_1)(q_1, \delta_2, q_2) \dots (q_k, \delta_{k+1}, q_{k+1})$ *of state transitions* r_i *taken from* ρ*, such that the state* q_{i+1} *reached by transition* r_i *is taken as originating state in the next transition* r_{i+1}*.*

If π *ends in an accepting state* $q_{k+1} \in A$*, then* π *is an* accepting path *for the word* $\delta_1 \delta_2 \dots \delta_{k+1}$*.*

To check whether there exists a suitable finite subset $F \subseteq M'$, we need to check whether we can cover $\varphi(M')$ while adhering to a finite limit on the number of specific Kleene-star back-edges taken. If this is the case, the correspondingly generated language F must be finite as well. Distance automata associate a distance with each edge. Thereby, each word is generated at a certain distance which is simply the sum of the individual distances of the edges taken along the shortest accepting path. Then, we construct an automaton for $\varphi(M')$ and associate with each back-edge that corresponds to the Kleene star of one of the $S_i^\star$, for $1 \leq i \leq m$, a distance of 1. All other edges are associated with distance 0. In particular, back-edges resulting from the automata for $\varphi(S_i)$ and $\varphi(N_j)$, for $1 \leq i \leq m$ and $1 \leq j \leq m+1$, are also associated with distance 0. If this automaton generates $\varphi(M')$ while obeying a limit on the maximal distance then there is a global bound on the number of iterations that are applied to any S_i and, therefore, it generates $\varphi(M')$ from some finite set $F \subseteq M'$.

Definition 43 (Distance Automaton [16])**.** *A* distance automaton *over an alphabet Δ is a tuple $\mathcal{A} = (\Delta, Q, \rho, q_0, A, d)$ where $(\Delta, Q, \rho, q_0, A)$ is an NFA and $d : \rho \to \{0, 1\}$ is a distance function, which can be extended to a function on words as follows. The distance function $d(\pi)$ of a path π is the sum of the distances of all transitions in π, i.e., $d(\pi) = \Sigma_{i=0}^{k} d(r_i)$ for $\pi = r_0 \dots r_k$ and $r_i \in \rho$. The distance $\mu(w)$ of a word $w \in L(\mathcal{A})$ is the minimum of $d(\pi)$ for all paths π accepting w.*

A distance automaton $\mathcal{A}$ is called limited *if there exists a constant U such that $\mu(w) < U$ for all words $w \in L(\mathcal{A})$.*

In our check for **(1)**, we build a distance automaton that is limited iff a finite F with $\varphi(F) = \varphi(M')$ exists. Then, we rely on the PSpace-decidability [30] of the limitedness of distance automata to check whether F exists or not.

Distance-automaton Construction. Here, we exploit the assumption that M' is a union-free language over Δ: Given the regular expression defining M', we construct the distance automaton $A_{M'}$ following the form of this regular expression:

- $\delta \in \Delta$: We construct the finite automaton A_δ with $L(A_\delta) = \varphi(\delta)$. We extend A_δ to a distance automaton by labelling each transition in A_{δ_i} with 0.

Algorithm 2: basiccheck(R, M', φ)

input : regular languages $R \subseteq \Sigma^\star$, $M' \subseteq \Delta^\star$, and regular language substitution φ with $\varphi(\delta) \subseteq \Sigma^\star$ for all $\delta \in \Delta$
requires : M' is union-free and $\varphi(M') \subseteq R$
returns : **true** iff $\exists$ finite $F \subseteq M' : \varphi(F) = \varphi(M') = R$

1 **build** $A_{M'}$;
2 **if** $A_{M'}$ *limited* **then**
3 **if** $\varphi(M') = R$ **then return true** ;
4 **return false**;

- $e{\cdot}f$: Given distance automata A_e and A_f with $A_e = (Q_e, \Sigma, \rho_e, q_{0,e}, F_e, d_e)$ and $A_f = (Q_f, \Sigma, \rho_f, q_{0,f}, F_f, d_f)$, we set $A_{e\cdot f} = (Q_e \uplus Q_f, \Sigma, \rho_e \cup \rho_f \cup \rho, q_{0,e}, F_f, d_{e\cdot f})$ where $\rho = \{(q, \varepsilon, q_{0,f}) \mid q \in F_e\}$ and $d_{e\cdot f} = d_e \cup d_f \cup \{(t, 0) \mid t \in \rho\}$, i.e., we connect each final state of A_e to the initial state of A_f and assign the distance 0 to these connecting transitions.

- $e^\star$: We construct the distance automaton $A_e = (Q_e, \Sigma, \rho_e, q_{0,e}, F_e, d_e)$. Then, $A_{e^\star} = (Q_e, \Sigma, \rho_e \cup \rho, q_{0,e}, F_e \cup \{q_{0,e}\}, d_{e^\star})$, where $\rho = \{(q, \varepsilon, q_{0,e}) \mid q \in F_e\}$ and $d_{e^\star} = d_e \cup \{((q, \varepsilon, p), 1) \mid (q, \varepsilon, p) \in \rho\}$, i.e., we connect each final state of A_e to the initial states of A_e and assign the corresponding transitions the distance 1.

