Testing the equivalence of regular expressions

Marco Almeida Nelma Moreira Rogério Reis

Technical Report Series: DCC-2007-07
Version 1.1 January 2008

Departamento de Ciência de Computadores
&
Laboratório de Inteligência Artificial e Ciência de Computadores

Faculdade de Ciências da Universidade do Porto
Rua do Campo Alegre, 1021/1055,
4169-007 PORTO,
PORTUGAL

Tel: 220 402 900 Fax: 220 402 950
`http://www.dcc.fc.up.pt/Pubs/`

Testing the equivalence of regular expressions

Marco Almeida Nelma Moreira Rogério Reis
{mfa,nam,rvr}@ncc.up.pt
DCC-FC & LIACC, Universidade do Porto
R. do Campo Alegre 1021/1055, 4169-007 Porto, Portugal

October 2007

Abstract

Antimirov and Mosses presented a rewrite system for deciding the equivalence of two (extended) regular expressions and argued that this method could lead to a better average-case algorithm than those based on the comparison of the equivalent minimal DFAs. In this paper we present a functional approach of a variant of that method, prove its correctness and give some experimental comparative results. Although being a refutation method, our preliminary results lead to the conclusion that indeed this method is feasible and it is, almost always, faster than the classical methods.

Keywords regular languages, regular expressions, derivatives, regular expression comparison, minimal automata

1 Introduction

Regular expressions have many applications as a good notation for regular languages. But for its manipulation, however, finite automata are normally used, because the methods for dealing with automata are usually considered much more efficient. Examples of this attitude are the decision if a word belongs to the language represented by a regular expression or if two regular expressions represent the same language (*i.e.* if they are equivalent). The problem of deciding whether two regular expressions are equivalent is PSPACE-complete [SM73]. This problem is normally solved by transforming each regular expression to equivalent nondeterministic finite automata; convert those automata into equivalent deterministic ones, and finally minimise both deterministic finite automata, and determine if the resulting automata are isomorphic.

Antimirov and Mosses [AM94] presented a rewrite system for deciding the equivalence of two extended regular expressions (*i.e.* with intersection) based on a new complete axiomatization of the extended algebra of regular sets. This axiomatization, or other classical complete axiomatizations of the algebra of regular sets, can be used to construct an algorithm for deciding the equivalence of two regular expressions, but normally deduction systems are quite inefficient. That rewrite system is a refutation method that normalises regular expressions in such way that testing their equivalence corresponds to an iterated process of testing the equivalence of their derivatives. Termination is ensured because the considered set of derivatives is finite and possible cycles are detected using *memoisation.* Antimirov and Mosses suggested that their method could lead to a better average-case algorithm than those based on the comparison of the equivalent minimal deterministic finite automata. In this paper we present a functional approach of a variant of that method, prove its correctness and give some experimental comparative results. Although being a refutation method, our preliminary results lead to the conclusion that indeed this method is feasible and it is, almost always, faster than the other classical methods.

2 Preliminaries

We recall here some definitions and facts concerning regular languages, regular expressions and finite automata. For further details we refer the reader to the works of Hopcroft *et.al* [HMU00], Kozen [Koz97] and Kuich and Salomaa [KS86].

2.1 Regular languages

Let Σ be a nonempty finite (or countably infinite) set. We call this set an *alphabet* and say that its elements are *letters* or *symbols*. A *word* over an alphabet Σ is a finite sequence of zero or more symbols of Σ. The string with zero symbols is called *empty word* and we denote it by ϵ. Thus,

$$\epsilon, 0, 011, 0000, 1101,$$

are words of the alphabet $\Sigma = \{0, 1\}$. The set of all words over an alphabet Σ is denoted by $\Sigma^\star$.

If w_0 and w_1 are words over an alphabet Σ, then their *concatenation* $w_0 \cdot w_1$ is also a word over Σ. We define the concatenation by the creation of a new word w, which is formed by appending all the symbols of w_1 to w_0. It is clear that the concatenation is an associative operation and that ϵ is an identity with respect to this operation: $\epsilon \cdot w = w \cdot \epsilon = w$. We usually omit $\cdot$, the concatenation operator. Given a word w and any natural number i, w^i means the word obtained by concatenating i copies of w. Naturally, w^0 denotes the empty word ϵ.

By *length* of a word w we mean the number of symbols in w when each symbol is counted as many times as it occurs. We denote the length of a word w by $|w|$, and $|\epsilon| = 0$.

Any subset of $\Sigma^\star$ is called a *language*. Because languages are sets, we may consider Boolean operations. The *sum* or *union* of two languages L_1 and L_2 is denoted by $L_1 + L_2$ and their *intersection* by $L_1 \cap L_2$. The complement of a language L, with respect to the set $\Sigma^\star$, is denoted by $\overline{L}$, and we can combine these operations in order to have the *difference* of two languages as follows:

$$L_1 - L_2 = L_1 \cap \overline{L_2}.$$

All of these operations have the expected set theoretical meanings:

$$\begin{aligned} L_1 + L_2 &= \{w \mid w \in L_1 \cup L_2\}; \\ L_1 \cap L_2 &= \{w \mid w \in L_1 \cap L_2\}; \\ \overline{L} &= \Sigma^\star - L. \end{aligned}$$

Just like with words, we can consider the concatenation of two languages L_1 and L_2. It is denoted $L_1 \cdot L_2$ (although, again, sometimes we omit the operator $\cdot$) and defined to be the language

$$L_1 \cdot L_2 = \{w_1 w_2 \mid w_1 \in L_1, w_2 \in L_2\}.$$

Also, L^i, for $i \geq 0$, is meaningful for a language L. We define L^0 to be the language of the empty word, $L_\epsilon = \{\epsilon\}$, and L^i to be $i \geq 1$ concatenations of L. Being a set, we also accept a language to be *empty*, i.e., a language with no words, denoted $L_\emptyset$. Having no words, $L_\emptyset = \emptyset$. Note that the languages $L_\emptyset$ and L_ϵ are the zero and the identity elements with respect to the concatenation, respectively.

The *concatenation closure* (or *iteration*) of a language L, written $L^\star$, is the sum of all the powers of L:

$$L^\star = \sum_{i=0}^{\infty} L^i.$$

The star operator $\star$ is also called *Kleene star*. In the same way, the concatenation closure is sometimes also referred to as the *Kleene closure*.

Some subsets of $\Sigma^\star$ are called *regular languages*. These languages correspond to sets of cardinality less or equal to one and closed under union, concatenation, and Kleene star operations, which can be defined with *regular expressions*.

Definition 1 (Regular expression) *A regular expression over an alphabet Σ is inductively defined by:*

- *$\emptyset$ and ϵ are regular expressions;*
- *each $a \in \Sigma$ is a regular expression;*
- *if α and β are regular expressions, then so it is the* disjunction *$(\alpha + \beta)$;*
- *if α and β are regular expressions, then so it is the* concatenation *$(\alpha \cdot \beta)$;*
- *if α is a regular expression, then so it is the* Kleene closure *$(\alpha^\star)$.*

Definition 2 (Language of a regular expression) *The language defined by a regular expression α, denoted $L(\alpha)$, is defined in the following way:*

- *$L(\emptyset) = \emptyset$ and $L(\epsilon) = \{\epsilon\}$;*
- *$L(a) = \{a\}$, for $a \in \Sigma$;*
- *$L((\alpha + \beta)) = L(\alpha) \cup L(\beta)$;*
- *$L((\alpha \cdot \beta)) = L(\alpha) \cdot L(\beta)$;*
- *$L((\alpha^\star)) = L(\alpha)^\star$.*

We say that two regular expressions α and β are equivalent*, and write $\alpha \sim \beta$, if they represent the same language, i.e., $L(\alpha) = L(\beta)$.*

2.1.1 Monoids, semirings, and Kleene algebras

A *monoid* consists of a set M, an associative binary operation $\bullet$ on M, and a neutral element e such that $e \bullet a = a \bullet e = a$ for every $a \in M$. Usually, the binary operation — sometimes called *product* — is omitted and we simply write $ea = ae = a$. A *monoid* is called *commutative* if and only if $a \bullet b = b \bullet a$ for all $a, b \in M$. The *free monoid* $S^\star$, generated by a nonempty countable set S contains all the finite words

$$a_1 \cdots a_n, \quad a_i \in S$$

as its elements. We call *semiring* to a structure $\langle S, \oplus, \bullet, 0, 1\rangle$, where S is a set and $\oplus$, $\bullet$ are two binary operations, called addition and multiplication, such that:

$\langle S, \oplus, 0\rangle$	is a commutative monoid,
$\langle S, \bullet, 1\rangle$	is a monoid,
$a \bullet (b \oplus c) = a \bullet b \oplus a \bullet c$	the multiplication distributes over addition,
$(a \oplus b) \bullet c = a \bullet c \oplus b \bullet c$	
$0 \bullet a = a \bullet 0 = 0$	0 annihilates the multiplication.

If we take a semiring and add a unary operation $*$ which complies with certain axioms, we get what is called a *Kleene algebra*.

