Symbolic Solving of Extended Regular Expression Inequalities

Matthias Keil and Peter Thiemann

Institute for Computer Science
University of Freiburg
Freiburg, Germany
{keilr,thiemann}@informatik.uni-freiburg.de

Abstract

This paper presents a new algorithm for the containment problem for *extended regular expressions* that contain intersection and complement operators and that range over infinite alphabets. The algorithm solves extended regular expressions inequalities symbolically by term rewriting and thus avoids the translation to an expression-equivalent automaton.

Our algorithm is based on Brzozowski's regular expression derivatives and on Antimirov's term-rewriting approach to check containment. To deal with large or infinite alphabets effectively, we generalize Brzozowski's derivative operator to work with respect to (potentially infinite) representable character sets.

1998 ACM Subject Classification F.4.3 Formal Languages

Keywords and phrases Extended regular expression, containment, infinite alphabet, infinite character set, effective boolean algebra

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

Regular expressions have many applications in the context of software development and information technology: text processing, program analysis, compiler construction, query processing, and so on. Modern programming languages either come with standard libraries for regular expression processing or they provide built-in facilities to this end (e.g., Perl, Ruby, and JavaScript). Many of these implementations augment the basic regular operations $+$, $\cdot$, and * (union, concatenation, and Kleene star) with enhancements like character classes and wildcard literals, cardinalities, sub-matching, intersection, complement, and so on.

Regular expressions (RE) are advantageous in these domains because they provide a concise means to encode many interesting problems. REs are well suited for verification applications, because there are decision procedures for many problems involving them: the word problem ($w \in [\![r]\!]$), emptiness ($[\![r]\!] = \emptyset$), finiteness, containment ($[\![r]\!] \subseteq [\![s]\!]$), and equivalence ($[\![r]\!] = [\![s]\!]$). Here we let r and s range over RE and write $[\![\cdot]\!]$ for the function that maps a regular expression to the regular language that it denotes. There are also effective constructions for operations like union, intersection, complement, prefixes, suffixes, etc on regular languages.

Recent applications impose new demands on operations involving regular expressions. The Unicode character set with its more than 1.1 million code points requires the ability to deal effectively with very large character sets and hence character classes. Similarly, formalizing access contracts for objects in scripting languages even requires regular expressions over an infinite alphabet: in this application, the alphabet itself is an infinite formal language (the language of field names) and a "character class" (i.e., a set of field names) is

Conference title on which this volume is based on.
Editors: Billy Editor and Bill Editors; pp. 1–12

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

itself described by a regular expression [12, 8]. Hence, a "character class" may have infinitely many elements.

To enable such applications, we study the containment problem for regular expressions with two enhancements. First, we consider *extended regular expressions* (ERE) that contain intersection and complement operators beyond the standard regular operators of union, concatenation, and Kleene star. An ERE also denotes a regular language but it can be much more concise than a standard RE. Second, we consider EREs on any alphabet that is presented as an effective boolean algebra. This extension encompasses some practically useful instances of infinite alphabets like the set of all field names in a scripting language.

The first enhancement is known to be decidable, but we give a new symbolic decision procedure based on Brzozowski's regular expression derivatives [4] and Antimirov's rewriting approach to check containment [1]. The second enhancement has been studied previously [21, 19, 20], but in the context of automata and finite state transducers. It has not been investigated at the level of regular expressions and in particular not in the context of Brzozowski's and Antimirov's work. We give sufficient conditions to ensure applicability of our modification of Brzozowski's and Antimirov's approach to the containment problem while retaining decidability.

1.1 Related Work

The practical motivation for considering this extension is drawn from the authors' previous work on checking access contracts for objects in a scripting language at run time [12]. In that work, an access contract specifies a set of access paths that start from a specific anchor object. An access path is a word over the field names of the objects traversed by the path and we specify such a set of paths by a regular expression on the field names. We claim that such a regular expression draws from an infinite alphabet because a field name in a scripting language is an arbitrary string (of characters). For succinctness, we specify sets of field names using a second level of regular expressions on characters.

In our implementation, checking containment is required to reduce memory consumption. If the same object is restricted by more than one contract, then we apply containment checking to remove redundant contracts. In our previous work, contracts were limited to basic regular expressions and the field-level expressions were limited to disjunctions of literals. Applying the results of the present paper enables us to lift both restrictions.

The textbook approach to checking regular expression containment is via translation to finite automata, which may involve an exponential blowup, and then by constructing a simulation (or a bisimulation for equivalence) [9]. A related approach based on non-deterministic automata is presented by Bonchi and Pous [3].

The exponential blowup is due to the construction of a deterministic automaton from the regular expression. Thompson's construction [18], creates a non-deterministic finite automaton with ϵ-transitions where the number of states and transitions is linear to the length of the (standard) regular expression. Glushkov's [7] and McNaughton and Yamada's [14] position automaton computes an $n + 1$-state non-deterministic automaton with up to n^2 transitions from an n-symbol expression. They are the first to use the notion of a first symbol. Brzozowski's regular expression derivatives [4] directly calculate a deterministic automaton from an ERE. Antimirov's partial derivative approach [2] computes a $n+1$-state non-deterministic automation, but his work does not consider intersection and complement. We are not aware of an extension of Glushkov's algorithm to extended regular expressions.

Owens and other have implemented an extension of Brzozowski's approach with character classes and wildcards [16].

Antimirov [1] also proposes a symbolic method for solving regular expression inequalities, based on partial derivatives, with exponential worst-case run time. His *containment calculus* is closely related to the simulation technique used by Hopcroft and Karp [9] for proving equivalence of automata. In fact, a decision procedure for containment of regular expressions leads to one for equivalence and vice versa. Ginzburg [6] gives an equivalence procedure based on Brzozowski derivatives. Antimirov's original work does not consider intersection and complement. Caron and coworkers [5] extend Antimirov's work to ERE using antichains, but the resulting procedure is very complex compared to ours.