If the resulting distance automaton $A_{M'}$ is limited, then there exists a finite subset $F \subseteq M'$ such that $\varphi(F) = \varphi(M')$. This implies that **(1)** holds.

So, given M' and R together with all languages in the domain of φ as regular expressions, basiccheck(R, M', φ) in Algorithm 2 first builds $A_{M'}$ (Line 1) and checks its limitedness (Line 2), amounting to condition **(1)**. For condition **(2)**, basiccheck verifies that $\varphi(M')$ and R are equivalent (Line 3) and returns **true** if both checks succeed.

Lemma 44 (basiccheck(R, M', φ) has PSPACE complexity)**.** *The algorithm* basiccheck(R, M', φ) *runs in* PSPACE *which is optimal as it solves a* PSPACE*-complete problem (assuming that* PSPACE *does not collapse with a lower class).*

Proof. Membership. The construction of the automaton $A_{M'}$ (Line 1) runs in polynomial time and hence produces a polynomially sized distance automaton.

Thus, the check for limitedness of $A_{M'}$ (Line 2) retains its PSPACE complexity [30]. Given M', R, and all $\varphi(\delta)$ for $\delta \in \Delta$ as regular expressions, we can build a polynomially sized regular expression for $\varphi(M')$ by substituting $\varphi(\delta)$ for each occurrence of δ in M'. Then we check the equivalence of the regular expressions for $\varphi(M')$ and R (Line 3), again keeping the original PSPACE complexity of regular expression equivalence [29]. This yields an overall PSPACE procedure.

Hardness. We reduce the PSPACE-complete problem of deciding whether a regular expression X over Σ is equivalent to $\Sigma^\star$ [29] to a single $\mathsf{basiccheck}$ invocation – proving that $\mathsf{basiccheck}$ solves a PSPACE complete problem. Given an arbitrary regular expression X, we set $M' = \{a\}$, $\varphi(a) = X$ and $R = \Sigma^\star$. Then $\mathsf{basiccheck}(R, M', \varphi)$ returns **true** iff X is equivalent to $\Sigma^\star$. □

6.3. Implementing $\mathsf{enumerate}(K, R, \varphi)$

Our enumeration algorithm must produce the languages $\mathsf{rep}(M, B, \varphi)$, guaranteeing that all $M' \in \mathsf{rep}(M, B, \varphi)$ have 1-word summaries, and that $M[B, \varphi] \subseteq \bigcup_{M' \in \mathsf{rep}(M,B,\varphi)} M' \subseteq M$ holds (as specified by Lemma 39). To this end, we rely on a sufficient condition for the existence of 1-word summaries. First we show the sufficiency of this condition with Proposition 45, before turning to the enumeration algorithm itself.

Proposition 45 (Sufficient Condition for 1-Word Summaries)**.** *Let L be a union-free language over Δ, given as $L = N_1 S_1^\star N_2 \dots N_m S_m^\star N_{m+1}$, with words $N_h \in \Delta^\star$ and union-free languages $S_h \subseteq \Delta^\star$. If $\varepsilon \in \varphi(w)$ for all $w \in S_h$ and all S_h, then L has 1-word summaries.*

Proof. We construct the desired word: Choose an arbitrary finite subset $F = \{f_1, \dots, f_p\} \subseteq L$. Then each word $f_i \in F$ is of the form

$$f_i = N_1 s_{1,i} N_2 \dots N_m s_{m,i} N_{m+1}$$

with $s_{h,i} \in S_h^\star$. We set $s_{h,F} = s_{h,1} \cdot s_{h,2} \cdots s_{h,p}$, and observe, because of $\varepsilon \in \varphi(w)$ for all $w \in S_h$ and S_h, we obtain

$$\begin{aligned} \varphi(s_{h,i}) &= \varepsilon \cdot \varphi(s_{h,i}) \cdot \varepsilon \\ &\subseteq \varphi(s_{h,1}) \cdots \varphi(s_{h,i-1}) \cdot \varphi(s_{h,i}) \cdot \varphi(s_{h,i+1}) \cdots \varphi(s_{h,p}) = \varphi(s_{h,F}) \,. \end{aligned}$$

Thus we choose the summary word $w = N_1 s_{1,F} N_2 \dots N_m s_{m,F} N_{m+1}$ and obtain $\varphi(f_i) = \varphi(N_1 s_{1,i} N_2 \dots N_m s_{m,i} N_{m+1}) \subseteq \varphi(w)$, and hence $\varphi(F) \subseteq \varphi(w)$. □