2.1.2 Axioms system

We will now introduce an *axiom system* for regular expressions:

$$\begin{aligned}
\alpha + \emptyset &\sim \alpha, && (A_1)\\
\alpha + \alpha &\sim \alpha, && (A_2)\\
\alpha + \beta &\sim \beta + \alpha, && (A_3)\\
\alpha + (\beta + \gamma) &\sim (\alpha + \beta) + \gamma, && (A_4)\\
\epsilon \cdot \alpha &\sim \alpha, && (A_5)\\
\emptyset \cdot \alpha &\sim \emptyset, && (A_6)\\
\alpha(\beta\gamma) &\sim (\alpha\beta)\gamma, && (A_7)\\
\alpha(\beta + \gamma) &\sim \alpha\beta + \alpha\gamma, && (A_8)\\
(\alpha + \beta)\gamma &\sim \alpha\gamma + \beta\gamma, && (A_9)\\
\alpha^\star &\sim \epsilon + \alpha\alpha^\star, && (A_{10})\\
\alpha^\star &\sim (\epsilon + \alpha)^\star. && (A_{11})
\end{aligned}$$

This is essentially Salomaa's [Sal66] axiom system F_1, which includes the following two rules of inference:

- *Substitution*

$$\frac{\gamma' \sim \gamma[\alpha/\beta], \quad \alpha \sim \beta, \quad \gamma \sim \delta}{\gamma' \sim \delta, \quad \gamma' \sim \gamma} \qquad (R_1)$$

- *Solution of equations*

$$\frac{\alpha \sim \alpha\beta + \gamma, \quad \epsilon \notin L(\beta)}{\alpha \sim \beta^\star\gamma} \qquad (R_2)$$

We use ϵ instead of $\emptyset^\star$, but $\emptyset^\star \sim \epsilon$ may actually be derived with the following sequence of equations:

$$\begin{aligned}
\emptyset^\star &\sim \epsilon + \emptyset^\star \cdot \emptyset && \text{by Axiom } A_{10}; && (1)\\
&\sim \epsilon + \emptyset && \text{by Equation 3;} && (2)\\
&\sim \epsilon && \text{by Axiom } A_1.
\end{aligned}$$

The step from 1 to 2 is possible because $\emptyset \cdot \alpha \sim \alpha \cdot \emptyset \sim \emptyset$, as can be shown from Axiom A_6 and the following equalities:

$$\begin{aligned} \alpha \cdot \emptyset &\sim \alpha \cdot \emptyset + \emptyset && \text{by Axiom } A_1; \\ &\sim \alpha \cdot (\emptyset \cdot \emptyset) + \emptyset && \text{by Axiom } A_6; \\ &\sim (\alpha \cdot \emptyset) \cdot \emptyset + \emptyset && \text{by Axiom } A_7; \\ &\sim \emptyset \cdot \emptyset^\star && \text{by the inference rule } R_2; \\ &\sim \emptyset && \text{by Axiom } A_6. \end{aligned} \tag{3}$$

The disjunction of regular expressions is associative, commutative and idempotent (Axioms A_4, A_3 and A_2, respectively). The concatenation is associative (Axiom A_7), and the star is idempotent (Lemma 1). We call these *ACI properties* and, when referring to an arbitrary regular expression, unless stated otherwise, we will always be considering the regular operations modulo these properties. This means, for example, that we do not distinguish the following regular expressions:

$$\begin{gathered} (\alpha + (\beta^\star)^\star) + \gamma(\alpha'\beta') + \alpha, \\ \alpha + (\beta^\star + (\gamma\alpha')\beta' + \alpha), \\ \alpha + \beta^\star + \gamma\alpha'\beta'. \end{gathered}$$

In order to resolve ambiguity in less clear situations, we assign precedence to the operators. For example,

$$\alpha + \beta\gamma$$

could be interpreted as either

$$\alpha + (\beta\gamma) \quad \text{or} \quad (\alpha + \beta)\gamma,$$

which are not equivalent. We adopt the convention that the concatenation operator $\cdot$ has higher precedence than the disjunction operator $+$. In the same way, we assign the star operator $\star$ higher precedence than $\cdot$ or $+$.

Lemma 1 *The Kleene star is idempotent, i.e., $\alpha^\star \sim (\alpha^\star)^\star$.*

Proof *By definition, $L((\alpha^\star)^\star) = (L(\alpha)^\star)^\star$. The language $(L(\alpha)^\star)^\star$ is the set of all words created by concatenating words of the language $L(\alpha)^\star$. But the words of $L(\alpha)^\star$ are themselves composed of words from $L(\alpha)$. Thus, every word in $(L(\alpha)^\star)^\star$ is also a concatenation of words from $L(\alpha)$ and is therefore in the language of $L(\alpha)^\star$.*

Let us now consider the set RE of all the regular expressions, and the regular operations $\cdot$, $+$, and $\star$. Clearly, both $\langle RE, \cdot, \epsilon\rangle$ and $\langle RE, +, \emptyset\rangle$ are monoids:

$(\alpha\beta)\gamma = \alpha(\beta\gamma)$	$(\alpha + \beta) + \gamma = \alpha + (\beta + \gamma)$	the operations are associative,
$\alpha\beta \in RE$	$\alpha + \beta \in RE$	the set is closed under the operations,
$\epsilon\alpha = \alpha\epsilon = \alpha$	$\emptyset + \alpha = \alpha + \emptyset = \alpha$	has an identity,

for $\alpha, \beta, \gamma \in RE$. These two monoids of regular expressions form the semiring $\langle RE, +, \cdot, \emptyset, \epsilon\rangle$. The regular expressions set RE, with the regular operations $\cdot$, $+$, $\star$, the Axioms A_1–A_{11}, and the inference rules R_1 and R_2, form a Kleene algebra.

2.2 Definitions and notation

We will now introduce some definitions and notation not usually found in the common bibliography.

Definition 3 (Size of a regular expression) *Let the* ordinary length, *denoted by $|\alpha|$, be the measure for the* size *of a regular expression α. This measure counts the total number of symbols in the regular expression, including parentheses and operators. The regular expression $(a+b)^\star(a+\epsilon)$, for example, has size 11.*

There are other possible measures, but, because we will only be considering irreducible regular expressions (as defined bellow), there will be no syntactic redundancy and this one is enough.

Definition 4 (Uncollapsible regular expression) *We say that a regular expression α is* uncollapsible *if <u>none</u> of the following conditions hold:*

- *α contains the proper sub-expression $\emptyset$, and $|\alpha| > 1$;*
- *α contains a sub-expression of the form $\beta\gamma$ or $\gamma\beta$ where $L(\beta) = \{\epsilon\}$;*
- *α contains a sub-expression of the form $\beta + \gamma$ where $L(\beta) = \{\epsilon\}$ and $\epsilon \in L(\gamma)$.*

Definition 5 (Irreducible regular expression) *We say α is an* irreducible *regular expression if α is uncollapsible and <u>both</u> of the following conditions are true:*

- *α does not contain superfluous parentheses;*
- *α does not contain a sub-expression of the form ${\beta^\star}^\star$.*

Whenever using the term regular expression *we will be referring to irreducible regular expressions.*

Definition 6 (Languages containing the empty word) *A regular expression α possesses the* empty word property *(e.w.p.) [Sal69] if and only if one of the following conditions holds:*

- *$\alpha = \epsilon$;*
- *$\alpha = \beta^\star$ (where β is an arbitrary r.e.);*
- *α is a disjunction of regular expressions, one of which possesses the e.w.p.;*
- *α is a concatenation of regular expressions, all of which possess the e.w.p..*

This is the same as saying that α possesses the e.w.p. if and only if $\epsilon \in L(\alpha)$.

Definition 7 (Constant part of a regular expression) *We say that the* constant part *of a regular expression α is ϵ if α has the e.w.p. property, and $\emptyset$ otherwise.*

Definition 8 (Linear regular expression) *A regular expression α is* linear *if it is generated by the following context free grammar, where A is the initial symbol:*

$$\begin{aligned} &A \rightarrow C \mid C \cdot B \mid A + A \\ &B \rightarrow C \mid B + B \mid B \cdot B \mid B^\star \\ &C \rightarrow a \in \Sigma \end{aligned} \tag{G_1}$$

This means that α has the form $a_1\alpha_1 + \cdots + a_n\alpha_n$ for $a_i \in \Sigma$.

We say that an expression $a \cdot \alpha$ has head *a and* tail *α. We use* $\mathrm{head}(\alpha)$ *and* $\mathrm{tail}(\alpha)$ *to denote, respectively, the multiset of all heads and the multiset of all tails in α. The set of all the linear regular expressions is denoted by RE_{lin}.*

Definition 9 (Deterministic linear regular expression) *A linear regular expression α is* deterministic *if each element in* $\mathrm{head}(\alpha)$ *occurs exactly once. We denote the set of all deterministic linear regular expressions by RE_{det}.*

Definition 10 (Pre-linear regular expression) *We call* pre-linear *to a regular expression generated by following context free grammar with initial symbol A':*

$$\begin{aligned} A' &\to \emptyset \mid D \\ D &\to A \mid D \cdot B \mid D + D \\ A &\to C \mid C \cdot B \mid A + A \\ B &\to C \mid B + B \mid B \cdot B \mid B^\star \\ C &\to a \in \Sigma \end{aligned} \tag{G_2}$$

Simply stated, a pre-linear regular expression is either $\emptyset$, an already linear r.e. or a disjunction of concatenations where the first argument of each concatenation is a linear regular expression. Just like with linear regular expressions, we say that $a \cdot \alpha$ has head a and tail α. The set of all the linear regular expressions is denoted by RE_{plin}.

