Antimirov and Mosses's Rewrite System Revisited*

Marco Almeida, Nelma Moreira, and Rogério Reis

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

Abstract. Antimirov and Mosses proposed a rewrite system for deciding the equivalence of two (extended) regular expressions. In this paper we present a functional approach to that method, prove its correctness, and give some experimental comparative results. Besides an improved version of Antimirov and Mosses's algorithm, we present a version using partial derivatives. Our preliminary results lead to the conclusion that, indeed, these methods are feasible and, generally, faster than the classical methods.

Keywords: regular languages, regular expressions, derivatives, partial derivatives, regular expression equivalence, minimal automata, rewriting systems.

1 Introduction

Although, because of their efficiency, finite automata are normally used for regular language manipulation, regular expressions (*re*) provide a particularly good notation for their representation. The problem of deciding whether two *re* are equivalent is PSPACE-complete [SM73]. This is normally solved by transforming each *re* into an equivalent NFA, convert those automata to equivalent deterministic ones, and finally minimize both DFAs, and decide if the resulting automata are isomorphic. The worst case complexity of the automata determinization process is exponential in the number of states.

Antimirov and Mosses [AM94] presented a rewrite system for deciding the equivalence of extended *re* based on a new complete axiomatization of the extended algebra of regular sets. This axiomatization, or any other classical complete axiomatization of the algebra of regular sets, can be used to construct an algorithm for deciding the equivalence of two *re*. Normally, however, these deduction systems are quite inefficient. This rewrite system is a refutation method that normalizes regular expressions in such a way that testing their equivalence corresponds to an iterated process of testing the equivalence of their derivatives. Termination is ensured because the set of derivatives to be considered is finite,

* This work was partially funded by Fundação para a Ciência e Tecnologia (FCT) and Program POSI, and by project ASA (PTDC/MAT/65481/2006).

O.H. Ibarra and B. Ravikumar (Eds.): CIAA 2008, LNCS 5148, pp. 46–56, 2008.

and possible cycles are detected using *memoization*. 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 DFAs. In this paper we present a functional approach to that method, prove its correctness, and give some experimental comparative results. Besides an improved version of Antimirov and Mosses's algorithm, we present a new version using partial derivatives. Our preliminary results lead to the conclusion that indeed these methods are feasible and, quite often, faster than the classical methods.

The paper is organized as follows. Section 2 contains several basic definitions and facts concerning regular languages and *re*. In Section 3 we present our variant of Antimirov and Mosses's method for testing the equivalence of two *re*. An improved version using partial derivatives is also presented. Section 4 gives some experimental comparative results between classical methods and the one presented in Section 3. Finally, in Section 5 we discuss some open problems, as ongoing and future work.

2 Regular Expressions and Automata

Here we recall 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].

Let Σ be an alphabet and $\Sigma^\star$ be the set of all *words* over Σ. The *empty word* is denoted by ϵ and the length of a word w is denoted by $|w|$. A language is a subset of $\Sigma^\star$, and if L_1 and L_2 are two languages, then $L_1 \cdot L_2 = \{xy \mid x \in L_1 \text{ and } y \in L_2\}$. A *re* α over Σ represents a (regular) language $L(\alpha) \subseteq \Sigma^\star$ and is inductively defined by: $\emptyset$ is a *re* and $L(\emptyset) = \emptyset$; ϵ is a *re* and $L(\epsilon) = \{\epsilon\}$; $a \in \Sigma$ is a *re* and $L(a) = \{a\}$; if α and β are *re*, $(\alpha + \beta)$, $(\alpha \cdot \beta)$ and $(\alpha)^\star$ are *re*, respectively with $L((\alpha + \beta)) = L(\alpha) \cup L(\beta)$, $L((\alpha \cdot \beta)) = (L(\alpha) \cdot L(\beta))$ and $L((\alpha)^\star) = L(\alpha)^\star$. The operator $\cdot$ is often omitted. We adopt the usual convention that $\star$ has precedence over $\cdot$, and $\cdot$ has higher priority than $+$. Let RE be the set of *re* over Σ. The size of α is denoted by $|\alpha|$ and represents the number of symbols, operators, and parentheses in α. We denote by $|\alpha|_\Sigma$ the number of symbols in α. We define the *constant part* of α as $\varepsilon(\alpha) = \epsilon$ if $\epsilon \in L(\alpha)$, and $\varepsilon(\alpha) = \emptyset$ otherwise. Two *re* α and β are *equivalent*, and we write $\alpha \sim \beta$, if $L(\alpha) = L(\beta)$. The algebraic structure $(RE, +, \cdot, \emptyset, \epsilon)$, constitutes an idempotent semi-ring, and, with the unary operator $\star$, a Kleene algebra. There are several well-known complete axiomatizations of Kleene algebras [Sal66, Koz94], but we will essentially consider Salomaa's axiom system F_1 which, besides the usual axioms for an idempotent semi-ring, contains the following two axioms for the $\star$ operator:

$$\alpha^\star \sim \epsilon + \alpha\alpha^\star; \;\; \alpha^\star \sim (\epsilon + \alpha)^\star.$$

As for rules of inference, system F_1 has the usual rule of substitution and the following rule of *solution of equations*:

$$\frac{\alpha \sim \beta\alpha + \gamma, \quad \varepsilon(\beta) = \emptyset}{\alpha \sim \beta^\star\gamma} \qquad (R_{se})$$

A *nondeterministic finite automaton* (NFA) A is a tuple $(Q, \Sigma, \delta, q_0, F)$ where Q is the finite set of states, Σ is the alphabet, $\delta \subseteq Q \times \Sigma \cup \{\epsilon\} \times Q$ the transition relation, q_0 the initial state and $F \subseteq Q$ the set of final states. A NFA without ϵ-transitions is *deterministic* (DFA) if, for each pair $(q, a) \in Q \times \Sigma$ there exists at most one q' such that $(q, a, q') \in \delta$. Two NFA are *equivalent* if they accept the same language. A DFA is called *minimal* if there is no equivalent DFA with fewer states. Minimal DFAs are unique up to isomorphism. DFAs, NFAs, and *re* represent the same set of languages, *i.e.*, regular languages.

2.1 Succinct Regular Expressions

Equivalent *re* do not need to have the same size. *Irreducible re* as defined by Ellul *et.al* [ESW02] have no redundant occurrences of $\emptyset$, ϵ, $\star$, and parentheses. A *re* α is *uncollapsible* if **none** 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 $\epsilon \in L(\gamma)$.

A *re* α is *irreducible* if it is uncollapsible and **both** two conditions are true:

- α does not contain superfluous parentheses (we adopt the usual operator precedence conventions and omit outer parentheses);
- α does not contain a sub-expression of the form $\beta^{\star^\star}$.

The previous reductions rely on considering *re modulo* some algebraic properties of *re*: identity elements of $+$ and ., annihilator element for $\cdot$, and idempotence of $\star$. We also consider $\emptyset^\star = \epsilon$ and $\epsilon^\star = \epsilon$.

Let ACI be the set of axioms that includes the associativity, commutativity and idempotence of disjunction and let $ACIA$ be the set ACI plus the associativity of concatenation. In this work, besides where otherwise stated, we consider irreducible regular expressions modulo $ACIA$ (and denote them by RE). This allows a more succinct representation of *re*, and is essential for ensuring the termination of the algorithms described in the next section.

Our implementation of *re* follows the object-oriented model. We use a different class for each operator which assures that the *re* are kept irreducible modulo $ACIA$, while trying to make the overhead of these transformations negligible. ACI properties are ensured by representing disjunctions as sets, that are coded using hash tables. This allows for a very efficient way of ensuring idempotence (as repeated elements result in a hash value clash), prohibiting $\emptyset$ as an argument, and rending the use of parentheses needless. In a similar way, concatenations are implemented with ordered lists, and the idempotence of the Kleene star is assured by not allowing double stared *re* in the constructor.