Algorithm 3: $\mathsf{enumerate}(R, K, \varphi)$

input : regular languages $R \subseteq \Sigma^\star$, $K \subseteq \Delta^\star$, and
regular language substitution φ with $\varphi(\delta) \subseteq \Sigma^\star$ for all $\delta \in \Delta$

yields : $L \in \mathsf{rep}(M, \mathsf{minlen}(R), \varphi)$ for $M = M_\varphi(R) \cap K$

1 $M := M_\varphi(R) \cap K$;

2 **for** $L \in \mathsf{unionfreedecomp}(M)$ **do** $\mathsf{unfold}(L, \varphi, \mathsf{minlen}(R))$;

Algorithm 4: $\mathsf{unfold}(L, \varphi, B)$

input : union-free regular language $L \subseteq \Delta^\star$, written as
$L = N_1 S_1^\star N_2 \dots N_m S_m^\star N_{m+1} \subseteq \Delta^\star$ with $N_i \in \Delta^\star$ and
union-free $S_h \subseteq \Delta^\star$, regular language substitution φ with
$\varphi(\delta) \subseteq \Sigma^\star$ for all $\delta \in \Delta$, and bound B

yields : $L' \in \mathsf{rep}(L, B, \varphi)$

1 **if** $\forall S_h \forall w \in S_h \ : \ \varepsilon \in \varphi(w)$ **then yield** L;

2 **else**

3 fix S_h arbitrarily with $\exists w \in S_h \ : \ \varepsilon \notin \varphi(w)$;

4 $E := S_h \cap \Delta_\varepsilon^\star$; // $\Delta_\varepsilon = \{\delta \in \Delta \mid \varepsilon \in \varphi(\delta)\}$

5 $L_0 := N_1 S_1^\star N_2 \dots \ N_h E^\star N_{h+1} \dots N_m S_m^\star N_{m+1}$;

6 $\mathsf{unfold}(L_0, \varphi, B)$;
// $L_p := N_1 S_1^\star N_2 \dots \ N_h E^\star \bar{E}_p S_h^\star N_{h+1} \dots N_m S_m^\star N_{m+1}$ (see text)

7 **for** $p \in \mathsf{critical}(S_h)$ *with* $\mathsf{minlen}(\varphi(L_p)) \leq B$ **do** $\mathsf{unfold}(L_p, \varphi, B)$;

We are ready to design our enumeration algorithm, shown in Algorithm 3, and its recursive subprocedure in Algorithm 4. These algorithms do not return a result but yield their result as an enumeration: Upon invocation, these algorithms run through a sequence of **yield** statements, each time appending the argument of **yield** to the enumerated sequence. Thus, the algorithm never stores the entire sequence but only the stack of the invoked procedures.

Initializing the recursive enumeration, Algorithm 3 obtains the maximum rewriting $M := M_\varphi(R) \cap K$ of R (Line 1) and iterates over the languages L in the union-free decomposition of M (Line 2) to call for each L the recursive procedure unfold. The union-free decomposition unionfreedecomp yields possibly exponentially many union-free languages, however, each of them has linear size, using the recursive rewriting rules, $(A+B)C = AC+BC$, $A(B+C) = AB+AC$, $(A+B)(C+D) = AC+AD+BC+BD$, and $(A+B)^\star = (A^\star B^\star)^\star$.[6]

In turn, the procedure unfold in Algorithm 4 takes a union-free language $L = N_1 S_1^\star N_2 \dots N_m S_m^\star N_{m+1}$ and a bound B to unfold the Kleene-star expressions of L until the precondition of Proposition 45 is satisfied or $\mathsf{minlen}(\varphi(L)) > B$. More specifically, unfold exploits a rewriting, based on the following terms: Given a union-free language S_h, let $E = S_h \cap \Delta_\varepsilon^\star$ with $\Delta_\varepsilon = \{\delta \in \Delta \mid \varepsilon \in \varphi(\delta)\}$ denote all words w in S_h with $\varepsilon \in \varphi(w)$ and let $\bar{E} = S_h - E$.

Proposition 46 (Rewriting for 1-Word Summaries (first step))**.** *For every union-free language $S_h^\star$ with $E = S_h \cap \Delta_\varepsilon^\star$, we have $S_h^\star = E^\star \cup E^\star \bar{E} S_h^\star$.*

Proof. $S_h^\star = E^\star(\bar{E}E^\star)^\star = E^\star \cup E^\star\bar{E}E^\star(\bar{E}E^\star)^\star = E^\star \cup E^\star\bar{E}S_h^\star$ □

Since $\bar{E}$ is in general not union free, we need to split $\bar{E}$ further. To this end, we define $\mathsf{ufs}(S_h, p)$ recursively for an integer sequence $p = \langle p_H \mid p_T \rangle$ with head element p_H and tail sequence p_T. Intuitively, a sequence p identifies a subexpression in S_h by recursively selecting a nested Kleene star expression; $\mathsf{ufs}(S_h, p)$ unfolds S_h such that this selected expression is instantiated at least once. Formally, for $S_h = N_1 S_1^\star N_2 \dots N_n S_n^\star N_{n+1}$ we set

$$\begin{aligned} \mathsf{ufs}(S_h, \varepsilon) &= S_h \\ \mathsf{ufs}(S_h, p) &= N_1 \dots N_{p_H} S_{p_H}^\star \mathsf{ufs}(S_{p_H}, p_T) S_{p_H}^\star N_{p_H+1} \dots N_{n+1} \ . \end{aligned}$$

[6] In practical implementations, however, one might prefer to generate less but larger individual expressions, employing, e.g., [31].