Definition 11 (Similar regular expressions) *Two regular expressions are* similar *[Brz64] if one can be transformed to the other using only the Axioms A_2, A_3, and A_4. When not similar, the regular expressions are called dissimilar.*

Definition 12 (Derivative of a regular expression — with respect to a symbol) *The* derivative *[Brz64] of a regular expression α with respect to a* symbol *$a \in \Sigma$, denoted $a^{-1}(\alpha)$, is recursively defined on the structure of α as follows:*

$$\begin{aligned} a^{-1}(\emptyset) &= \emptyset; \\ a^{-1}(\epsilon) &= \emptyset; \\ a^{-1}(\alpha) &= \begin{cases} \epsilon & \text{if } \alpha = a; \\ \emptyset & \text{if } \alpha \neq a\ ; \end{cases} \\ a^{-1}(\alpha + \beta) &= a^{-1}(\alpha) + a^{-1}(\beta); \\ a^{-1}(\alpha\beta) &= a^{-1}(\alpha)\beta + \mathrm{const}(\alpha)a^{-1}(\beta); \\ a^{-1}(\alpha^\star) &= a^{-1}(\alpha)\alpha^\star. \end{aligned}$$

where α, β are arbitrary, not necessarily irreducible, regular expressions.

In the particular case of a deterministic linear regular expression, the calculation of the derivative with respect to a symbol a can be simplified as follows:

$$a^{-1}(\alpha) = \begin{cases} \beta & \text{if } a \cdot \beta \text{ is a sub-expression of } \alpha; \\ \epsilon & \text{if } \alpha = a; \\ \emptyset & \text{otherwise.} \end{cases}$$

Definition 13 (Derivative of a regular expression — with respect to a word) *The* derivative *[Brz64] of a regular expression α (not necessarily irreducible) with respect to the* word $w \in \Sigma^\star$, *denoted $w^{-1}(\alpha)$, is found recursively in the structure of w:*

$$\epsilon^{-1}(\alpha) = \alpha;$$
$$w^{-1}(\alpha) = (u \cdot a)^{-1}(\alpha) = a^{-1}(u^{-1}(\alpha)).$$

3 Regular expressions equivalency

The classical approach to the problem of comparing two regular expressions α and β, i.e., deciding if $L(\alpha) = L(\beta)$, typically consists on:

1. obtain non-deterministic finite automata, N_α and N_β, which accept the same language as α and β, respectively;

2. convert the non-deterministic automata to equivalent deterministic ones, $D_\alpha \equiv N_\alpha$ and $D_\beta \equiv N_\beta$;

3. minimise both D_α and D_β.

Because, for a given regular language, the minimal automaton is unique up to isomorphism, D_α and D_β can be compared using a canonical representation [RMA05], and thus check if $L(\alpha) = L(\beta)$.

In this section, we present a method for verifying the equivalence of two regular expressions that does not require the minimisation process. This method is a variant of the rewrite system presented by Antimirov and Mosses[AM94], which provides an algebraic calculus for testing the equivalence of two regular expressions avoiding the construction of the canonical minimal automata. The main difference of our method from Antimirov and Mosses's rewrite system is that our method is based on a functional approach and we consider the regular expressions to be irreducible and thus take all operations modulo ACI. We do not consider extended regular expressions (with intersection), as was the case in Antimirov and Mosses's rewrite system.

3.1 Auxiliary functions

We start by exposing some auxiliary functions that will be used in the definition of the main function, `equiv`, which performs the actual comparison of the languages defined by two regular expressions.

We will define and prove the correctness of the functions that compute both the constant and the deterministic linear part of any regular expression. We will also show that every regular expression α can be written in the form

$$\text{const}(\alpha) + \text{det}(\text{lin}(\alpha))$$

where `const(`α`)` and `det(lin(`α`))` denote the constant and the deterministic linear part of α, respectively.

Let $a \in \Sigma$, and α, β, γ be arbitrary regular expressions. We define the functions `const` and `lin` as follows:

$$\mathrm{const} : RE \to \{\emptyset, \epsilon\} \qquad\qquad \mathrm{lin} : RE \to RE_{lin} \cup \{\emptyset\}$$

$$\mathrm{const}(\alpha) = \begin{cases} \epsilon & \text{if } \epsilon \in L(\alpha); \\ \emptyset & \text{otherwise.} \end{cases} \qquad\qquad \mathrm{lin} = \mathrm{lin}_2 \circ \mathrm{lin}_1 \,.$$

Where

$$\begin{aligned}
\mathrm{lin}_1 &: RE \to RE_{plin} \\
\mathrm{lin}_1(\emptyset) &= \emptyset; \\
\mathrm{lin}_1(\epsilon) &= \emptyset; \\
\mathrm{lin}_1(a) &= a; \\
\mathrm{lin}_1(\alpha + \beta) &= \mathrm{lin}_1(\alpha) + \mathrm{lin}_1(\beta); \\
\mathrm{lin}_1(\alpha^\star) &= \mathrm{lin}_1(\alpha) \cdot \alpha^\star; \\
\mathrm{lin}_1(a \cdot \alpha) &= a \cdot \alpha; \\
\mathrm{lin}_1((\alpha + \beta) \cdot \gamma) &= \mathrm{lin}_1(\alpha \cdot \gamma) + \mathrm{lin}_1(\beta \cdot \gamma); \\
\mathrm{lin}_1(\alpha^\star \cdot \beta) &= \mathrm{lin}_1(\alpha) \cdot \alpha^\star \cdot \beta + \mathrm{lin}_1(\beta).
\end{aligned}$$

$$\begin{aligned}
\mathrm{lin}_2 &: RE_{plin} \to RE_{lin} \cup \{\emptyset\} \\
\mathrm{lin}_2(\alpha + \beta) &= \mathrm{lin}_2(\alpha) + \mathrm{lin}_2(\beta); \\
\mathrm{lin}_2((\alpha + \beta) \cdot \gamma) &= \mathrm{lin}_2(\alpha \cdot \gamma) + \mathrm{lin}_2(\beta \cdot \gamma); \\
\mathrm{lin}_2(\alpha) &= \alpha.
\end{aligned}$$

We also define the function `det`, which takes a linear regular expression α as argument and returns a deterministic linear regular expression β, such that $L(\alpha) = L(\beta)$, in the following way:

$$\begin{aligned}
\mathrm{det} &: RE_{lin} \cup \{\emptyset\} \to RE_{det} \cup \{\emptyset\} \\
\mathrm{det}(a \cdot \alpha + a \cdot \beta + \gamma) &= \mathrm{det}(a \cdot (\alpha + \beta) + \gamma); \\
\mathrm{det}(a \cdot \alpha + a \cdot \beta) &= a \cdot (\alpha + \beta); \\
\mathrm{det}(a \cdot \alpha + a) &= a \cdot (\alpha + \epsilon); \\
\mathrm{det}(\alpha) &= \alpha.
\end{aligned}$$

These functions implement Antimirov and Mosses's LF rewrite system. Function lin_1 corresponds to function f which, contrary to what is claimed by Antimirov and Mosses, returns a pre-linear regular expression, not a linear one.

It is easy to see that $\mathrm{const}(\alpha)$ returns the constant part of the regular expression α.

Lemma 2 *The function* lin_1 *is well defined.*

Proof *Let $a \in \Sigma$ and α, β, γ be arbitrary regular expressions.*

It is clear that for $\emptyset$, ϵ, *and* $a \in \Sigma$, *that the function* lin_1 *is well defined. By induction on the structure of a regular expression we can also conclude that for* $\alpha = \beta + \gamma$ *and* $\alpha = \beta^\star$ *the function* lin_1 *is well defined. We need only to show that* $\text{lin}_1(\alpha)$ *is also well defined when* α *is a concatenation of regular expressions. These are all the possible cases:*

$$\emptyset \cdot \alpha; \tag{4}$$
$$\alpha \cdot \emptyset; \tag{5}$$
$$\epsilon \cdot \alpha; \tag{6}$$
$$\alpha \cdot \epsilon; \tag{7}$$
$$a \cdot \alpha; \tag{8}$$
$$(\alpha + \beta) \cdot \gamma; \tag{9}$$
$$\alpha^\star \cdot \beta. \tag{10}$$

Because we are dealing with irreducible regular expressions modulo ACI, $\emptyset \cdot \alpha \sim \alpha \cdot \emptyset \sim \emptyset$ *and* $\text{lin}_1(\emptyset)$ *is well defined. For the same reason,* $\epsilon \cdot \alpha \sim \alpha \cdot \epsilon \sim \alpha$, *so we do not have to consider concatenations with the empty word* ϵ. *This leaves us with the cases 8, 9, and 10, all of which are explicitly considered.*

Lemma 3 *Given an arbitrary regular expression* α, $\text{lin}_1(\alpha)$ *is a pre-linear regular expression.*

Proof *A regular expression is pre-linear if it is generated by the context free grammar* G_2. *We will show that for an arbitrary r.e.* α, $\text{lin}_1(\alpha) \in L(G_2)$. *The proof follows by induction on the structure of* α.

Base:

$$\begin{aligned} \text{lin}_1(\emptyset) &= \emptyset; \\ \text{lin}_1(\epsilon) &= \emptyset; \\ \text{lin}_1(a) &= a, \qquad a \in \Sigma; \end{aligned}$$

Every one of these regular expressions is clearly generated by the grammar G_2.