A shortcoming of all existing approaches is their restriction to finite alphabets. Supporting both makes a significant difference in practice: an iteration over the alphabet Σ is feasible for small alphabets, but it is impractical for very large alphabets (e.g., Unicode) or infinite ones (e.g., another level of regular languages as for our contracts). Furthermore, most regular expressions used in practice contain character sets. We apply techniques developed for symbolic finite automata to address these issues [20].

1.2 Overview

This paper is organized as follows. In Section 2, we recall notations and concepts. Section 3 introduces the notion of an effective boolean algebra for representing sets of symbols abstractly. Section 4 explains Antimirov's algorithm for checking containment, which is the starting point of our work. Next, Section 5 defines two notions of derivatives on regular expressions with respect to symbol sets. It continues to introduce the key notion of *next literals*, which ensures finiteness of our extension to Antimirov's algorithm. Section 6 contains the heart of our extended algorithm, a deduction system that determines containment of extended regular expressions along with a soundness proof.

A technical report [13] extends this paper by an appendix with further technical details, examples, and proofs of theorems.

2 Regular Expressions

An *alphabet* Σ is a denumerable, potentially infinite set of symbols. Σ^* is the set of all finite words over symbols from Σ with ϵ denoting the empty word. Let $a, b, c \in \Sigma$ range over symbols; $u, v, w \in \Sigma^*$ over words; and $A, B, C \subseteq \Sigma$ over sets of symbols.

Let $\mathcal{L}, \mathcal{L}' \subseteq \Sigma^*$ be languages. The *left quotient* of $\mathcal{L}$ by a word u, written $u^{-1}\mathcal{L}$, is the language $\{v \mid uv \in \mathcal{L}\}$. It is immediate from the definition that $(au)^{-1}\mathcal{L} = u^{-1}(a^{-1}\mathcal{L})$ and that $u \in \mathcal{L}$ iff $\epsilon \in u^{-1}\mathcal{L}$. Furthermore, $\mathcal{L} \subseteq \mathcal{L}'$ iff $u^{-1}\mathcal{L} \subseteq u^{-1}\mathcal{L}'$ for all words $u \in \Sigma^*$. The left quotient of one language by another is defined by $\mathcal{L}^{-1}\mathcal{L}' = \{v \mid uv \in \mathcal{L}', u \in \mathcal{L}\}$. We write $\mathcal{L}{\cdot}\mathcal{L}'$ for the concatenation of languages $\{uv \mid u \in \mathcal{L}, v \in \mathcal{L}'\}$ and $\mathcal{L}^*$ for the Kleene closure $\{v_1 \dots v_n \mid n \in \mathbb{N}, v_i \in \mathcal{L}\}$. We sometimes write $\overline{\mathcal{L}}$ for the complement $\Sigma^* \setminus \mathcal{L}$ and $\overline{A}$ for $\Sigma \setminus A$.

An *extended regular expression* (ERE) on an alphabet Σ is a syntactic phrase derivable from non-terminals r, s, t. It comprises the the empty word, literals, union, concatenation, Kleene star, as well as intersection and negation operators.

$$r, s, t \;:=\; \epsilon \mid A \mid r{+}s \mid r{\cdot}s \mid r^* \mid r\&s \mid !r$$

Compared to standard definitions, a *literal* is a set A of symbols, which stands for an abstract, possibly empty, character class. We write a instead of $\{a\}$ for the frequent case of a single letter literal. We consider regular expressions up to similarity [4], that is, up to associativity and commutativity of the union operator with the empty set as identity.

The language $[\![r]\!] \subseteq \Sigma^*$ of a regular expression r is defined inductively by:

$$\begin{array}{lll} [\![\epsilon]\!] = \{\epsilon\} & [\![r{+}s]\!] = [\![r]\!] \cup [\![s]\!] & [\![r\&s]\!] = [\![r]\!] \cap [\![s]\!] \\ [\![A]\!] = \{a \mid a \in A\} & [\![r{\cdot}s]\!] = [\![r]\!]{\cdot}[\![s]\!] & [\![!r]\!] = \overline{[\![r]\!]} \\ & [\![r^*]\!] = [\![r]\!]^* & \end{array}$$

For finite alphabets, $[\![r]\!]$ is a regular language. For arbitrary alphabets, we *define* a language to be regular, if it is equal to $[\![r]\!]$, for some ERE r.

We write $r \sqsubseteq s$ (r is *contained* in s) to express that $[\![r]\!] \subseteq [\![s]\!]$.

The *nullable* predicate $\nu(r)$ indicates whether $[\![r]\!]$ contains the empty word, that is, $\nu(r)$ iff $\epsilon \in [\![r]\!]$. It is defined inductively by:

$$\begin{array}{llll} \nu(\epsilon) = \mathit{true} & \nu(r{+}s) = \nu(r) \vee \nu(s) & \nu(r\&s) = \nu(r) \wedge \nu(s) \\ \nu(A) = \mathit{false} & \nu(r{\cdot}s) = \nu(r) \wedge \nu(s) & \nu(!r) = \neg\nu(r) \\ & \nu(r^*) = \mathit{true} & \end{array}$$

The *Brzozowski derivative* $\partial_a(r)$ of an ERE r w.r.t. a symbol a computes a regular expression for the left quotient $a^{-1}[\![r]\!]$ (see [4]). It is defined inductively as follows:

$$\begin{array}{ll} \partial_a(\epsilon) = \emptyset & \partial_a(r{\cdot}s) = \begin{cases} \partial_a(r){\cdot}s{+}\partial_a(s), & \nu(r) \\ \partial_a(r){\cdot}s, & \neg\nu(r) \end{cases} \\ \partial_a(A) = \begin{cases} \epsilon, & a \in A \\ \emptyset, & a \notin A \end{cases} & \partial_a(r^*) = \partial_a(r){\cdot}r^* \\ \partial_a(r{+}s) = \partial_a(r){+}\partial_a(s) & \partial_a(r\&s) = \partial_a(r)\&\partial_a(s) \\ & \partial_a(!r) = !\partial_a(r) \end{array}$$

The case for the set literal A generalizes Brzozowski's definition. The definition is extended to words by $\partial_{au}(r) = \partial_u(\partial_a(r))$ and $\partial_\epsilon(r) = r$. It is easy to see that $u \in [\![r]\!]$ iff $\epsilon \in [\![\partial_u(r)]\!]$.