In the latter case, we replace $S^\star_{p_H}$ with $S^\star_{p_H}\mathsf{ufs}(S_{p_H}, p_T)S^\star_{p_H}$ such that there is at least one instantiation of $\mathsf{ufs}(S_{p_H}, p_T)$ within the otherwise possibly empty sequence of S_{p_H} instantiations.

Consider $S_h = A^\star(B^\star C^\star)^\star D^\star$ (with $S_1 = A, \dots S_4 = D$ and all $N_i = \varepsilon$), then we obtain

$$\begin{array}{rcllllll}
\mathsf{ufs}(S_h, \langle 2,1\rangle) &=& A^\star & (B^\star C^\star)^\star & & \mathsf{ufs}(B^\star C^\star, \langle 1\rangle) & & (B^\star C^\star)^\star \; D^\star \\
&=& A^\star & (B^\star C^\star)^\star & (B^\star \; \mathsf{ufs}(B, \varepsilon) \; B^\star & & C^\star) & (B^\star C^\star)^\star \; D^\star \\
&=& A^\star & (B^\star C^\star)^\star & (B^\star \quad (B) \quad B^\star & & C^\star) & (B^\star C^\star)^\star \; D^\star
\end{array}$$

instantiating B at position $\langle 2,1\rangle$ at least once in an otherwise equivalent expression.

Let $\mathsf{critical}(S_h)$ be the set of integer sequences that identify all subexpressions of S_h which directly contain a symbol δ with $\varepsilon \notin \varphi(\delta)$ (and not only via another Kleene-star expression). Since $\bar{E}$ is the subset of S_h of words which contain at least one such symbol δ, $\bar{E}$ equals the union of sublanguages of S_h which contain at least one instantiation of a subexpression identified by $\mathsf{critical}(S_h)$. Thus, we have $\bar{E} = \bigcup_{p \in \mathsf{critical}(S_h)} \bar{E}_p$ with $\bar{E}_p = \mathsf{ufs}(S_h, p)$.

For example, if we have $S_h = A'A^\star(B'B^\star C'C^\star)^\star D'D^\star$ (with $S_1 = A, \dots S_4 = D$ and $N_i = A', \dots N_4 = D'$), and B and C' both contain a symbol δ with $\varepsilon \notin \varphi(\delta)$, then $\mathsf{critical}(S_h) = \{\langle 2,1\rangle, \langle 2\rangle\}$, since

$$\begin{array}{rcl}
\mathsf{ufs}(S_h, \langle 2,1\rangle) &=& A'A^\star(B'B^\star C'C^\star)^\star(B'B^\star BB^\star C'C^\star)(B'B^\star C'C^\star)^\star D'D^\star \\
\mathsf{ufs}(S_h, \langle 2,1\rangle) &=& A'A^\star(B'B^\star C'C^\star)^\star(B'B^\star C'C^\star)(B'B^\star C'C^\star)^\star D'D^\star
\end{array}$$

are guaranteed to contain at least one instantiation of B and C' respectively. Thus, we obtain $\bar{E} = \mathsf{ufs}(S_h, \langle 2,1\rangle) \cup \mathsf{ufs}(S_h, \langle 2\rangle)$, as $\bar{E}$ is the subset of S_h consisting of words that contain at least one symbol δ with $\varepsilon \notin \varphi(\delta)$.

This discussion leads to the following rewriting:

Proposition 47 (Rewriting for 1-Word Summaries (second step))**.** *For every union-free language $S^\star_h$ with $E = S_h \cap \Delta^\star_\varepsilon$ for $\Delta_\varepsilon = \{\delta \in \Delta \mid \varepsilon \in \varphi(\delta)\}$ and $\bar{E}_p = \mathsf{ufs}(S_h, p)$, we have $S^\star_h = E^\star \cup \bigcup_{p \in \mathsf{critical}(S_h)} E^\star \bar{E}_p S^\star_h$.*

All languages in the rewriting, i.e., $E^\star$ and $E^\star \bar{E}_p S^\star_h$, are union free, $E^\star$ has 1-word summaries, and $\mathsf{minlen}(S^\star_h) < \mathsf{minlen}(E^\star \bar{E}_p S^\star_h)$ holds for all $p \in \mathsf{critical}(S_h)$.