Induction:

- $\text{lin}_1(a \cdot \alpha) = a \cdot \alpha$

  By induction hypothesis $\text{lin}_1(\alpha) \in L(G_2)$, *and clearly* $a \cdot \alpha$ *is generated by the grammar* G_2;

- $\text{lin}_1(\alpha + \beta) = \text{lin}_1(\alpha) + \text{lin}_1(\beta)$

  By induction hypothesis, both $\text{lin}_1(\alpha) \in L(G_2)$ *and* $\text{lin}_1(\beta) \in L(G_2)$. *The disjunction of two pre-linear regular expressions is derived by the production* $D \rightarrow D + D$, *therefore* $\text{lin}_1(\alpha + \beta) \in L(G_2)$;

- $\text{lin}_1((\alpha + \beta) \cdot \gamma) = \text{lin}_1(\alpha \cdot \gamma) + \text{lin}_1(\beta \cdot \gamma)$

  By induction hypothesis, $\text{lin}_1(\alpha \cdot \gamma) \in L(G_2)$ *and* $\text{lin}_1(\beta \cdot \gamma) \in L(G_2)$. *Again, the disjunction of two pre-linear regular expressions is also a pre-linear regular expression;*

- $\text{lin}_1(\alpha^\star \cdot \beta) = \text{lin}_1(\alpha) \cdot \alpha^\star \cdot \beta + \text{lin}_1(\beta)$

  By induction hypothesis, both $\text{lin}_1(\alpha)$ *and* $\text{lin}_1(\beta)$ *are generated by* G_2*. The concatenation* $\text{lin}_1(\alpha) \cdot \alpha^\star \cdot \beta$ *is clearly pre-linear and may be generated by the rule* $D \rightarrow D \cdot B$*. Just like with the previous cases, the disjunction of two pre-linear regular expressions is also a pre-linear regular expression and therefore* $\text{lin}_1(\alpha^\star \cdot \beta) \in L(G_2)$*;*

- $\text{lin}_1(\alpha^\star) = \text{lin}_1(\alpha) \cdot \alpha^\star$

  By induction hypothesis, $\text{lin}_1(\alpha) \in L(G_2)$*. Using the production* $D \rightarrow D \cdot B$ *we can derive* $\text{lin}_1(\alpha) \cdot \alpha^\star$*, so* $\text{lin}_1(\alpha^\star) \in L(G_2)$*.*

Lemma 4 *Given an arbitrary regular expression* α*,* $\text{lin}(\alpha)$ *returns either a linear r.e., or the empty language* $\emptyset$*.*

Proof *The function* lin *is defined as the composition of* lin_1 *with* lin_2*, and we have already seen that, for any r.e.* α*, the function* $\text{lin}_1(\alpha)$ *returns a pre-linear regular expression. We have to show that, given a pre-linear regular expression* $\alpha' \in L(G_2)$*, either* $\text{lin}_2(\alpha') = \emptyset$ *or* $\text{lin}_2(\alpha') \in L(G_1)$*. The proof follows by induction on the structure of* α'*. Let* $a \in \Sigma$*.*

Base:

- $\text{lin}_2(\emptyset) = \emptyset$*;*
- $\text{lin}_2(a) = a$*;*
- $\text{lin}_2(\alpha) = \alpha$*, if* α *is already linear, i.e.* $\alpha \in L(G_1)$*;*

Induction:

- $\text{lin}_2(\alpha + \beta) = \text{lin}_2(\alpha) + \text{lin}_2(\beta)$ *Note that if* $\alpha + \beta \in L(G_2)$ *then* $\alpha, \beta \in L(G_2)$*. Then, by induction hypothesis, both* $\text{lin}_2(\alpha) \in L(G_1)$ *and* $\text{lin}_2(\beta) \in L(G_1)$*. The disjunction of two linear regular expressions is also linear, and generated by the rule* $A \rightarrow A + A$ *of the grammar* G_1*.*

- $\text{lin}_2((\alpha + \beta) \cdot \gamma) = \text{lin}_2(\alpha \cdot \gamma) + \text{lin}_2(\beta \cdot \gamma)$

  Note that $\alpha \cdot \gamma$ *and* $\beta \cdot \gamma$ *are pre-linear. Then, by induction hypothesis, both* $\text{lin}_2(\alpha \cdot \gamma) \in L(G_1)$ *and* $\text{lin}_2(\beta \cdot \gamma) \in L(G_1)$*. As we have already seen, the disjunction of two linear regular expression is also a linear regular expression, and so,* $\text{lin}_2((\alpha + \beta) \cdot \gamma) \in L(G_1)$*.*

Lemma 5 *Given a linear regular expression* α*,* $\text{det}(\alpha)$ *will return a deterministic linear regular expression.*

Proof *We have to show that, for any given linear regular expression* α*,* $\text{head}(\text{det}(\alpha))$ *does not have repeated elements. Without loss of generality (we can repeat the process for each symbol), let us consider only one alphabet symbol* $a \in \Sigma$*.*

When there is only one sub-expression of the form $a \cdot \alpha'$ *in* α*,* $\text{det}(\alpha) = \alpha$ *and, considering only the symbol* a*, there will be no repeated elements in* $\text{head}(\text{det}(\alpha))$*. With two sub-expressions of the same form, we will have that either* $\text{det}(\alpha) = a \cdot (\beta + \gamma)$ *or* $\text{det}(\alpha) = a \cdot (\beta + \epsilon)$ *and again, considering only the symbol* a*,* $\text{head}(\text{det}(\alpha))$ *will have no repeated elements.*

Now, suppose that there are n sub-expressions with prefix a. By definition, $\det(\alpha)$ will use the distributive property, reduce the number of sub-expressions of the form $a \cdot \alpha'$ to $n-1$, and apply itself again. After $n-1$ applications of $\det(\alpha)$, the resulting regular expression β will have only one element of the form $a \cdot \alpha'$, and there will not be repeated elements in $\text{head}(\beta)$.

Lemma 6 *For every regular expression $\alpha \in RE_{lin} \cup \{\emptyset\}$ we have that*

$$\alpha \sim \det(\alpha).$$

The proof follows by induction on the number of operators in α.

Proof *If $\alpha \in RE_{lin}$, α is generated by the grammar G_1 and has one of the following forms:*

$$\begin{gathered} a_0, \\ a_0 \cdot \alpha_0 + \cdots + a_n \cdot \alpha_n, \end{gathered}$$

where $a_i \in \Sigma$, and $\alpha_i \in RE$. We have the following base cases:

$$\begin{aligned} \emptyset &\sim \det(\emptyset) = \emptyset; \\ a_0 &\sim \det(a_0) = a_0; \\ a_0\alpha_0 &\sim \det(a_0\alpha_0) = a_0\alpha_0; \\ a_0\alpha_0 + a_0 &\sim \det(a_0\alpha_0 + a_0) = a_0(\alpha_0 + \epsilon); \\ a_0\alpha_0 + a_0\alpha_1 &\sim \det(a_0\alpha_0 + a_0\alpha_1) = a_0(\alpha_0 + \alpha_1). \end{aligned}$$

Let $\alpha \in RE_{lin}$ have n operators and suppose, by induction hypothesis, that for every linear regular expression β with $n-1$ or less operators,

$$\beta \sim \det(\beta),$$

i.e., $L(\beta) = L(\det(\beta))$.

$$\begin{aligned} \det(\alpha) &= \det(a_0\alpha_0 + a_0\alpha_1 + \gamma) \\ &= \det(a_0(\alpha_0 + \alpha_1) + \gamma) \end{aligned}$$

By induction hypothesis,

$$L(a_0(\alpha_0 + \alpha_1) + \gamma) = L(\det(a_0(\alpha_0 + \alpha_1) + \gamma)).$$

As

$$L(a_0\alpha_0 + a_0\alpha_1 + \gamma) = L(a_0(\alpha_0 + \alpha_1) + \gamma),$$

we have that

$$L(a_0\alpha_0 + a_0\alpha_1 + \gamma) = L(\det(a_0(\alpha_0 + \alpha_1) + \gamma)),$$

i.e.,

$$a_0\alpha_0 + a_0\alpha_1 + \gamma \sim \det(a_0(\alpha_0 + \alpha_1) + \gamma).$$

Lemma 7 *For every regular expression $\alpha \in RE_{plin}$ we have that*

$$\alpha \sim \text{lin}_2(\alpha).$$

The proof follows by induction on the number of operators in α.

Proof *If $\alpha \in RE_{plin}$, then $\alpha \in L(G_2)$.We have the following base cases:*

$$\begin{aligned} \emptyset &\sim \text{lin}_2(\emptyset) = \emptyset; \\ a &\sim \text{lin}_2(a) = a; \\ a \cdot \alpha &\sim \text{lin}_2(a \cdot \alpha) = a \cdot \alpha. \end{aligned}$$

Let α be a r.e. with n operators and suppose, by induction hypothesis, that for every regular expression β with $n-1$ or less operators

$$\beta \sim \text{lin}_2(\beta).$$

We have the following cases:

- $\alpha = \beta + \gamma$

$$\begin{aligned} \text{lin}_2(\beta + \gamma) &= \text{lin}_2(\beta) + \text{lin}_2(\gamma) && \textit{by definition,} \\ &= \beta + \gamma && \textit{by induction hypothesis.} \end{aligned}$$

- $\alpha = (\beta + \gamma) \cdot \psi$, *and β is linear*

$$\begin{aligned} \text{lin}_2((\beta + \gamma) \cdot \psi) &= \beta \cdot \psi + \text{lin}_2(\gamma \cdot \psi) && \textit{by definition,} \\ &= \beta \cdot \psi + \gamma \cdot \psi && \textit{by induction hypothesis,} \\ &= (\beta + \gamma) \cdot \psi && \textit{by Axiom } A_9. \end{aligned}$$

- $\alpha = (\beta + \gamma) \cdot \psi$, *and β is pre-linear*

$$\begin{aligned} \text{lin}_2((\beta + \gamma) \cdot \psi) &= \text{lin}_2(\beta \cdot \psi) + \text{lin}_2(\gamma \cdot \psi) && \textit{by definition,} \\ &= \beta \cdot \psi + \gamma \cdot \psi && \textit{by induction hypothesis,} \\ &= (\beta + \gamma) \cdot \psi && \textit{by Axiom } A_9. \end{aligned}$$