3 Representing Sets of Symbols

The definition of an ERE in Section 2 just states that a literal is a set of symbols $A \subseteq \Sigma$. However, to define tractable algorithms, we require that A is an element of an effective boolean algebra [20] $(U, \sqcup, \sqcap, \bar{\cdot}, \bot, \top)$ where $U \subseteq \wp(\Sigma)$ is closed under the boolean operations. Here $\sqcup$ and $\sqcap$ denote union and intersection of symbol sets, $\bar{\cdot}$ the complement, and $\bot$ and $\top$ the empty set and the full set Σ, respectively. In this algebra, we need to be able to decide equality of sets (hence the term *effective*) and to represent singleton symbols.

- For finite (small) alphabets, we may just take $U = \wp(\Sigma)$. A set of symbols may be enumerated and ranges of symbols may be represented by character classes, as customarily supported in regular expression implementations. Alternatively, a bitvector representation may be used.
- If the alphabet is infinite (or just too large), then the boolean algebra of finite and cofinite sets of symbols is the basis for a suitable representation. That is, the set $U = \{A \in \wp(\Sigma) \mid A \text{ finite} \vee \overline{A} \text{ finite}\}$ is effectively closed under the boolean operations.
- In our application to checking access contracts in scripting languages [12], the alphabet itself is a set of words (the field names of objects) composed from another set Γ of symbols: $\Sigma \subseteq \wp(\Gamma^*)$. To obtain an effective boolean algebra, we choose the set $U = \{A \subseteq \wp(\Gamma^*) \mid A \text{ is regular}\}$, which is effectively closed under the boolean operations.
- Sets of symbols may also be represented by formulas drawn from a decidable first-order theory over a (finite or infinite) alphabet. For example, the character range `[a-z]` would be represented by the formula $x \geq$ `'a'` $\wedge x \leq$ `'z'`. In this case, the boolean operations get

mapped to the disjunction, conjunction, or negation of predicates; bottom and top are false and true, respectively. An SMT solver can decide equality and subset constraints. This approach has been demonstrated to be effective for very large character sets in the work on symbolic finite automata [20].

The rest of this paper is generic with respect to the choice of an effective boolean algebra.

4 Antimirov's algorithm for checking containment

Given two regular expressions r, s, the *containment problem* asks whether $r \sqsubseteq s$. This problem is decidable using standard techniques from automata theory: construct a deterministic finite automaton for $r\&!s$ and check it for emptiness. The drawback of this approach is the expensive construction of the automaton. In general, this expense cannot be avoided because problem is PSPACE-complete [10, 11, 15].

Antimirov [1] proposed an algorithm for deciding containment of standard regular expressions (without intersection and negation) that is based on rewriting of inequalities. His algorithm has the same asymptotic complexity as the automaton construction, but it can fail early and is therefore better behaved in practice. We phrase the algorithm in terms of Brzozowski derivatives to avoid introducing Antimirov's notion of partial derivatives.

▶ **Theorem 1** (Containment [1, Proposition 7(2)]). *For regular expressions r and s,*

$$r \sqsubseteq s \Leftrightarrow (\forall u \in \Sigma^*)\ \partial_u(r) \sqsubseteq \partial_u(s).$$

Antimirov's algorithm applies this theorem exhaustively to an inequality $r \dot{\sqsubseteq} s$ (i.e., a proposed containment) to generate all pairs $\partial_u(r) \dot{\sqsubseteq} \partial_u(s)$ of iterated derivatives until it finds a contradiction or saturation. More precisely, Antimirov defines a *containment calculus* $\mathcal{CC}$ which works on sets S of atoms, where an atom is either an inequality $r \dot{\sqsubseteq} s$ or a boolean constant *true* or *false*. It consists of the rule CC-Disprove which infers *false* from a trivially inconsistent inequality and the rule CC-Unfold that applies Theorem 1 to generate new inequalities.

$$\text{CC-Disprove}\ \frac{\nu(r) \wedge \neg\nu(s)}{r \dot{\sqsubseteq} s \vdash_{\mathcal{CC}} \mathit{false}} \qquad \text{CC-Unfold}\ \frac{\nu(r) \Rightarrow \nu(s)}{r \dot{\sqsubseteq} s \vdash_{\mathcal{CC}} \{\partial_a(r) \dot{\sqsubseteq} \partial_a(s) \mid a \in \Sigma\}}$$

An inference in the calculus for checking whether $r_0 \sqsubseteq s_0$ is a sequence $S_0 \vdash_{\mathcal{CC}} S_1 \vdash_{\mathcal{CC}} S_2 \vdash_{\mathcal{CC}} \ldots$ where $S_0 = \{r_0 \dot{\sqsubseteq} s_0\}$ and S_{i+1} is an extension of S_i by selecting an inequality in S_i and adding the consequences of applying one of the $\mathcal{CC}$ rules to it. That is, if $r \dot{\sqsubseteq} s \in S_i$ and $r \dot{\sqsubseteq} s \vdash_{\mathcal{CC}} S$, then $S_{i+1} = S_i \cup S$.

Antimirov argues [1, Theorem 8] that this algorithm is sound and complete by proving (using Theorem 1) that $r \sqsubseteq s$ does not hold if and only if a set of atoms containing *false* is derivable from $r \dot{\sqsubseteq} s$. The algorithm terminates because there are only finitely many different inequalities derivable from $r \dot{\sqsubseteq} s$ using rule CC-Unfold.

The containment calculus $\mathcal{CC}$ has two drawbacks. First, the choice of an inequality for the next inference step is nondeterministic. Second, an adaptation to a setting with an infinite alphabet seems doomed because rule CC-Unfold requires us to compute the derivative for infinitely many $a \in \Sigma$ at each application. We address the second drawback next.