Proof. Taking Proposition 46 and the discussion before Proposition 47, we obtain $S^\star_h = E^\star \cup E^\star \bar{E} S^\star_h = E^\star \cup \bigcup_{p \in \mathsf{critical}(S_h)} E^\star \bar{E}_p S^\star_h$. It remains to show the desired properties of the sublanguages in the rewriting: **(1)** *Union*

freeness: We construct the regular expression for $E^\star$ by dropping all Kleene-stared subexpressions in $S_h^\star$ which contain a symbol δ with $\varepsilon \notin \varphi(\delta)$ (possibly producing the empty language), preserving union freeness. The construction of $\bar{E}_p$ only unrolls Kleene star expressions, also preserving the union freeness from S_h. **(2)** *1-word summaries for $E^\star$:* For all $w \in E$, we have $\varepsilon \in \varphi(w)$, since all symbols δ in E have $\varepsilon \in \varphi(\delta)$. **(3)** *Increasing minimal length in $E^\star \bar{E}_p S_h^\star$:* Since $S_h^\star$ is a subexpression of $E^\star \bar{E}_p S_h^\star$ the minimal length can only increase, and since $\bar{E}_p$ instantiates an expression with a symbol δ and $\varepsilon \notin \varphi(\delta)$, it actually increases. □

If L already satisfies the precondition imposed by Proposition 45, Algorithm 4 **yield**s L and terminates (Line 1). Otherwise, it fixes an arbitrary S_h violating this precondition and rewrites L recursively with Proposition 47 (Lines 3-7). **(1)** *Termination:* In each recursive call, unfold either eliminates in L_0 an occurrence of a subexpression S_h violating the precondition of Proposition 45 (Line 6), or increases the minimum length in L_p, eventually running into the upper bound B (Line 7). **(2)** *Correctness:* Setting $B = \infty$, unfold **yield**s a possibly infinite sequence of union-free languages, which have 1-word summaries such that their union equals the original language L: As the generation of these languages is based on the equality of Proposition 47 each rewriting step is sound and complete, leading to an infinite recursion tree the leaves of which **yield** the languages in the sequence. The upper bound on minimum length only cuts off languages L_p producing words of minimum length beyond B, i.e., $L_p \cap L[B, \varphi] = \emptyset$, and in consequence, it is safe to drop L_p, since we only need to construct $\mathsf{rep}(L, B, \varphi)$ with $\mathsf{rep}(L, B, \varphi) \supseteq L[B, \varphi]$.

Algorithm 4 implements a representation, as required by Lemma 39, and hence is also proving this lemma.

Proof of Lemma 39. $\mathsf{unfold}(L, \varphi, B)$ yields $\mathsf{rep}(L, B, \varphi)$ for union-free languages L, hence using a union-free decomposition unionfreedecomp, we obtain $\mathsf{rep}(M, B, \varphi) = \bigcup_{L \in \mathsf{unionfreedecomp}(M)} \mathsf{unfold}(L, \varphi, B)$. □

6.4. Upper Bound of the Complexity

The proof of Theorem 41 is based on the size of the maximum rewriting $M = M_\varphi(R) \cap K$ of $||K||2^{2^{(||R||+||\varphi||)^l}}$ for some constant l, shown in [14], and unfold's complexity: In Proposition 48, we give an upper bound for the maximum size of the generated expressions $||\mathsf{ufs}(L, p)||$, where $||L||$ denotes the *length of the regular expression* representing L. Next, in Proposition 49,

we show an upper bound on the space complexity of unfold, leading to the complexity of enumerate in Lemma 50 and the desired proof of Theorem 41. Recall the definition of ufs for $L = N_1 S_1^\star N_2 \dots N_n S_n^\star N_{n+1}$ and $p = \langle p_H \mid p_T \rangle$ with $\mathsf{ufs}(L,p) = N_1 \dots N_{p_H} S_{p_H}^\star \mathsf{ufs}(S_{p_H}, p_T) S_{p_H}^\star N_{p_H+1} \dots N_{n+1}$.

Proposition 48 (Upper Bound for $||\mathsf{ufs}(L,p)||$)**.** *With κ as maximum length of a Kleene star subexpression in L, $||\mathsf{ufs}(L,p)|| = \mathcal{O}(\kappa \cdot ||L||)$ holds.*

Proof. ufs duplicates S_{p_H} of L and continues recursively on a third copy of S_{p_H}. Since ufs does not introduce new Kleene star subexpressions but only duplicates some, all Kleene star expressions occurring during the *entire* recursion are at most of length κ. Hence, each recursive step of ufs increases the length of the current expression at most by 2κ, and because of the Kleene star nesting depth of at most L, we obtain $||\mathsf{ufs}(L,p)|| = \mathcal{O}(\kappa \cdot ||L||)$. □

Proposition 49 ($\mathsf{unfold}(L,\varphi,B)$ runs in $\textsc{DSpace}(B^2||L||^4 + ||\varphi||)$)**.**

Proof. We denote with L_{init} the language given in the first call to unfold, while L denotes the language given to current call of unfold. We show the claim in four steps.