Lemma 8 *Let α be an arbitrary regular expression. We have that*

$$L(\text{lin}(\alpha)) = \begin{cases} L(\alpha) & \textit{if } \epsilon \notin L(\alpha); \\ L(\alpha) - \{\epsilon\} & \textit{if } \epsilon \in L(\alpha). \end{cases}$$

That is, the application of the function lin *will remove the empty word from the language defined by α.*

Proof *Because $\text{lin} = \text{lin}_2 \circ \text{lin}_1$ and $\alpha \sim \text{lin}_2(\alpha)$, cf. Lemma 7, we need only to consider the application of lin_1. Without lost of generality, we are going to prove that $L(\text{lin}_1(\alpha)) = L(\alpha) - \{\epsilon\}$. The proof follows by induction on the number of operators of α. Let $a \in \Sigma$ and $\alpha, \beta, \gamma \in RE$.*

Base:

- $L(\text{lin}_1(\emptyset)) = \emptyset = L(\emptyset) - \{\epsilon\}$;
- $L(\text{lin}_1(\epsilon)) = \emptyset = L(\epsilon) - \{\epsilon\}$;
- $L(\text{lin}_1(a)) = \{a\} = L(a) - \{\epsilon\}$;

Induction:

- $\mathrm{lin}_1(a \cdot \alpha)$

$$\begin{aligned} L(\mathrm{lin}_1(a \cdot \alpha)) &= L(a \cdot \alpha) \\ &= L(a \cdot \alpha) - \{\epsilon\}; \end{aligned}$$

- $\mathrm{lin}_1(\alpha + \beta)$

$$\begin{aligned} L(\mathrm{lin}_1(\alpha + \beta)) &= L(\mathrm{lin}_1(\alpha) + \mathrm{lin}_1(\beta)) \\ &= L(\mathrm{lin}_1(\alpha)) \cup L(\mathrm{lin}_1(\beta)) \\ &= L(\alpha) - \{\epsilon\} \cup L(\beta) - \{\epsilon\} \\ &= (L(\alpha) \cup L(\beta)) - \{\epsilon\} \\ &= L(\alpha + \beta) - \{\epsilon\}; \end{aligned}$$

- $\mathrm{lin}_1((\alpha + \beta) \cdot \gamma)$

$$\begin{aligned} L(\mathrm{lin}_1((\alpha + \beta) \cdot \gamma) &= L(\mathrm{lin}_1(\alpha \cdot \gamma) + \mathrm{lin}_1(\beta \cdot \gamma)) \\ &= L(\mathrm{lin}_1(\alpha \cdot \gamma)) \cup L(\mathrm{lin}_1(\beta \cdot \gamma)) \\ &= L(\alpha \cdot \gamma) - \{\epsilon\} \cup L(\beta \cdot \gamma) - \{\epsilon\} \\ &= (L(\alpha \cdot \gamma) \cup L(\beta \cdot \gamma)) - \{\epsilon\} \\ &= L(\alpha \cdot \gamma + \beta \cdot \gamma) - \{\epsilon\} \\ &= L((\alpha + \beta) \cdot \gamma) - \{\epsilon\}; \end{aligned}$$

- $\mathrm{lin}_1(\alpha^\star)$

$$\begin{aligned} L(\mathrm{lin}_1(\alpha^\star)) &= L(\mathrm{lin}_1(\alpha) \cdot \alpha^\star) \\ &= L(\mathrm{lin}_a(\alpha)) \cdot L(\alpha^\star) \\ &= (L(\alpha) - \{\epsilon\}) \cdot L(\alpha^\star) \\ &= (L(\alpha) \cap \overline{\{\epsilon\}}) \cdot L(\alpha^\star) \\ &= L(\alpha \cdot \alpha^\star) \cap \overline{\{\epsilon\}} \cdot L(\alpha^\star) \\ &= L(\alpha \cdot \alpha^\star) \cap \overline{\{\epsilon\}} \\ &= (L(\alpha \cdot \alpha^\star) \cap \overline{\{\epsilon\}}) \cup (\{\epsilon\} \cap \overline{\{\epsilon\}}) \\ &= (L(\alpha \cdot \alpha^\star) \cup \{\epsilon\}) \cap \overline{\{\epsilon\}} \\ &= L(\alpha \cdot \alpha^\star) - \{\epsilon\} \\ &= L(\alpha^\star) - \{\epsilon\}; \end{aligned}$$

- $\mathrm{lin}_1(\alpha^\star \cdot \beta)$

$$\begin{aligned}
L(\mathrm{lin}_1(\alpha^\star \cdot \beta)) &= L(\mathrm{lin}_1(\alpha) \cdot \alpha^\star \cdot \beta + \mathrm{lin}_1(\beta)) \\
&= L(\mathrm{lin}_1(\alpha) \cdot \alpha^\star \cdot \beta) \cup L(\mathrm{lin}_1(\beta)) \\
&= L(\mathrm{lin}_1(\alpha)) \cdot L(\alpha^\star \cdot \beta) \cup L(\mathrm{lin}_1(\beta)) \\
&= (L(\alpha) - \{\epsilon\}) \cdot L(\alpha^\star \cdot \beta) \cup (L(\beta) - \{\epsilon\}) \\
&= (L(\alpha) \cap \overline{\{\epsilon\}}) \cdot L(\alpha^\star \cdot \beta) \cup (L(\beta) \cap \overline{\{\epsilon\}}) \\
&= (L(\alpha) \cdot L(\alpha^\star \cdot \beta) \cap \overline{\{\epsilon\}} \cdot L(\alpha^\star \cdot \beta)) \cup (L(\beta) \cap \overline{\{\epsilon\}}) \\
&= (L(\alpha) \cdot L(\alpha^\star \cdot \beta) \cap \overline{\{\epsilon\}}) \cup (L(\beta) \cap \overline{\{\epsilon\}}) \\
&= (L(\alpha) \cdot L(\alpha^\star) \cdot L(\beta) \cap \overline{\{\epsilon\}}) \cup (L(\beta) \cap \overline{\{\epsilon\}}) \\
&= (L(\alpha) \cdot L(\alpha^\star) \cdot L(\beta) \cup L(\beta)) \cap \overline{\{\epsilon\}} \\
&= (L(\alpha) \cdot L(\alpha^\star) \cup \{\epsilon\}) \cdot L(\beta) \cap \overline{\{\epsilon\}} \\
&= L(\alpha \cdot \alpha^\star + \epsilon) \cdot L(\beta) \cap \overline{\{\epsilon\}} \\
&= L(\alpha^\star) \cdot L(\beta) \cap \overline{\{\epsilon\}} \\
&= L(\alpha^\star \cdot \beta) \cap \overline{\{\epsilon\}} \\
&= L(\alpha^\star \cdot \beta) - \{\epsilon\}.
\end{aligned}$$

Theorem 1 *For any regular expression α,*

$$\alpha \sim \mathrm{const}(\alpha) + \mathrm{lin}(\alpha) \quad \textit{and} \quad \alpha \sim \mathrm{const}(\alpha) + \mathrm{det}(\mathrm{lin}(\alpha)).$$

Proof *There are two cases to consider.*

1. *If $\epsilon \in L(\alpha)$:*

$$\begin{aligned}
L(\mathrm{const}(\alpha) + \mathrm{lin}(\alpha)) &= L(\mathrm{const}(\alpha)) \cup L(\mathrm{lin}(\alpha)) && \textit{by Definition 2} \\
&= L(\epsilon) \cup L(\mathrm{lin}(\alpha)) && \epsilon \in L(\alpha) \\
&= \{\epsilon\} \cup (L(\alpha) - \{\epsilon\}) && \textit{by Lemma 8} \\
&= L(\alpha)
\end{aligned}$$

2. *If $\epsilon \notin L(\alpha)$:*

$$\begin{aligned}
L(\mathrm{const}(\alpha) + \mathrm{lin}(\alpha)) &= L(\mathrm{const}(\alpha)) \cup L(\mathrm{lin}(\alpha)) && \textit{by Definition 2} \\
&= L(\emptyset) \cup L(\mathrm{lin}(\alpha)) && \epsilon \notin L(\alpha) \\
&= \emptyset \cup L(\mathrm{lin}(\alpha)) && \textit{by Definition 2} \\
&= L(\alpha) - \{\epsilon\} && \textit{by Lemma 8} \\
&= L(\alpha) && \epsilon \notin L(\alpha)
\end{aligned}$$

As the definition of the function det *does not change the language defined by its argument, cf. Lemma 6, it is clear that $L(\mathrm{det}(\mathrm{lin}(\alpha))) = L(\mathrm{lin}(\alpha))$. So, if $\alpha \sim \mathrm{const}(\alpha) + \mathrm{lin}(\alpha)$, we also have that $\alpha \sim \mathrm{const}(\alpha) + \mathrm{det}(\mathrm{lin}(\alpha))$.*

Lemma 9 *If a and $\alpha \in RE$, then $a^{-1}(\alpha) = a^{-1}(\mathrm{lin}_1(\alpha))$.*

Proof *The proof follows by induction on the number of operators in the expressions.*

Base:

$$
\begin{aligned}
a^{-1}(\emptyset) &= \emptyset, & a^{-1}(\mathrm{lin}_1(\emptyset)) &= a^{-1}(\emptyset) = \emptyset; \\
a^{-1}(\epsilon) &= \emptyset, & a^{-1}(\mathrm{lin}_1(\epsilon)) &= a^{-1}(\emptyset) = \emptyset; \\
a^{-1}(b) &= \begin{cases} \epsilon & \text{if } a = b \ , \\ \emptyset & \text{if } a \neq b \ ; \end{cases} & a^{-1}(\mathrm{lin}_1(b)) &= a^{-1}(b) = \begin{cases} \epsilon & \text{if } a = b \ , \\ \emptyset & \text{if } a \neq b \ ; \end{cases}
\end{aligned}
$$