5 Derivatives on Literals

In this section, we develop a variant of Theorem 1 that enables us to define a variant of the CC-Unfold rule that is guaranteed to add finitely many atoms, even if the alphabet is

infinite. First, we observe that we may restrict the symbols considered in rule CC-Unfold to initial symbols of the left hand side of an inequality.

▶ **Definition 2** (First). Let $\mathsf{first}(r) := \{a \mid aw \in [\![r]\!]\}$ be the set of initial symbols derivable from regular expression r.

Clearly, $(\forall a \in \Sigma)\ \partial_a(r) \sqsubseteq \partial_a(s)$ iff $(\forall b \in \mathsf{first}(r))\ \partial_b(r) \sqsubseteq \partial_b(s)$ because $\partial_b(r) = \emptyset$ for all $b \notin \mathsf{first}(r)$. Thus, CC-Unfold does not have to consider the entire alphabet, but unfortunately $\mathsf{first}(r)$ may still be an infinite set of symbols. For that reason, we propose to compute derivatives with respect to *literals* (i.e., non-empty sets of symbols) instead of single symbols. However, generalizing derivatives to literals has some subtle problems.

To illustrate these problems, let us recall the specification of the Brzozowski derivative:

$$[\![\partial_a(r)]\!] = a^{-1}[\![r]\!]$$

We might be tempted to consider the following naive extension of the derivative to a set of symbols A.

$$[\![\partial_A(r)]\!] \;=\; A^{-1}[\![r]\!] \;=\; \bigcup_{a\in A} a^{-1}[\![r]\!] = \bigcup_{a\in A} [\![\partial_a(r)]\!] \qquad \text{(wrong)}$$

However, this attempt at a specification yields inconsistent results. To see why, consider the case where $r = !s$. Generalizing from $\partial_a(!s) = !\partial_a(s)$, we might try to define $\partial_A(!s) := !\partial_A(s)$. If this definition was sensible, then (1) and (2) should yield the same results:

$$[\![\partial_A(!s)]\!] \overset{\text{(wrong)}}{=} \bigcup_{a\in A} [\![\partial_a(!s)]\!] \overset{\text{def } \partial_a}{=} \bigcup_{a\in A} \overline{[\![\partial_a(s)]\!]} \tag{1}$$

$$[\![!\partial_A(s)]\!] \overset{\text{def } \partial_a}{=} \overline{[\![\partial_A(s)]\!]} \overset{\text{(wrong)}}{=} \overline{\bigcup_{a\in A} [\![\partial_a(s)]\!]} \overset{\text{de Morgan}}{=} \bigcap_{a\in A} \overline{[\![\partial_a(s)]\!]} \tag{2}$$

However, we obtain a contradiction: with $A = \{a, b\}$ and $s = a{\cdot}a{+}b{\cdot}b$, (1) yields Σ^* whereas (2) yields $\overline{\{a, b\}}$, which is clearly different.

5.1 Positive and Negative Derivatives

To address this problem, we introduce two types of derivative operators with respect to symbol sets. The *positive derivative* $\Delta_A(r)$ computes an expression that contains the union of all $\partial_a(r)$ with $a \in A$, whereas the *negative derivative* $\nabla_A(r)$ computes an expression contained in the intersection of all $\partial_a(r)$ with $a \in A$.

The positive and negative derivative operators are defined by mutual induction and flip at the complement operator. Most cases of their definition are identical to the Brzozowski derivative (cf. Section 2), thus we only show the cases that are different. For all literals A with $[\![A]\!] \neq \emptyset$:

$$\begin{array}{lcl@{\qquad}lcl} \Delta_B(A) & := & \begin{cases}\epsilon, & A \sqcap B \neq \bot \\ \emptyset, & \textit{otherwise}\end{cases} & \nabla_B(A) & := & \begin{cases}\epsilon, & \overline{A} \sqcap B = \bot \\ \emptyset, & \textit{otherwise}\end{cases} \\ \Delta_B(!r) & := & !\nabla_B(r) & \nabla_B(!r) & := & !\Delta_B(r) \end{array}$$

For single symbol literals of the form $B = \{a\}$, it holds that $\Delta_a(r) = \nabla_a(r) = \partial_a(r)$. Derivatives with respect to the empty set are defined as $\Delta_\emptyset(r) = \emptyset$ and $\nabla_\emptyset(r) = \Sigma^*$.

The following lemma states the connection between the derivative by a literal and the derivative by a symbol.

▶ **Lemma 3** (Positive and negative derivatives). *For any r and B, it holds that:*

$$[\![\Delta_B(r)]\!] \supseteq \bigcup_{a\in B} [\![\partial_a(r)]\!] \qquad\qquad [\![\nabla_B(r)]\!] \subseteq \bigcap_{a\in B} [\![\partial_a(r)]\!]$$

Proof of Lemma 3. Both inclusions are proved simultaneously by induction on r. ◀

The following examples illustrate the properties of the derivatives.

▶ **Example 4** (Positive derivative). Let r be $(a \cdot c)\&(b \cdot c)$ and let the literal $A = \{a, b\}$.

$$\Delta_A(r) = \Delta_A(a \cdot c)\&\Delta_A(b \cdot c) = c\&c \sqsupseteq \partial_a(r)+\partial_b(r) = \emptyset+\emptyset$$

▶ **Example 5** (Negative derivative). Let r be $(a \cdot c)+(b \cdot c)$ and let the literal $A = \{a, b\}$.

$$\nabla_A(r) = \nabla_A(a \cdot c)+\nabla_A(b \cdot c) = \emptyset+\emptyset \sqsubseteq \partial_a(r)\&\partial_b(r) = c\&c$$

Positive (negative) derivatives yield an upper (lower) approximation to the information expected from a derivative. This approximation arises because we tried to define the derivative with respect to an *arbitrary* literal A. To obtain the precise information, we need to restrict these literals suitably to *next literals*.