2.2 Linear Regular Expressions

A *re* α is *linear* if it is of the form $a_1\alpha_1 + \cdots + a_n\alpha_n$ for $a_i \in \Sigma$ and $\alpha_i \in RE$. The set of all the linear *re* is denoted by RE_{lin}, and can be defined by the

following (abstract syntax) context-free grammar, where A is the initial symbol, $L(B) = RE - \{\epsilon, \emptyset\}$ and $L(C) = \Sigma$:

$$A \to C \mid C \cdot B \mid A + A. \tag{G_1}$$

We say that an expression $a\beta$ has *head* $a \in \Sigma$ and *tail* β. We denote by $\mathrm{head}(\alpha)$ and $\mathrm{tail}(\alpha)$, respectively, the multiset of all heads and the multiset of all tails in a linear *re* α. A linear regular expression α is *deterministic* if no element of $\mathrm{head}(\alpha)$ occurs more than once. We denote the set of all deterministic linear *re* by RE_{det}. Every *re* α can be written as a disjunction of its constant part and a (deterministic) linear *re* [Sal66]. A *re* is said to *pre-linear* if it belongs to the language generated by the following context-free grammar (abstract syntax) with initial symbol A', and A and B are as in G_1:

$$\begin{aligned} A' &\to \emptyset \mid D \\ D &\to A \mid D \cdot B \mid (D + D). \end{aligned} \tag{G_2}$$

The set of all pre-linear *re* is denoted by RE_{plin}.

2.3 Derivatives

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

$$\begin{aligned} &a^{-1}(\emptyset) = \emptyset; && a^{-1}(\alpha + \beta) = a^{-1}(\alpha) + a^{-1}(\beta); \\ &a^{-1}(\epsilon) = \emptyset; && a^{-1}(\alpha\beta) = a^{-1}(\alpha)\beta + \varepsilon(\alpha)a^{-1}(\beta); \\ &a^{-1}(b) = \begin{cases} \epsilon, & \text{if } b = a; \\ \emptyset, & \text{otherwise;} \end{cases} && a^{-1}(\alpha^\star) = a^{-1}(\alpha)\alpha^\star. \end{aligned}$$

If α is a deterministic linear *re*, we have:

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

The *derivative* of a *re* α with respect to the *word* $w \in \Sigma^\star$, denoted $w^{-1}(\alpha)$, is defined recursively on the structure of w:

$$\epsilon^{-1}(\alpha) = \alpha; \quad (ua)^{-1}(\alpha)a^{-1}(u^{-1}(\alpha)), \text{ for any } u \in \Sigma^\star.$$

Considering *re* modulo the *ACI* axioms, Brzozowski [Brz64] proved that, for every *re* α, the set of its derivatives with respect to any word w is finite.

3 Regular Expression Equivalence

The classical approach to the problem of comparing two *re* α and β, *i.e.*, deciding if $L(\alpha) = L(\beta)$, typically consists of transforming each *re* into an equivalent NFA,

convert those automata to equivalent deterministic ones, and minimize both DFAs. Because, for a given regular language, the minimal DFA is unique up to isomorphism, these can be compared using a canonical representation [RMA05], and thus checked if $L(\alpha) = L(\beta)$. In this section, we present two methods to verify the equivalence of two *re*. The first 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 *re* without the construction of the canonical minimal automata. It is a functional approach on which we always consider the *re* to be irreducible and not extended (with intersection). The use of irreducible *re* allows us to avoid the simplification step of Antimirov and Mosses's system with little overhead. The second method improves this first one by using the notion of partial derivative.