(1) $||L|| = \mathcal{O}(d||L_{\mathsf{init}}||^2)$ holds at any point during the recursion, given d is the number of recursive calls going through Line 7. First, recursive calls through Line 6 cannot increase the size of the expression, i.e., $||L_0|| \leq ||L||$, since we obtain L_0 by removing from $S_h^\star$ all subexpressions directly containing a symbol δ with $\varepsilon \in \varphi(\delta)$ (and not only via another Kleene star expression). Thus, only recursive calls going through Line 7 possibly increase the size of the expression. Now, in such a call, we unroll a subexpression S_h with $S_h^\star = E^\star \bar{E}_p S_h^\star$ and $\bar{E} = \mathsf{ufs}(S_h, p)$ for some integer sequence p. From Proposition 48, we have $||\mathsf{ufs}(S_h,p)|| = \mathcal{O}(\kappa||S_h||)$. Since ufs and unfold only duplicate already existing Kleene star subexpressions, we have both $||S_h|| \leq ||L_{\mathsf{init}}||$ and $\kappa \leq ||L_{\mathsf{init}}||$, and hence $||\mathsf{ufs}(S_h,p)|| = \mathcal{O}(||L_{\mathsf{init}}||^2)$. Together with $||E|| \leq ||S_h||$ and $||S_h|| \leq ||L_{\mathsf{init}}||$, this leads to $||E^\star \bar{E}_p S_h^\star|| = \mathcal{O}(||L_{\mathsf{init}}||^2)$. d recursive calls through Line 7 substitute d subexpressions S_h with $E^\star \bar{E}_p S_h^\star$ to unfold L_{init} into L, each time adding $\mathcal{O}(||L_{\mathsf{init}}||^2)$ to the size of the expression representing L. Hence $||L|| = \mathcal{O}(d||L_{\mathsf{init}}||^2)$.

(2) $||L|| = \mathcal{O}(B||L_{\mathsf{init}}||^2)$ holds for all recursive calls to unfold while computing $\mathsf{unfold}(L_{\mathsf{init}},\varphi,B)$. unfold makes at most B recursive steps through Line 7, since otherwise $\mathsf{minlen}(\varphi(L_p)) > \mathsf{minlen}(\varphi(L)) = B$ would hold (this is true, since $\bar{E}_p$ in L_p instantiates some δ with $\varepsilon \notin \varphi(\delta)$). Then this claim follows from the previous one by setting $d = B$.

(3) The total recursion depth of unfold is at most $\mathcal{O}(B||L_{\mathsf{init}}||^2)$. In the previous claim, we saw that there are at most B recursive calls through Line 7. It remains to give an upper bound for the calls through Line 6: In each such call, at least one Kleene star subexpression in L is removed in substituting E for S_h. At any point there are at most $||L|| = \mathcal{O}(B||L_{\mathsf{init}}||^2)$ subexpressions in L, hence we get a maximum recursion depth of $\mathcal{O}(B||L_{\mathsf{init}}||^2)$.

(4) The space required to compute $\mathsf{unfold}(L_{\mathsf{init}}, \varphi, B)$ is bounded by the depth of the recursion times the stack frame size, which is dominated by $||L||$, plus $||\varphi||$. This gives $\mathcal{O}\left((B||L_{\mathsf{init}}||^2)^2 + ||\varphi||\right) = \mathcal{O}(B^2||L_{\mathsf{init}}||^4 + ||\varphi||)$ as desired. □

Lemma 50 ($\mathsf{enumerate}(R, K, \varphi)$ runs in $\textsc{DSpace}\left(||K||^4 2^{2^{(||R||+||\varphi||)^k}}\right)$**.**

Proof. The construction of $M = M_\varphi(R) \cap K$ yields an expression in the size $||K||2^{2^{(||R||+||\varphi||)^l}}$ for some constant l [14]. The union-free decomposition yields possibly exponentially many union-free languages, however, each of them has linear size (see the description of Algorithm 3 for a simple but fitting decomposition). With Proposition 49, we obtain the overall space complexity of $\mathsf{enumerate}$ with $\textsc{DSpace}(B^2||L||^4 + ||\varphi||)$ for $B = \mathsf{minlen}(R) \leq ||R||$ and $||L|| = ||K||2^{2^{(||R||+||\varphi||)^k}}$. This leads to the desired result with $\textsc{DSpace}\left(||K||^4 2^{\cdot 2^{(||R||+||\varphi||)^k}}\right)$ for some other constant k. □