5.2 Next Literals

An occurrence of a literal A in a regular expression r is *initial* if there is some $a \in \Sigma$ such that $\partial_a(r)$ reduces this occurrence. That is, the computation of $\partial_a(r)$ involves $\partial_a(A)$. Intuitively, A helps determine the first symbol of an element of $[\![r]\!]$.

▶ **Example 6** (Initial Literals).
1. Let $r_1 = \{a, b\}.a^*$. Then $\{a, b\}$ is an initial literal.
2. Let $r_2 = \{a, b\}.a^* + \{b, c\}.c^*$. Then $\{a, b\}$ and $\{b, c\}$ are initial.

Generalizing from the first example, we might be tempted to conjecture that if A is initial in r, then $(\forall a, b \in A)\ \partial_a(r) = \partial_b(r)$. However, the second example shows that this conjecture is wrong: $\{a, b\}$ is initial in r_2, but $\partial_a(r_2) = a^*$ and $\partial_b(r_2) = a^* + c^*$.

The problem with the second example is that $\{a, b\} \cap \{b, c\} \neq \emptyset$. Hence, instead of identifying initial literals of an ERE r, we define a set $\mathsf{next}(r)$ of next literals which are mutually disjoint, whose union contains $\mathsf{first}(r)$, and where the symbols in each literal yield the same derivative. In the second example, it must be that $\mathsf{next}(r_2) = \{\{a\}, \{b\}, \{c\}\}$.

It turns out that this problem arises in a number of cases when defining $\mathsf{next}(r)$ inductively. Hence, we define an operation $\bowtie$ that builds a set of mutually disjoint literals that cover the union of two sets of mutually disjoint literals.

▶ **Definition 7** (Join). Let $\mathfrak{L}_1$ and $\mathfrak{L}_2$ be two sets of mutually disjoint literals.

$$\mathfrak{L}_1 \bowtie \mathfrak{L}_2 := \{(A_1 \sqcap A_2), (A_1 \sqcap \overline{\bigsqcup \mathfrak{L}_2}), (\overline{\bigsqcup \mathfrak{L}_1} \sqcap A_2) \mid A_1 \in \mathfrak{L}_1, A_2 \in \mathfrak{L}_2\}$$

The following lemma states the properties of the join operation.

▶ **Lemma 8** (Properties of Join). *Let $\mathfrak{L}_1$ and $\mathfrak{L}_2$ be non-empty sets of mutually disjoint literals.*
1. $\bigcup(\mathfrak{L}_1 \bowtie \mathfrak{L}_2) = \bigcup \mathfrak{L}_1 \cup \bigcup \mathfrak{L}_2$.
2. $(\forall A \neq A' \in \mathfrak{L}_1 \bowtie \mathfrak{L}_2)\ A \sqcap A' = \emptyset$.
3. $(\forall A \in \mathfrak{L}_1 \bowtie \mathfrak{L}_2)\ (\forall A_i \in \mathfrak{L}_i)\ A \sqcap A_i \neq \emptyset \Rightarrow A \sqsubseteq A_i$.

$$
\begin{array}{lcl}
\mathsf{next}(\epsilon) & = & \{\emptyset\} \\
\mathsf{next}(A) & = & \{A\}
\end{array}
\qquad
\begin{array}{lcl}
\mathsf{next}(r{+}s) & = & \mathsf{next}(r) \bowtie \mathsf{next}(s) \\
\mathsf{next}(r{\cdot}s) & = & \begin{cases} \mathsf{next}(r) \bowtie \mathsf{next}(s), & \nu(r) \\ \mathsf{next}(r), & \neg\nu(r) \end{cases} \\
\mathsf{next}(r^*) & = & \mathsf{next}(r) \\
\mathsf{next}(r\&s) & = & \mathsf{next}(r) \sqcap \mathsf{next}(s) \\
\mathsf{next}(!r) & = & \mathsf{next}(r) \cup \{\bigsqcap\{\overline{A} \mid A \in \mathsf{next}(r)\}\}
\end{array}
$$

Figure 1 Computing next literals.

Proof of Lemma 8. Immediate from the definition. ◀

Figure 1 contains the definition of $\mathsf{next}(r)$. For ϵ the set of next literals consists of the empty set. The next literal of a literal A is A. The next literals of a union $r{+}s$ are computed as the join of the next literals of r and s as explained in Example 6. The next literals of a concatenation $r{\cdot}s$ are the next literals of r if r is not nullable. Otherwise, they are the join of the next literals of both operands. The next literals of a Kleene star expression r^* are the next literals of r. For an intersection $r\&s$, the set of next literals is the set of all intersections $A \sqcap A'$ of the next literals of both operands. In this case, the join operation $\bowtie$ is not needed because symbols that only appear in literals from one operand can be elided. To see this, consider $\mathsf{next}(a\&b) = \{\{a\} \sqcap \{b\}\} = \{\emptyset\}$ whereas $\{\{a\}\} \bowtie \{\{b\}\} = \{\emptyset, \{a\}, \{b\}\}$.

The set of next literals of $!r$ comprises the next literals of r and a new literal, which is the intersection of the complements of all literals in $\mathsf{next}(r)$. We might contemplate to exclude literals that contain symbols a such that $\partial_a(r)$ is equivalent to Σ^*, but we refrain from doing so because this equivalence cannot be decided with a finite set of rewrite rules [17].

The function $\mathsf{next}(r) \setminus \{\emptyset\}$ computes the equivalence classes of a partial equivalence relation $\sim$ on Σ such that equivalent symbols yield the same derivative on r. The relation is defined by $a \sim b$ if there exists $A \in \mathsf{next}(r)$ such that $a \in A$ and $b \in A$. Furthermore, the derivative by a symbol that is not part of the relation yields the empty set.

▶ **Lemma 9** (Partial Equivalence). *Let* $\mathfrak{L} = \mathit{next}(r)$.