3.1 Regular Expression's Linearization

Let $a \in \Sigma$, and α, β, γ be arbitrary *re*. We define the functions $\mathrm{lin} = \mathrm{lin}_2 \circ \mathrm{lin}_1$, and det as follows:

$$
\begin{aligned}
\mathrm{lin}_1 : RE &\to RE_{plin} \\
\mathrm{lin}_1(\emptyset) &= \emptyset; \\
\mathrm{lin}_1(\epsilon) &= \emptyset; \\
\mathrm{lin}_1(a) &= a; \\
\mathrm{lin}_1(\alpha+\beta) &= \mathrm{lin}_1(\alpha) + \mathrm{lin}_1(\beta); \\
\mathrm{lin}_1(\alpha^\star) &= \mathrm{lin}_1(\alpha)\alpha^\star; \\
\mathrm{lin}_1(a\alpha) &= a\alpha; \\
\mathrm{lin}_1((\alpha+\beta)\gamma) &= \mathrm{lin}_1(\alpha\gamma) + \mathrm{lin}_1(\beta\gamma); \\
\mathrm{lin}_1(\alpha^\star\beta) &= \mathrm{lin}_1(\alpha)\alpha^\star\beta + \mathrm{lin}_1(\beta).
\end{aligned}
\qquad
\begin{aligned}
\mathrm{lin}_2 : RE_{plin} &\to RE_{lin} \cup \{\emptyset\} \\
\mathrm{lin}_2(\alpha+\beta) &= \mathrm{lin}_2(\alpha) + \mathrm{lin}_2(\beta); \\
\mathrm{lin}_2((\alpha+\beta)\gamma) &= \mathrm{lin}_2(\alpha\gamma) + \mathrm{lin}_2(\beta\gamma); \\
\mathrm{lin}_2(\alpha) &= \alpha. \quad \text{(Otherwise)}
\end{aligned}
$$

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

The functions lin and lf linearize regular expressions. Function lin_1 corresponds to the function f of the original rewrite system which, contrary to what is claimed by Antimirov and Mosses, returns a pre-linear *re*, and not a linear one. We use the function lin for efficiency reasons because in a single pass returns a *re* in a form such that the derivative w.r.t. *any* symbol of the alphabet is readily available. To show that $\mathrm{lin}(\alpha)$ returns either the linear part of α or $\emptyset$, it is enough to observe the following facts, which have straightforward proofs that can be found, along with all the missing proofs, in an extended version of the present paper.

- The function lin_1 is well defined.
- For $\alpha \in RE$, $\mathrm{lin}_1(\alpha) \in L(G_2)$.
- For $\alpha \in RE_{plin}$, $\alpha \sim \mathrm{lin}_2(\alpha)$.
- For $\alpha \in RE$, $\mathrm{lin}(\alpha) \in L(G_1) \cup \{\emptyset\}$.
- For $\alpha \in RE_{lin} \cup \{\emptyset\}$, $\det(\alpha) \in RE_{det}$ and $\alpha \sim \det(\alpha)$.
- For $\alpha \in RE$, $L(\mathrm{lin}(\alpha)) = \begin{cases} L(\alpha), & \text{if } \epsilon \notin L(\alpha); \\ L(\alpha) - \{\epsilon\}, & \text{if } \epsilon \in L(\alpha). \end{cases}$

Thus we have:

Theorem 1. *For any re* α, $\alpha \sim \varepsilon(\alpha) + \mathrm{lin}(\alpha)$, *and* $\alpha \sim \varepsilon(\alpha) + \det(\mathrm{lin}(\alpha))$.

Considering the definition of derivative (Subsection 2.3), we also have:

Theorem 2. *Let* $a \in \Sigma$ *and* $\alpha \in RE$, *then* $a^{-1}(\alpha) = a^{-1}(\det(\mathrm{lin}(\alpha)))$.