Proof of Theorem 41. The enumeration via $\mathsf{enumerate}(R, K, \varphi)$ runs in $\textsc{DSpace}\left(||K||^4 2^{2^{(||R||+||\varphi||)^k}}\right)$ (Lemma 50), producing expressions later fed into $\mathsf{basiccheck}$ at most of the same size. Since $\mathsf{basiccheck}$ is in $\textsc{PSpace}$ (Lemma 44), we obtain the overall complexity $\textsc{DSpace}\left(||K||^r 2^{2^{(||R||+||\varphi||)^s}}\right) \subseteq$ $2\textsc{ExpSpace}$ for some constants r and s. □

6.5. Differences to Afonin and Hazova [8]

Afonin and Hazova show that the membership problem is decidable. In determining an upper bound for the complexity of membership the problem, we had to expand their approach significantly: In general, we follow a top-down approach to describe the overall algorithm, whereas Afonin and Khazova go bottom-up, focusing on the building blocks enabling the decision procedure. More specifically, $\mathsf{basiccheck}$ is described in [8], while $\mathsf{enumerate}$ is omitted, as [8] deals with decidability only, deeming the bound on the enumeration size irrelevant. Hence Algorithms 3 and 4 are new, as well as the construction

in Section 6.3, leading to Proposition 47. Based on the new algorithms, we contribute Theorem 41, together with Proposition 49, and Lemma 50. Moreover, in [8], the overall algorithm and the proof for Theorem 40 are only described in a brief paragraph. Finally, Section 6.1, albeit technically not new, provides a much more conceptual and hopefully accessible description of the algorithm.

7. Conclusion

Motivated by applications in test case specifications with FQL, we have studied general and finite RSRLs. While we show that general RSRLs are not closed under most common operators, *finite* RSRLs are closed under all operators except Kleene stars and complementation (Theorem 14). This shows that our restriction to Kleene-star free and hence finite RSRLs in FQL results in a natural framework with good closure properties. Likewise, the proven PSpace-completeness results for Kleene-star free RSRLs provide a starting point to develop practical reasoning procedures for Kleene-star free RSRLs and FQL. Experience with LTL model checking shows that PSpace-completeness often leads to algorithms which are feasible in practice. In contrast, for general and possibly infinite RSRLs, we have described a 2ExpSpace membership checking algorithm – leaving the question for matching lower bounds open. Nevertheless, reasoning on general RSRLs seems to be rather infeasible.

Last but not least, RSRLs give rise to new and interesting research questions, for instance the decidability of inclusion and equivalence for general RSRLs, and the closure properties left open in this paper. In our future work, we want to generalize RSRLs to other base formalisms. For example, we want φ to substitute symbols by context-free expressions, thus enabling FQL test patterns to recognize, e.g., matching of parentheses or emptiness of a stack.

Acknowledgements This work received funding in part by the National Research Network RiSE on Rigorous Systems Engineering (Austrian Science Fund (FWF): S11403-N23), by the Vienna Science and Technology Fund (WWTF) through grant PROSEED, by an Erwin Schrödinger Fellowship (Austrian Science Fund (FWF): J3696-N26), and by the European Research Council under the European Community's Seventh Framework Programme (FP7/2007–2013) / ERC grant agreement DIADEM no. 246858.

References

[1] RTCA DO-178B, Software Considerations in Airborne Systems and Equipment Certification (1992).

[2] A. Holzer, C. Schallhart, M. Tautschnig, H. Veith, How did you specify your test suite, in: ASE, 2010, pp. 407–416.

[3] A. Holzer, V. Januzaj, S. Kugele, B. Langer, C. Schallhart, M. Tautschnig, H. Veith, Seamless Testing for Models and Code, in: FASE'11, 2011, pp. 278–293.

[4] A. Holzer, M. Tautschnig, C. Schallhart, H. Veith, An Introduction to Test Specification in FQL, in: HVC, 2010, pp. 9–22.

[5] A. Holzer, M. Tautschnig, C. Schallhart, H. Veith, Query-Dirven Program Testing, in: VMCAI, 2009, pp. 151–166.

[6] A. Holzer, M. Tautschnig, C. Schallhart, H. Veith, FSHELL: Systematic Test Case Generation for Dynamic Analysis and Measurement, in: CAV, 2008, pp. 209–213.

[7] D. Beyer, A. Holzer, M. Tautschnig, H. Veith, Information Reuse for Multi-goal Reachability Analyses, in: ESOP, 2013, pp. 472–491.

[8] S. Afonin, E. Hazova, Membership and Finiteness Problems for Rational Sets of Regular Languages, in: DLT, 2005, pp. 88–99.

[9] A. Holzer, C. Schallhart, M. Tautschnig, H. Veith, On the Structure and Complexity of Rational Sets of Regular Languages, in: FSTTCS, 2013, pp. 377–388.

[10] S. Eilenberg, M. P. Schützenberger, Rational sets in commutative monoids., J. Algebra 13 (1969) 173–191. doi:10.1016/0021-8693(69)90070-2.

[11] S. C. Kleene, Representation of Events in Nerve Nets and Finite Automata, RAND Corporation Memorandum.