1. $(\forall A \in \mathfrak{L})\ (\forall a, b \in A)\ \partial_a(r) = \partial_b(r)$
2. $(\forall a \notin \bigcup \mathfrak{L})\ \partial_a(r) = \emptyset$

Proof of Lemma 9. Both proofs are by induction on r. ◀

It remains to show that $\mathsf{next}(r)$ covers all symbols in $\mathsf{first}(r)$.

▶ **Lemma 10** (First). *For all* r, $\bigcup \mathit{next}(r) \supseteq \mathit{first}(r)$.

Proof of Lemma 10. The proof is by induction on r. ◀

Moreover, there are only finitely many different next literals for each regular expression.

▶ **Lemma 11** (Finiteness). *For all* r, $|\mathit{next}(r)|$ *is finite.*

Proof of Lemma 11. By induction on r. The base cases construct finite sets and the inductive cases build a finite number of combinations of the results from the subexpressions. ◀

Now, we put next literals to work. If we only take positive or negative derivatives with respect to next literals, then the inclusions in Lemma 3 turn into equalities. The result is that both the positive and the negative derivative, when applied to a next literal A, calculate a regular expression for the left quotient $A^{-1}[\![r]\!]$.

▶ **Theorem 12** (Left Quotient). *For all r, $A \in \mathit{next}(r) \setminus \{\emptyset\}$, and $a \in [\![A]\!]$:*

$$[\![\Delta_A(r)]\!] = [\![\nabla_A(r)]\!] = [\![\partial_a(r)]\!]$$

Proof of Lemma 12. By induction on r. ◀

Motivated by this result, we define the Brzozowski derivative for a non-empty subset A of a literal in $\mathsf{next}(r)$. This definition involves an arbitrary choice of $a \in A$, but this choice does not influence the calculated derivative according to Lemma 9, part 1.

▶ **Definition 13.** Let $A' \in \mathsf{next}(r)$. For each $\emptyset \neq A \subseteq A'$ define $\partial_A(r) := \partial_a(r)$, where $a \in A$.

▶ **Lemma 14** (Coverage). *For all a, u, and r it holds that:*

$$u \in [\![\partial_a(r)]\!] \;\Leftrightarrow\; \exists A \in \mathit{next}(r) : a \in A \wedge u \in [\![\Delta_A(r)]\!] \wedge u \in [\![\nabla_A(r)]\!]$$

Proof of Lemma 14. This result follows from Theorem 12 and Lemma 10. ◀

We conclude that to determine a finite set of representatives for all derivatives of a regular expression r it is sufficient to select one symbol a from each equivalence class $A \in \mathsf{next}(r) \setminus \{\emptyset\}$ and calculate $\partial_a(r)$. Alternatively, we may calculate $\Delta_A(r)$ or $\nabla_A(r)$ according to Theorem 12. It remains to lift this result to solving inequalities.

6 Solving Inequalities

Theorem 1 is the foundation of Antimirov's algorithm. It turns out that we can prove a stronger version of this theorem, which makes the rules CC-Disprove and CC-Unfold sound and complete and which also encompasses the soundness of the restriction to first sets.

▶ **Theorem 15** (Containment).

$$r \sqsubseteq s \;\Leftrightarrow (\nu(r) \Rightarrow \nu(s)) \;\wedge\; (\forall a \in \mathit{first}(r))\ \partial_a(r) \sqsubseteq \partial_a(s)$$

Proof of Theorem 15. $r \sqsubseteq s$ iff $[\![r]\!] \subseteq [\![s]\!]$ iff $(\forall w)\ w \in [\![r]\!] \Rightarrow w \in [\![s]\!]$.

Induction on w. If $w = \varepsilon$, then $\varepsilon \in [\![r]\!] \Rightarrow \varepsilon \in [\![s]\!]$ iff $\nu(r) \Rightarrow \nu(s)$. If $w = aw'$, then $a \in \mathsf{first}(r) \subseteq \mathsf{first}(s)$, $w' \in a^{-1}[\![r]\!] \subseteq a^{-1}[\![s]\!]$, which is equivalent to $\partial_a(r) \sqsubseteq \partial_a(s)$. ◀

As we remarked before, it may be very expensive (or even impossible) to construct all derivatives with respect to the first symbols, particularly for negated expressions and for large or infinite alphabets. To obtain a decision procedure for containment, we need a finite set of derivatives. Therefore, we use next literals as representatives of the first symbols and use Brzozowski derivatives *on literals* (Definition 13) on both sides.

To define the next literals of an inequality $r \dot{\sqsubseteq} s$, it would be sound to use the join of the next literals of both sides: $\mathsf{next}(r) \bowtie \mathsf{next}(s)$. However, we can do slightly better. Theorem 15 proves that the first symbols of r are sufficient to prove containment. Using the full join operation, however, would cover $\mathsf{first}(r) \cup \mathsf{first}(s)$ (by Lemma 10). Hence, we define a left-biased version of the join operator that only covers the symbols of its left operand.

▶ **Definition 16** (Left Join). Let $\mathfrak{L}_1$ and $\mathfrak{L}_2$ be two sets of mutually disjoint literals.

$$\mathfrak{L}_1 \ltimes \mathfrak{L}_2 \ := \{(A_1 \sqcap A_2), (A_1 \sqcap \overline{\bigsqcup \mathfrak{L}_2}) \mid A_1 \in \mathfrak{L}_1, A_2 \in \mathfrak{L}_2\}$$

The following lemma states the properties of the left join operation.

▶ **Lemma 17** (Properties of Left Join). *Let $\mathfrak{L}_1$ and $\mathfrak{L}_2$ be non-empty sets of mutually disjoint literals.*

1. $\bigcup(\mathfrak{L}_1 \ltimes \mathfrak{L}_2) = \bigcup \mathfrak{L}_1$.
2. $(\forall A \neq A' \in \mathfrak{L}_1 \ltimes \mathfrak{L}_2)\ A \sqcap A' = \emptyset$.
3. $(\forall A \in \mathfrak{L}_1 \ltimes \mathfrak{L}_2)\ (\forall A_i \in \mathfrak{L}_i)\ A \sqcap A_i \neq \emptyset \Rightarrow A \sqsubseteq A_i$.