3.2 Regular Expression Equivalence

We now present the main functions of the comparison processes. The first one, the function derivatives, computes the set of the derivatives of a pair of deterministic linear *re* (α, β), with respect to each symbol of the alphabet. It is enough to consider only the symbols in $\mathrm{head}(\alpha) \cup \mathrm{head}(\beta)$, and we do that for efficiency reasons. The function is, then, defined as follows:

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

The equiv function, applied to two *re* α and β, returns $True$ if and only if $\alpha \sim \beta$. It is defined in the following way:

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

where

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

In each step the function equiv proceeds by rewriting a pair of *re* into a set S of pairs of derivatives. When either a disagreement pair is found, *i.e.*, a pair of derivatives such that their constant parts are different, or the set S is empty, the function returns. If $\alpha \sim \beta$ the call $\text{equiv}(\{(\alpha, \beta)\})$ returns the value $True$, otherwise returns $False$. Comparing with the Antimirov and Mosses's rewrite system TR, we note that in each call to $\text{equiv}(S, H)$, the set S contains only pairs of *re* which are not in H, thus rendering the rule (IND) of TR unnecessary. On the other hand, our data structures avoid the need of the rule (SIM) by assuring that the *re* are always irreducible.

Theorem 3. *The function* equiv *is terminating.*

Lemma 1. *Given* $\alpha, \beta \in RE_{det} \cup \{\emptyset\}$, $\forall (\alpha', \beta') \in \text{derivatives}(\alpha, \beta)$, $\alpha \sim \beta \Rightarrow \alpha' \sim \beta'$.

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

3.3 Improved Equivalence Method Using Partial Derivatives

Antimirov [Ant96] introduced the notion of the partial derivatives set of a regular expression α and proved that its cardinality is bounded by the number of alphabetic symbols that occurs in α. He showed that this set can be obtained directly from a new linearization process of α. This new process can be easily implemented in our approach, as a variant of lin function, as we already consider disjunctions as sets. We now briefly review this notions and show how they can be used to improve the equiv algorithm.

Linear Forms. Let $\Sigma \times RE$ be the set of *monomials* over an alphabet Σ. Let $\mathcal{P}_{fin}(A)$ be the set of all finite parts of A. A linear *re* $a_1 \cdot \alpha_1 + \cdots + a_n \cdot \alpha_n$ can be represented by a finite set of monomials $l \in \mathcal{P}_{fin}(\Sigma \times RE)$, named *linear form*, and such that $l = \{(a_1, \alpha_1), \ldots, (a_n, \alpha_n)\}$. We define a function $\sum : \mathcal{P}_{fin}(\Sigma \times RE) \to RE_{lin}$ by $\sum(l) = a_1 \cdot \alpha_1 + \cdots + a_n \cdot \alpha_n$. Concatenation of a linear form l with a *re* β is defined by $l \cdot \beta = \{(a_1, \alpha_1 \cdot \beta), \ldots, (a_n, \alpha_n \cdot \beta)\}$. We can now define the linearization of a *re* α into a linear form as follows:

$$
\begin{aligned}
&\mathrm{lf} : RE \to \mathcal{P}_{fin}(\Sigma \times RE) \\
&\mathrm{lf}(\emptyset) = \emptyset; && \mathrm{lf}(\alpha^\star) = \mathrm{lf}(\alpha) \cdot \alpha^\star; \\
&\mathrm{lf}(\epsilon) = \emptyset; && \mathrm{lf}(a \cdot \alpha) = \{(a, \alpha)\}; \\
&\mathrm{lf}(a) = \{(a, \epsilon)\}; && \mathrm{lf}((\alpha + \beta) \cdot \gamma) = \mathrm{lf}(\alpha \cdot \gamma) \cup \mathrm{lf}(\beta \cdot \gamma); \\
&\mathrm{lf}(\alpha + \beta) = \mathrm{lf}(\alpha) \cup \mathrm{lf}(\beta); && \mathrm{lf}(\alpha^\star \cdot \beta) = \mathrm{lf}(\alpha) \cdot \alpha^\star \cdot \beta \cup \mathrm{lf}(\beta).
\end{aligned}
$$

The following theorem relates the method of linearization presented in the Section 2.2 with linear forms.

Theorem 5. *For any re* α, $\mathrm{lin}(\alpha) = \sum(lf(\alpha))$.