[12] A. Holzer, D. Kroening, C. Schallhart, M. Tautschnig, H. Veith, Proving Reachability using FShell (Competition Contribution), in: TACAS, 2012, pp. 538–541.

[13] T. A. Henzinger, R. Jhala, R. Majumdar, G. Sutre, Lazy abstraction, in: POPL, 2002, pp. 58–70.

[14] D. Calvanese, G. D. Giacomo, M. Lenzerini, M. Y. Vardi, Rewriting of Regular Expressions and Regular Path Queries, JCSS 64 (2002) 443–465.

[15] K. Hashiguchi, Representation Theorems on Regular Languages, J. Comput. Syst. Sci. 27 (1) (1983) 101–115.

[16] K. Hashiguchi, Limitedness Theorem on Finite Automata with Distance Functions, J. Comput. Syst. Sci. 24 (2) (1982) 233–244.

[17] D. Krob, The equality problem for rational series with multiplicities in the tropical semiring is undecidable, Intl. Journal of Algebra and Computation 4 (3) (1994) 405–425.

[18] T. Colcombet, L. Daviaud, Approximate Comparison of Distance Automata, in: STACS, 2013, pp. 574–585.

[19] D. Kirsten, Distance Desert Automata and the Star Height One Problem, in: FoSSaCS, 2004, pp. 257–272.

[20] D. Kirsten, Distance Desert Automata and the Star Height Problem, ITA 39 (3) (2005) 455–509.

[21] J. Berstel, Transductions and Context-Free Languages, Teubner Studienbücher, Stuttgart, 1979.

[22] J.-E. Pin, Mathematical foundations of automata theory, Lecture Notes (2011).

[23] P. Barceló, D. Figueira, L. Libkin, Graph Logics with Rational Relations and the Generalized Intersection Problem, in: LICS, 2012, pp. 115–124.

[24] P. Barceló, J. L. Reutter, L. Libkin, Parameterized regular expressions and their languages, Theor. Comput. Sci. 474 (2013) 21–45.

[25] G. A. Agasandyan, Variable-Structure Automata, Soviet Physics Doklady.

[26] A. Salomaa, On finite automata with a time-variant structure, Information and Control 13 (2) (1968) 85 – 98. doi:10.1016/S0019-9958(68)90706-7.
URL http://www.sciencedirect.com/science/article/pii/S0019995868907067

[27] O. Kupferman, Y. Lustig, Lattice Automata, in: VMCAI, 2007, pp. 199–213.

[28] J. E. Hopcroft, J. D. Ullman, Introduction to Automata Theory, Languages and Computation, Addison-Wesley, 1979.

[29] A. R. Meyer, L. J. Stockmeyer, The Equivalence Problem for Regular Expressions with Squaring Requires Exponential Space, in: SWAT (FOCS), 1972, pp. 125–129.

[30] H. Leung, V. Podolskiy, The limitedness problem on distance automata: Hashiguchi's method revisited, TCS 310 (1–3) (2004) 147–158.

[31] S. Afonin, D. Golomazov, Minimal Union-Free Decomposition of Regular Languages, in: LATA, 2009, pp. 83–92.

Andreas Holzer is a post-doctoral researcher at the University of Toronto. His current work focuses on automated systematic test generation techniques and compiler construction for secure computations. He developed the test generation tools CPAtiger and ConCrest as well as the first C compiler for secure two-party computations CBMC-GC. Before he joined the University of Toronto, he was a post-doctoral researcher at Vienna University of Technology where he also obtained his PhD.

Christian Schallhart has recently joined Google London to work on text-to-speech systems. Before that, he has built systems for automated web data extraction to fuel the exploding need for data. Within the Diadem project at Oxford University, he was developing unsupervised systems to autonomously explore sites, identify the relevant data, and induce wrappers for this data. Prior to that, he was working on test case generation and runtime verification at TU Darmstadt and Munich. He obtained his PhD at Vienna University of Technology after spending some time in industry.

Michael Tautschnig is a lecturer at Queen Mary University of London. He obtained his PhD from Vienna University of Technology developing an efficient, systematic, and automatic test input generation technique for C programs. Prior to his appointment as lecturer he was a post-doctoral researcher at the University of Oxford. His current and recent research focuses on automating verification for real-world software at large scale, and efficient software verification techniques for concurrent programs, shared memory systems with relaxed memory models.

Helmut Veith is a professor at the Faculty of Informatics of Vienna University of Technology (TU Vienna), and an adjunct professor at Carnegie Mellon University. He has a diploma in Computational Logic and a PhD sub auspiciis praesidentis in Computer Science, both from Vienna University of Technology. Prior to his appointment to Vienna, he was holding professor positions at TU Darmstadt and TU Munich. In his research, Helmut Veith applies formal and logical methods to problems in software technology and engineering. His current work is focussing on model checking, software verification and testing, embedded software and computer security.