Proof of Lemma 17. Immediate from the definition. ◀

▶ **Definition 18** (Next Literals of an Inequality). Let $r \mathrel{\dot{\sqsubseteq}} s$ be an inequality.

$$\mathsf{next}(r \mathrel{\dot{\sqsubseteq}} s) := \mathsf{next}(r) \ltimes \mathsf{next}(s)$$

Finally, we can state a generalization of Antimirov's containment theorem for EREs, where each unfolding step generates only finitely many derivatives.

▶ **Theorem 19** (Containment). *For all regular expressions r and s,*

$$r \sqsubseteq s \ \Leftrightarrow (\nu(r) \Rightarrow \nu(s)) \ \wedge \ (\forall A \in \mathsf{next}(r \mathrel{\dot{\sqsubseteq}} s))\ \partial_A(r) \sqsubseteq \partial_A(s).$$

Proof of Theorem 19. The proof is by contraposition. If $r \not\sqsubseteq s$ then $\exists A \in \mathsf{next}(r \mathrel{\dot{\sqsubseteq}} s)$: $\partial_A(r) \not\sqsubseteq \partial_A(s)$ or $\neg(\nu(r) \Rightarrow \nu(s))$. ◀

For $A \in \mathsf{next}(r \mathrel{\dot{\sqsubseteq}} s)$ define $\nabla_A(r \mathrel{\dot{\sqsubseteq}} s) := (\nabla_A(r) \mathrel{\dot{\sqsubseteq}} \Delta_A(s)) = (\partial_A(r) \mathrel{\dot{\sqsubseteq}} \partial_A(s))$.

▶ **Theorem 20** (Finiteness). *Let R be a finite set of regular inequalities. Define*

$$F(R) = R \cup \{\nabla_A(r \mathrel{\dot{\sqsubseteq}} s) \mid r \mathrel{\dot{\sqsubseteq}} s \in R, A \in \mathsf{next}(r \mathrel{\dot{\sqsubseteq}} s)\}$$

For each r and s, the set $\bigcup_{i \in \mathbb{N}} F^{(i)}(\{r \sqsubseteq s\})$ is finite.

Proof of Theorem 20. As we consider regular expressions up to similarity (cf. [4]) and $\nabla_A(r \mathrel{\dot{\sqsubseteq}} s) = \partial_A(r) \mathrel{\dot{\sqsubseteq}} \partial_A(s)$ is essentially applying the Brzozowski derivative to a pair of (extended) regular expressions, the set of these pairs is finite (because there are only finitely many dissimilar iterated Brzozowski derivatives for a regular expression [4]). ◀

These results are the basis for a complete decision procedure for solving inequalities on extended regular expressions where literals are defined via an effective boolean algebra. Figure 2 defines this procedure as a judgment of the form $\Gamma \vdash r \mathrel{\dot{\sqsubseteq}} s \ : \ b$, where Γ is a set of previous visited inequalities $r \mathrel{\dot{\sqsubseteq}} s$ with $\nu(r) \Rightarrow \nu(s)$ that are assumed to be true and $b \in \{\mathit{true}, \mathit{false}\}$. The effective boolean algebra comes into play in the computation of the next literals and in the computation of the derivatives.

Rule (Disprove) detects contradictory inequalities in the same way as Antimirov's rule CC-Disprove. Rule (Cycle) detects circular reasoning: Under the assumption that $r \mathrel{\dot{\sqsubseteq}} s$ holds we were not (yet) able to derive a contradiction and thus conclude that $r \mathrel{\dot{\sqsubseteq}} s$ holds. This rule guarantees termination because of the finiteness result (Theorem 20). The rules (Unfold-True) and (Unfold-False) apply only if $r \mathrel{\dot{\sqsubseteq}} s$ is neither contradictory nor in the context. A deterministic implementation would generate the literals $A \in \mathsf{next}(r \mathrel{\dot{\sqsubseteq}} s)$ and recursively check $\nabla_A(r \mathrel{\dot{\sqsubseteq}} s)$. If any of these checks returns false, then (Unfold-False) fires. Otherwise (Unfold-True) signals a successful containment proof. Theorem 19 is the basis for soundness and completeness of the unfolding rules.

$$\frac{\nu(r) \qquad \neg\nu(s)}{\Gamma \vdash r \mathrel{\dot{\sqsubseteq}} s \ : \ \mathit{false}}\ (\textsc{Disprove}) \qquad \frac{r \mathrel{\dot{\sqsubseteq}} s \in \Gamma}{\Gamma \vdash r \mathrel{\dot{\sqsubseteq}} s \ : \ \mathit{true}}\ (\textsc{Cycle})$$

$$\frac{r \mathrel{\dot{\sqsubseteq}} s \notin \Gamma \qquad \nu(r) \Rightarrow \nu(s) \qquad \forall A \in \mathsf{next}(r \mathrel{\dot{\sqsubseteq}} s) : \ \Gamma \cup \{r \mathrel{\dot{\sqsubseteq}} s\} \ \vdash \ \partial_A(r) \mathrel{\dot{\sqsubseteq}} \partial_A(s) \ : \ \mathit{true}}{\Gamma \ \vdash \ r \mathrel{\dot{\sqsubseteq}} s \ : \ \mathit{true}}\ (\textsc{Unfold-True})$$

$$\frac{r \mathrel{\dot{\sqsubseteq}} s \notin \Gamma \qquad \nu(r) \Rightarrow \nu(s) \qquad \exists A \in \mathsf{next}(r \mathrel{\dot{\sqsubseteq}} s) : \ \Gamma \cup \{r \mathrel{\dot{\sqsubseteq}} s\} \ \vdash \ \partial_A(r) \mathrel{\dot{\sqsubseteq}} \partial_A(s) \ : \ \mathit{false}}{\Gamma \ \vdash \ r \mathrel{\dot{\sqsubseteq}} s \ : \ \mathit{false}}\ (\textsc{Unfold-False})$$

Figure 2 Decision procedure for containment.

$$\Gamma \vdash r \sqsubseteq r \ : \ \mathit{true}\ (\textsc{Prove-Identity}) \qquad \Gamma \vdash \emptyset \sqsubseteq s \ : \ \mathit{true}\ (\textsc{Prove-Empty})$$

$$\frac{\nu(s)}{\Gamma \vdash \epsilon \sqsubseteq s \ : \ \mathit{true}}\ (\textsc{Prove-Nullable}) \qquad \frac{\exists A \in \mathsf{next}(r) : \ A \neq \emptyset}{\Gamma \vdash r \sqsubseteq \emptyset \ : \ \mathit{false}}\ (\textsc{Disprove-Empty})$$

Figure 3 Prove and disprove axioms.