Induction:

- *Disjunction:*

$$
\begin{aligned}
&a^{-1}(\alpha + \beta) \\
&\qquad = a^{-1}(\alpha) + a^{-1}(\beta) \\
&\qquad = a^{-1}(\mathrm{lin}_1(\alpha)) + a^{-1}(\mathrm{lin}_1(\beta)) \\
&\qquad = a^{-1}(\mathrm{lin}_1(\alpha) + \mathrm{lin}_1(\beta)) \\
&\qquad = a^{-1}(\mathrm{lin}_1(\alpha + \beta)).
\end{aligned} \tag{11}
$$

- *Star closure:*

$$
\begin{aligned}
&a^{-1}(\alpha^\star) \\
&\qquad = a^{-1}(\alpha) \cdot \alpha^\star \\
&\qquad = a^{-1}(\mathrm{lin}_1(\alpha)) \cdot \alpha^\star \\
&\qquad = a^{-1}(\mathrm{lin}_1(\alpha)) \cdot \alpha^\star + \emptyset \cdot \alpha^\star \\
&\qquad = a^{-1}(\mathrm{lin}_1(\alpha)) \cdot \alpha^\star + \mathrm{const}(\mathrm{lin}_1(\alpha)) \cdot \alpha^\star \\
&\qquad = a^{-1}(\mathrm{lin}_1(\alpha) \cdot \alpha^\star) \\
&\qquad = a^{-1}(\mathrm{lin}_1(\alpha^\star)).
\end{aligned} \tag{12}
$$

- *As for the concatenation, there are three cases to consider:*

$$\begin{aligned}
&a^{-1}(a\cdot\beta) = a^{-1}(\mathrm{lin}_1(a\cdot\beta));\\
&a^{-1}((\alpha+\beta)\cdot\gamma) = a^{-1}(\alpha\cdot\gamma+\beta\cdot\gamma)\\
&\qquad = a^{-1}(\alpha\cdot\gamma)+a^{-1}(\beta\cdot\gamma)\\
&\qquad = a^{-1}(\mathrm{lin}_1(\alpha\cdot\gamma))+a^{-1}(\mathrm{lin}_1(\beta\cdot\gamma))\\
&\qquad = a^{-1}(\mathrm{lin}_1(\alpha\cdot\gamma)+\mathrm{lin}_1(\beta\cdot\gamma))\\
&\qquad = a^{-1}(\mathrm{lin}_1(\alpha\cdot\gamma+\beta\cdot\gamma))\\
&\qquad = a^{-1}(\mathrm{lin}_1(\alpha+\beta)\cdot\gamma);\\
&a^{-1}(\alpha^\star\cdot\beta) = a^{-1}(\alpha^\star)\cdot\beta+\mathrm{const}(\alpha^\star)\cdot a^{-1}(\beta)\\
&\qquad = a^{-1}(\alpha)\cdot\alpha^\star\cdot\beta+\epsilon\cdot a^{-1}(\beta)\\
&\qquad = a^{-1}(\alpha)\cdot\alpha^\star\cdot\beta+a^{-1}(\beta)\\
&\qquad = a^{-1}(\mathrm{lin}_1(\alpha))\cdot\alpha^\star\cdot\beta+a^{-1}(\mathrm{lin}_1(\beta))\\
&\qquad = a^{-1}(\mathrm{lin}_1(\alpha))\cdot\alpha^\star\cdot\beta+\emptyset\cdot\alpha^\star\cdot a^{-1}(\beta)+a^{-1}(\mathrm{lin}_1(\beta))\\
&\qquad = a^{-1}(\mathrm{lin}_1(\alpha))\cdot\alpha^\star\cdot\beta+\mathrm{const}(\mathrm{lin}_1(\alpha))\cdot\alpha^\star\cdot a^{-1}(\beta)+a^{-1}(\mathrm{lin}_1(\beta))\\
&\qquad = a^{-1}(\mathrm{lin}_1(\alpha)\cdot\alpha^\star\cdot\beta)+a^{-1}(\mathrm{lin}_1(\beta))\\
&\qquad = a^{-1}(\mathrm{lin}_1(\alpha)\cdot\alpha^\star\cdot\beta+\mathrm{lin}_1(\beta))\\
&\qquad = a^{-1}(\mathrm{lin}_1(\alpha^\star\cdot\beta))
\end{aligned} \tag{13}$$

Theorem 2 *Let $a \in \Sigma$ and $\alpha \in RE$. The following equality holds:*

$$a^{-1}(\alpha) = a^{-1}(\det(\mathrm{lin}(\alpha))).$$

Proof *For every regular expression α, it is clear that*

$$L(\det(\mathrm{lin}(\alpha))) = L(\mathrm{lin}(\alpha)) = L(\mathrm{lin}_1(\alpha)). \tag{14}$$

By Lemma 9 we have that

$$a^{-1}(\alpha) \sim a^{-1}(\mathrm{lin}_1(\alpha)). \tag{15}$$

So, by equations 14 and 15

$$a^{-1}(\alpha) \sim a^{-1}(\mathrm{lin}_1(\alpha)) \sim a^{-1}(\mathrm{lin}(\alpha)) \sim a^{-1}(\det(\mathrm{lin}(\alpha))).$$

3.2 Core functions

We will now present the two main functions of the comparison process. The first one, derivatives, computes the set of the derivatives of a pair of deterministic linear regular expressions or $\emptyset$ (α, β), with respect to $\{\, a \mid a \in \mathrm{head}(\alpha) \cup \mathrm{head}(\beta) \,\}$. It is defined as follows.

$$\begin{aligned}
\text{derivatives} &\;:\; RE_{det}\cup\{\emptyset\}\times RE_{det}\cup\{\emptyset\}\to\mathcal{P}(RE\times RE)\\
\text{derivatives}(\emptyset,\emptyset) &\;=\; \{\};\\
\text{derivatives}(\alpha,\beta) &\;=\; \{\,(a^{-1}(\alpha),a^{-1}(\beta)) \mid a\in\mathrm{head}(\alpha)\cup\mathrm{head}(\beta)\,\}.
\end{aligned} \tag{16}$$

The equiv function, applies the method to two regular expressions α and β, returning *True* if and only if $\alpha \sim \beta$. It is defined in the following way:

$$\begin{aligned}
\text{equiv} \quad &: \quad \mathcal{P}(RE^2) \times \mathcal{P}(RE^2) \to \{True, False\} \\
\text{equiv}(\emptyset, H) \quad &= \quad True; \\
\text{equiv}(\{(\alpha,\beta)\} \cup S, H) \quad &= \quad \begin{cases} False & \text{if } \text{const}(a) \neq \text{const}(b); \\ \text{equiv}(S \cup S', H') & \text{otherwise;} \end{cases}
\end{aligned} \tag{17}$$

where

$$\begin{aligned}
\alpha' &= \det(\text{lin}(\alpha)); \\
\beta' &= \det(\text{lin}(\beta)); \\
S' &= \{\, p \mid p \in \text{derivatives}(\alpha', \beta'),\ p \notin H' \,\}; \\
H' &= \{\, (\alpha, \beta) \,\} \cup H.
\end{aligned}$$

Theorem 3 *The function* equiv *is terminating*

Proof *It is clear that the function terminates when its first argument, the set S, is empty.*

Each call to the function will remove one element from S, and append the set of the derivatives which have not yet been calculated, S'. By Theorem 2, we know that the linearisation process does not affect the derivation process. This sequence of derivatives with respect to a symbol is equivalent to a single derivative with respect to a word $w \in \Sigma^\star$, cf. Definition 13. Brzozowski [Brz64] showed that every regular expression α has a finite number of dissimilar derivatives with respect to any word w. Because all operations are performed modulo ACI, we consider only dissimilar regular expressions, and, from a given point on, $S \cup S' = S$. As each call to equiv *will remove one element from S, eventually $S = \emptyset$ and the function terminates.*

Lemma 10 *Let α and β be either deterministic linear regular expressions or $\emptyset$.*

$$\alpha \sim \beta \Rightarrow \alpha' \sim \beta', \quad \forall (\alpha', \beta') \in \text{derivatives}(\alpha, \beta) \tag{18}$$

Proof *If*

$$\begin{aligned}
\alpha = \beta = \emptyset, &\qquad \text{derivatives}(\alpha, \beta) = \{\}; \\
\alpha = \beta = a, &\qquad \text{derivatives}(\alpha, \beta) = \{(\epsilon, \epsilon)\}. \qquad \textit{(for } a \in \Sigma\textit{)}
\end{aligned}$$

Otherwise, let $\alpha = a_1\alpha_1 + \cdots + a_i\alpha_n$ and let $\beta = a_1\beta_1 + \cdots + a_m\beta_m$. Note that we must have $m = n$. Suppose that there exists $j \in [1, n]$ such that $\alpha_j \not\sim \beta_j$. Then $\alpha \not\sim \beta$. Absurd.

Lemma 11 *Given two regular expressions, α and β, such that $\alpha \sim \beta$,*

$$\text{equiv}(\{(\alpha, \beta)\}, \emptyset) = True.$$

Proof *If $\alpha = \beta = \emptyset$,*

$$\begin{aligned}
\text{equiv}(\{(\emptyset, \emptyset)\}, H) &= \text{equiv}(\emptyset, H \cup \{(\emptyset, \emptyset)\}) \\
&= True.
\end{aligned}$$

If $\alpha \sim \beta$ we know, by Lemma 10, that

$$\text{const}(\alpha') = \text{const}(\beta') \qquad \forall (\alpha', \beta') \in \text{derivatives}(\alpha, \beta)$$

and thus, the call to $\text{equiv}(\alpha, \beta, \emptyset)$ *will never return $False$.*

Lemma 12 *Given two regular expressions, α and β, such that $\alpha \not\sim \beta$,*

$$\text{equiv}(\{(\alpha,\beta)\},\emptyset) = False.$$

Proof *If $\alpha \not\sim \beta$, either*

$$\exists w \in L(\alpha) : w^{-1}(\beta) \neq \epsilon \qquad \text{or} \qquad \exists w \in L(\beta) : w^{-1}(\alpha) \neq \epsilon.$$

Without loss of generality, suppose the first case is true. As Brzozowski [Brz64] shows, if $w \in L(\alpha)$, $w^{-1}(\alpha) = \epsilon$. By definition of derivative, for any word w' such that $|w'| > |w|$, ${w'}^{-1}(\alpha) = \emptyset$. Moreover, if $w^{-1}(\beta) \neq \epsilon$ there will be some word wv such that $wv \in L(\beta)$ and $|wv| > |w|$. Under these conditions we have that

$$\left.\begin{array}{l}(wv)^{-1}(\alpha) = \emptyset\\ (wv)^{-1}(\beta) = \epsilon\end{array}\right\} \quad \text{const}((wv)^{-1}(\alpha)) \neq \text{const}((wv)^{-1}(\beta)),$$

and $\text{equiv}(\{((wv)^{-1}(\alpha),(wv)^{-1}(\beta))\} \cup S, H) = False.$

Theorem 4 *The function call* $\text{equiv}(\{(\alpha,\beta)\},\emptyset)$ *returns $True$ if and only if $L(\alpha) = L(\beta)$*

Proof *By direct application of Lemmas 11 and 12.*

4 Representation and implementation of regular expressions

As already stated, throughout this paper, we always consider irreducible regular expressions modulo ACI, i.e., *associativity* of the concatenation and disjunction, *commutativity* of the disjunction, and *idempotence* of both disjunction and star operations. For each operation, we used a data structure that enforces the ACI properties and simplifies the algorithms used to assure that the regular expressions are kept irreducible. Disjunctions are represented as a set of regular expressions. Concatenated regular expressions are kept in an ordered list. The idempotence of the Kleene star is assured by not allowing double-starred regular expressions.

4.1 Disjunctions

A disjunction is represented as a set of regular expressions. This gives us a natural way to enforce the ACI properties:

- *associativity*: there is no pairwise association of any two arguments, so $(\alpha + \beta) + \gamma = \alpha + (\beta + \gamma)$;
- *commutativity*: by definition of set, the order of the elements is irrelevant, so $\alpha + \beta = \beta + \alpha$;
- *idempotence*: also by definition of set, there are no repeated elements, so $\alpha + \alpha = \alpha$.

As for making a disjunction irreducible, there are only two conditions that must hold:

- the sub-expression $\emptyset$ may not occur;
- if any of the arguments possesses the e.w.p., the empty word ϵ is not allowed as a sub-expression.

Again, the set representation yields a linear algorithm to keep any disjunction irreducible. As an example, consider the regular expression $\alpha + \emptyset + \beta^\star\gamma + \alpha + \epsilon$. It would be represented by the set

$$\{\epsilon, \alpha, \beta^\star\gamma\}.$$

4.2 Concatenation

Concatenations of regular expressions are kept in an ordered list. This allows us to take advantage of the associative property and easily apply transformations to any pair of adjacent regular expressions. This representation also simplifies the following transformations:

$$\begin{aligned} \alpha \cdot \epsilon &\rightarrow \alpha, \\ \epsilon \cdot \alpha &\rightarrow \alpha, \\ \alpha \cdot \emptyset &\rightarrow \emptyset, \\ \emptyset \cdot \alpha &\rightarrow \emptyset, \end{aligned}$$

which are necessary to make the regular expressions irreducible, as we can simply go through the list and remove each occurrence of ϵ. If the $\emptyset$ r.e. if found, we simply return it as the equivalent irreducible regular expression. Consider the following examples:

$$\begin{aligned} \alpha \cdot \epsilon \cdot \beta &\rightarrow [\alpha, \beta]; \\ \alpha \cdot \emptyset \cdot \beta &\rightarrow \emptyset. \\ \alpha \cdot \beta^{\star} \cdot \gamma &\rightarrow [\alpha, \beta^{\star}, \gamma]. \end{aligned}$$

4.3 Kleene star

As for the Kleene star, we use a class to represent the $\star$ operator.

In order to keep it irreducible, the constructor of the class does not create regular expressions of the form $\alpha^{\star\star}$. This is done by checking if the r.e. passed as an argument is already of the same type. If this is the case, only the argument is kept, thus avoiding the double star.

We also added two simplifications to the star operator representation:

$$\begin{aligned} \emptyset^{\star} &\rightarrow \epsilon; \\ \epsilon^{\star} &\rightarrow \epsilon. \end{aligned}$$

Although these are not necessary to make the regular expressions irreducible, they may allow for significant simplifications which can be very useful to the system described in the Section 3.

5 Experimental results

We now present some experimental results. These are the running times of two methods for checking the equivalence of regular expressions. One uses the equivalent minimal DFA, the other is the direct regular expression comparison method, as described on Section 3. All tests were performed on batches of 10.000 pairs of randomly generated regular expressions, and the running times do not include the time necessary to parse each regular expression. Each batch contains regular expressions of size 10, 50, or 100, with either 2, 5, or 10 symbols. For the uniform generation of random regular expressions we implemented the method proppposed by Mairson [Mai94] for the generation of context-free languages defined by unambiguous grammars. We used the grammar of almost irreducible regular expressions presented by Shallit [Sha04].

5.1 Results

The equivalence of each pair of regular expressions was tested using both the classical approach and the direct comparison method. We used Thompson's algorithm to obtain the NFAs from the regular expressions, and the well-known subset construction to make each NFA deterministic. As for the DFA minimisation process, we applied two different algorithms: Hopcroft and Brzozowski's. On one hand, Hopcroft's algorithm has the best known worst-case running time complexity analysis, $O(kn\log(n))$. On the other, it is pointed out by Almeida *et. al* [AMR07] that when minimising NFAs, Brzozowski's algorithm has a better practical performance. As for the direct comparison method, we compared both the original rewriting system (**AM**) and our variant of the algorithm (**equiv**).

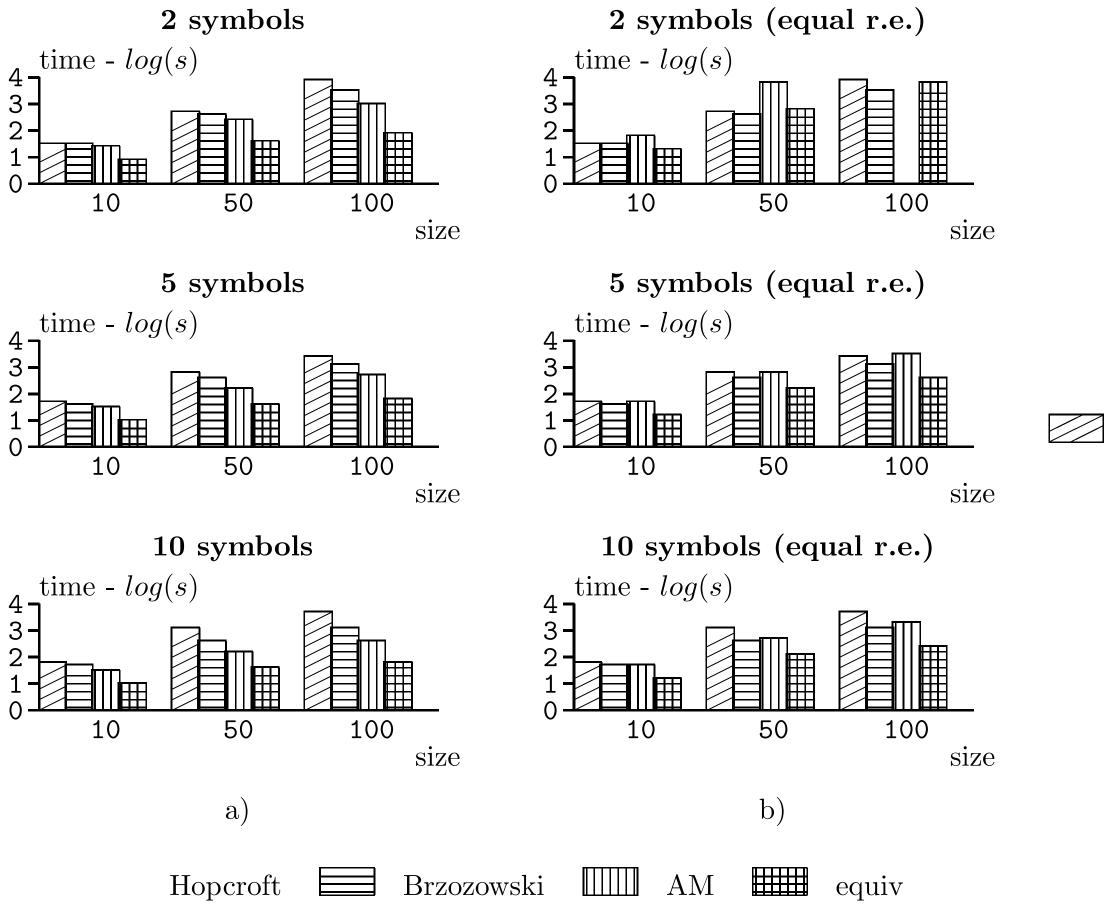

Figure 1: Running times of three different methods for checking the equivalence of regular expressions. a) 10.000 pairs of random r.e.; b) 10.000 pairs of syntactically equal random r.e.. The missing column corresponds to a larger than reasonable observed runtime.