▶ **Theorem 21** (Soundness). *For all regular expression r and s:*

$$\emptyset \vdash r \mathrel{\dot{\sqsubseteq}} s \ : \ \top \ \Leftrightarrow \ r \sqsubseteq s$$

Proof of Theorem 21. We prove that $\Gamma \vdash r \mathrel{\dot{\sqsubseteq}} s \ : \ \mathit{false}$ iff $r \not\sqsubseteq s$, for all contexts Γ where $r \mathrel{\dot{\sqsubseteq}} s \notin \Gamma$. This is sufficient because each regular inequality gives rise to a finite derivation by Theorem 20. ◀

In addition to the rules from Figure 2, we may add auxiliary rules to detect trivially consistent or inconsistent inequalities early (Figure 3 contains some examples). Such rules may be used to improve efficiency. They decide containment directly instead of unfolding repeatedly.

7 Conclusion

Antimirov's algorithm is a viable tool for proving containment of regular expressions to extended regular expressions on potentially infinite alphabets. To work effectively with such alphabets, we require that literals in regular expressions are drawn from an effective boolean algebra. As a slight difference, we work with Brzozowski derivatives instead of Antimirov's notion of partial derivative.

The main effort in lifting Antimirov's algorithm is to identify, for each regular inequality $r \mathrel{\dot{\sqsubseteq}} s$, a finite set of symbols such that calculating the derivation with respect to these symbols covers all possible derivations with all symbols. We regard the construction of the set of suitable representatives in an effective boolean algebra, embodied in the notion of next literals $\mathsf{next}(r \mathrel{\dot{\sqsubseteq}} s)$, as a key contribution of this work.

References

1 Valentin M. Antimirov. Rewriting regular inequalities. In Horst Reichel, editor, *FCT*, volume 965 of *LNCS*, pages 116–125. Springer, 1995.

2 Valentin M. Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theoretical Computer Science*, 155(2):291–319, 1996.

3 Filippo Bonchi and Damien Pous. Checking NFA equivalence with bisimulations up to congruence. In Roberto Giacobazzi and Radhia Cousot, editors, *POPL*, pages 457–468, Rome, Italy, January 2013. ACM.

4 Janusz A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11(4):481–494, 1964.

5 Pascal Caron, Jean-Marc Champarnaud, and Ludovic Mignot. Partial derivatives of an extended regular expression. In Adrian Horia Dediu, Shunsuke Inenaga, and Carlos Martín-Vide, editors, *LATA*, volume 6638 of *LNCS*, pages 179–191. Springer, 2011.

6 A. Ginzburg. A procedure for checking equality of regular expressions. *J. ACM*, 14(2):355–362, April 1967.

7 Victor M. Glushkov. The abstract theory of automata. *Russian Mathematical Surveys*, 16(5):1–53, 1961.

8 Phillip Heidegger, Annette Bieniusa, and Peter Thiemann. Access permission contracts for scripting languages. In John Field and Michael Hicks, editors, *Proc. 39th ACM Symp. POPL*, pages 111–122, Philadelphia, USA, January 2012. ACM Press.

9 John Edward Hopcroft and Richard Manning Karp. A linear algorithm for testing equivalence of finite automata. Technical report, Cornell University, 1971.

10 Harry B. Hunt III, Daniel J. Rosenkrantz, and Thomas G. Szymanski. On the equivalence, containment, and covering problems for the regular and context-free languages. *J. Comput. Syst. Sci.*, 12(2):222–268, 1976.

11 Tao Jiang and Bala Ravikumar. Minimal NFA problems are hard. *SIAM J. Comput.*, 22(6):1117–1141, 1993.

12 Matthias Keil and Peter Thiemann. Efficient dynamic access analysis using JavaScript proxies. In *Proceedings of the 9th Symposium on Dynamic Languages*, DLS '13, pages 49–60, New York, NY, USA, 2013. ACM.

13 Matthias Keil and Peter Thiemann. Symbolic solving of regular expression inequalities. Technical report, Institute for Computer Science, University of Freiburg, 2014.

14 Robert McNaughton and Hisao Yamada. Regular expressions and state graphs for automata. *Electronic Computers, IRE Transactions on*, EC-9(1):39–47, 1960.

15 Albert R. Meyer and Larry J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *SWAT (FOCS)*, pages 125–129. IEEE Computer Society, 1972.

16 Scott Owens, John H. Reppy, and Aaron Turon. Regular-expression derivatives reexamined. *J. Funct. Program.*, 19(2):173–190, 2009.

17 Valentin N. Redko. On defining relations for the algebra of regular events. *Ukrain. Mat.*, 16:120–126, 1964.

18 Ken Thompson. Regular expression search algorithm. *Commun. ACM*, 11(6):419–422, 1968.

19 Gertjan van Noord and Dale Gerdemann. Finite state transducers with predicates and identities. *Grammars*, 4(3):263–286, 2001.

20 Margus Veanes. Applications of symbolic finite automata. In Stavros Konstantinidis, editor, *CIAA*, volume 7982 of *Lecture Notes in Computer Science*, pages 16–23, Halifax, NS, Canada, 2013. Springer.

21 Bruce W. Watson. Implementing and using finite automata toolkits. *Nat. Lang. Eng.*, 2(4):295–302, December 1996.