As shown in Figure 1 (a), the direct method is always the fastest. Note also that Hopcroft's algorithm never achieves shorter running times than Brzozowki's. Because both Antimirov and Mosses's algorithm and our variation try to compute a refutation, we performed a set of tests for the worst case scenario of these methods: to test the equivalence of two syntactically equal regular expressions. Again, we used batches of $10,000$ pairs of regular expressions. Except for the samples of regular expressions with size 50 or 100, and an alphabet with 2 symbols, the direct regular expression comparison method is still the fastest. The results are presented on Figure 1 (b).

5.2 More experimental data

Considering the somewhat surprising results of the previous tests, we analysed the time spent on each step of both methods, *i.e.*, on the construction of the NFAs (**NFA**), the determinisation process (**DFA**), and the minimisation algorithms (**mDFA**). The running time for each of these steps is presented on Table 1. It is clear that, asymptotically, the bottleneck is the minimisation algorithm, which always takes over 50% of the total amount of time when the size of the regular expressions and/or the alphabet grows.

$k = 2$	$n = 10$		$n = 50$		$n = 100$	
	$\|Q\|$	time (s)	$\|Q\|$	time (s)	$\|Q\|$	time (s)
NFA	$(18.35, 18.34)$	14.73	$(83.85, 83.90)$	212.66	$(165.60, 165.69)$	563.94
DFA	$(6.52, 6.55)$	8.40	$(33.85, 33.96)$	119.60	$(114.68, 114.20)$	659.03
mDFA	$(6.43, 6.45)$	10.01	$(26.18, 26.24)$	212.84	$(74.79, 74.20)$	7680.89

$k = 5$	$n = 10$		$n = 50$	
	$\|Q\|$	time (s)	$\|Q\|$	time (s)
NFA	$(19.53, 19.54)$	18.30	$(90.30, 90.32)$	262.11
DFA	$(8.59, 8.60)$	11.71	$(34.88, 34.91)$	71.60
mDFA	$(8.41, 8.42)$	27.50	$(29.87, 29.89)$	417.59

$k = 10$	$n = 10$		$n = 50$	
	$\|Q\|$	time (s)	$\|Q\|$	time (s)
NFA	$(19.85, 19.85)$	21.01	$(93.98, 93.93)$	311.22
DFA	$(9.51, 9.51)$	13.80	$(39.96, 39.95)$	84.67
mDFA	$(9.51, 9.52)$	43.79	$(35.38, 35.38)$	1013.45

Table 1: Running times for each step of the r.e. comparison processes.

	$k = 2$			$k = 5$		$k = 10$	
	$n = 10$	$n = 50$	$n = 100$	$n = 10$	$n = 50$	$n = 10$	$n = 50$
AM	3.00	3.87	4.30	5.78	4.50	8.13	13.57
equiv	2.73	3.60	4.01	9.53	7.35	6.46	10.67

Table 2: Average number of functions calls to both **AM** and **equiv**.

We also collected some statistical data about the average behaviour of the algorithms. Table 1 shows the average number of states for the NFAs built from the regular expressions and the average number of states of the equivalent DFAs. The average number of recursive calls to the equiv function on both Antimirov and Mosses's algorithm and our own implementation is presented on Table 2. The values are similar, so the difference on the running time of the algorithms can not be justified by the number of function calls. The average number of states produced by the Thompson construction is quite large and not close to the resulting equivalent DFA. To ensure the fairness of the comparison for the methods using NFAs, we tried two other algorithms for computing NFAs from regular expressions.

Glushkov's method [Glu61, Yu97], and the one that produces the "follow NFA" of a given regular expression, as described by Ilie and Yu [IY03]. This statistical data is given on Table 3.

$k=2$	$n=10$			$n=50$			$n=100$		
Alg.	$\|Q\|$	$\|\delta\|$	time (s)	$\|Q\|$	$\|\delta\|$	time (s)	$\|Q\|$	$\|\delta\|$	time (s)
Thompson	18.35	17.46	8.79	83.82	82.90	110.47	165.72	164.80	411.42
Glushkov	7.45	7.37	4.48	29.45	35.96	33.83	57.05	72.90	98.02
Follow	5.91	6.10	26.75	20.30	25.36	813.64	35.93	46.52	5135.32

$k=5$	$n=10$			$n=50$			$n=100$		
Alg.	$\|Q\|$	$\|\delta\|$	time (s)	$\|Q\|$	$\|\delta\|$	time (s)	$\|Q\|$	$\|\delta\|$	time (s)
Thompson	19.54	18.60	10.79	90.38	89.42	133.31	178.38	177.42	484.02
Glushkov	8.91	9.19	5.16	36.53	45.83	36.18	70.98	93.55	101.55
Follow	7.54	7.98	41.71	28.63	35.80	1723.54	53.15	68.55	12413.52

$k=10$	$n=10$			$n=50$			$n=100$		
Alg.	$\|Q\|$	$\|\delta\|$	time (s)	$\|Q\|$	$\|\delta\|$	time (s)	$\|Q\|$	$\|\delta\|$	time (s)
Thompson	19.85	18.90	11.69	94.03	93.05	153.43	185.64	184.66	554.47
Glushkov	9.65	9.70	5.34	40.88	48.03	36.84	79.64	97.13	101.14
Follow	8.60	8.72	51.99	34.67	40.48	2585.59	66.18	78.84	21086.45

Table 3: Running time, average number of states and transitions for three types of NFA obtained from random expressions.

Glushkov's algorithm is always the fastest, and produces quite small NFAs, both in terms of number of states and transitions. As expected, the NFAs produced with Thompson's algorithm are the ones with the higher number of states.

6 Conclusions

We presented method for testing the equivalence of two regular expressions based on a rewrite system presented by Antimirov and Mosses. The method attempts to refute the equivalence by finding a pair of derivatives that disagree in their constant parts. Experimental results point to a good average-case performance for this method, even when using with equivalent regular expressions. Given the spread of dual-cores and grid computer systems, a parallel execution of the classic method and our direct comparison method can lead to an optimized framework for testing r.e. equivalence.

A better theoretical understanding of relationships between the two approaches would be helpful towards the characterization of their average-case complexity. Antimirov [Ant96] introduced the notion of *partial derivatives* of a regular expression α, and proved that its cardinality is bounded by the number of alphabetic symbols that occur in α. The set of the partial derivatives can be used to construct a small NFA equivalent to α. We would like to improve our method using that idea.

References

[AM94] V. M. Antimirov and P. D. Mosses. Rewriting extended regular expressions. In G. Rozenberg and A. Salomaa, editors, *Developments in Language Theory*, pages 195 – 209. World Scientific, 1994.

[AMR07] M. Almeida, N. Moreira, and R. Reis. On the performance of automata minimization algorithms. Technical Report DCC-2007-03, DCC - FC & LIACC, Universidade do Porto, June 2007.

[Ant96] V. M. Antimirov. Partial derivatives of regular expressions and finite automaton constructions. *Theor. Comput. Sci.*, 155(2):291–319, 1996.

[Brz64] J. A. Brzozowski. Derivatives of regular expressions. *Journal of the Association for Computing Machinery*, 11(4):481–494, October 1964.

[Glu61] V. M. Glushkov. The abstract theory of automata. *Russian Math. Surveys*, 16:1–53, 1961.

[HMU00] J. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison Wesley, 2000.

[IY03] L. Ilie and S. Yu. Follow automata. *Inf. Comput.*, 186(1):140–162, 2003.

[Koz97] D. C. Kozen. *Automata and Computability.* Undergrad. Texts in Computer Science. Springer-Verlag, 1997.

[KS86] W. Kuich and A. Salomaa. *Semirings, Automata, Languages*, volume 5. Springer–Verlag, 1986.

[Mai94] H. G. Mairson. Generating words in a context-free language uniformly at random. *Information Processing Letters*, 49:95–99, 1994.

[RMA05] R. Reis, N. Moreira, and M. Almeida. On the representation of finite automata. In C. Mereghetti, B. Palano, G. Pighizzini, and D.Wotschke, editors, *Proc. of DCFS'05*, pages 269–276, Como, Italy, 2005.

[Sal66] A. Salomaa. Two complete axiom systems for the algebra of regular events. *Journal of the Association for Computing Machinery*, 13(1):158–169, January 1966.

[Sal69] A. Salomaa. *Theory of Automata*, volume 100. Pergamon Press, first edition, 1969.

[Sha04] J. Shallit. Regular expressions, enumeration and state complexity. Invited talk at CIAA 2004, 9th International Conference on Implementation and Application of Automata, 2004.

[SM73] L.J. Stockmeyer and A. R. Meyer. Word problems requiring exponential time: Preliminary report. In *Conf. Record of 5th Annual ACM Symposium on Theory of Computing, Austin, Texas, USA*, pages 1–9. ACM, 1973.

[Yu97] S. Yu. Regular languages. In G. Rozenberg and A. Salomaa, editors, *Handbook of Formal Languages*, volume 1. Springer-Verlag, 1997.