Partial Derivatives. Given a *re* α and a symbol $a \in \Sigma$, a *partial derivative* of α w.r.t. a is a *re* ρ such that $(a, \rho) \in \mathrm{lf}(\alpha)$. The set of partial derivatives of α w.r.t. a is denoted by $\partial_a(\alpha)$. The notion of partial derivative of α can be extended to words $w \in \Sigma^\star$, sets of *re* $R \subseteq RE$, and sets of words $W \subseteq \Sigma^\star$, as follows:

$$
\begin{aligned}
&\partial_\epsilon(\alpha) = \{\alpha\}; && \partial_w(R) = \bigcup_{\alpha \in R} \partial_w(\alpha); \\
&\partial_{ua}(\alpha) = \partial_a(\partial_u(\alpha)), \text{ for any } u \in \Sigma^\star; && \partial_W(\alpha) = \bigcup_{w \in W} \partial_w(\alpha).
\end{aligned}
$$

There is a strong connection between the sets of partial derivatives and the derivatives of a *re*. Trivially extending the notion of language represented by a *re* to sets of *re*, we have that $L(\partial_w(\alpha)) = L(w^{-1}(\alpha))$, for any $w \in \Sigma^\star$, $\alpha \in RE$. One of the advantages of using partial derivatives is that for any $\alpha \in RE$, $|PD(\alpha) = \partial_{\Sigma^\star}(\alpha)| \leq |\alpha|_\Sigma$, where $PD(\alpha)$ stands for the set of all the syntactically different partial derivatives.

Improving equiv by Using Partial Derivatives. Let us now consider a determinization process for linear forms. We say that a linear form is deterministic if, for each symbol $a \in \Sigma$, there is at most one element of the form (a, α). Let lfX be an extended version of the linearization function lf, defined as follows:

$$\mathrm{lfX}(\alpha) = \{(a, \sum_{(a,\alpha') \in \mathrm{lf}(\alpha)} \alpha') \mid a \in \Sigma\}.$$

We can replace the function composition $\det \cdot \mathrm{lin}$ with the deterministic linear form obtained with lfX. This new extended linear form allows us to use the previously defined equiv function with only two slight modifications. We redefine the derivatives function as follows:

$$\begin{aligned} \text{derivatives} \quad &: \quad \mathcal{P}^{det}_{fin}(\Sigma \times RE) \times \mathcal{P}^{det}_{fin}(\Sigma \times RE) \to \mathcal{P}(RE \times RE) \\ \text{derivatives}(\alpha, \beta) &= \{(\alpha', \beta') \mid (a, \alpha') \in \alpha, (a, \beta') \in \beta\}. \end{aligned}$$

In the definition of equiv, we change $\alpha' = \mathrm{lfX}(\alpha)$ and $\beta' = \mathrm{lfX}(\beta)$. The new function will be called equivP in the next section.

4 Experimental Results

We will now present some experimental results. These are the running times for the two methods for checking the equivalence of regular expressions. One uses the equivalent minimal DFA, the other is the direct *re* comparison method, as described on the Section 3. All tests were performed on batches of $10,000$ pairs of uniformly random generated *re*, and the running times do not include the time necessary to parse each *re*. Each batch contains *re* of size 10, 50 or 100, with either 2, 5 or 10 symbols. For the uniform generation of random *re* we implemented the method described by Mairson [Mai94] for the generation of context-free languages. We used a grammar for almost irreducible *re* presented by Shallit [Sha04]. As the data sets were obtained with a uniform random generator, the size of each sample is sufficient to ensure a 95% confidence level within a 1% error margin. It is calculated with the formula $n = (\frac{z}{2\epsilon})^2$, where z is obtained from the normal distribution table such that $P(-z < Z < z)) = \gamma$, ϵ is the error margin, and γ is the desired confidence level.

We tested the equivalence of each pair of *re* using both the classical approach and the direct comparison method. We used Glushkov's algorithm to obtain the NFAs from the *re*, and the well-known subset construction to make each NFA deterministic. As for the DFA minimization 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 minimizing 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 both with (**equivP**) and without partial derivatives

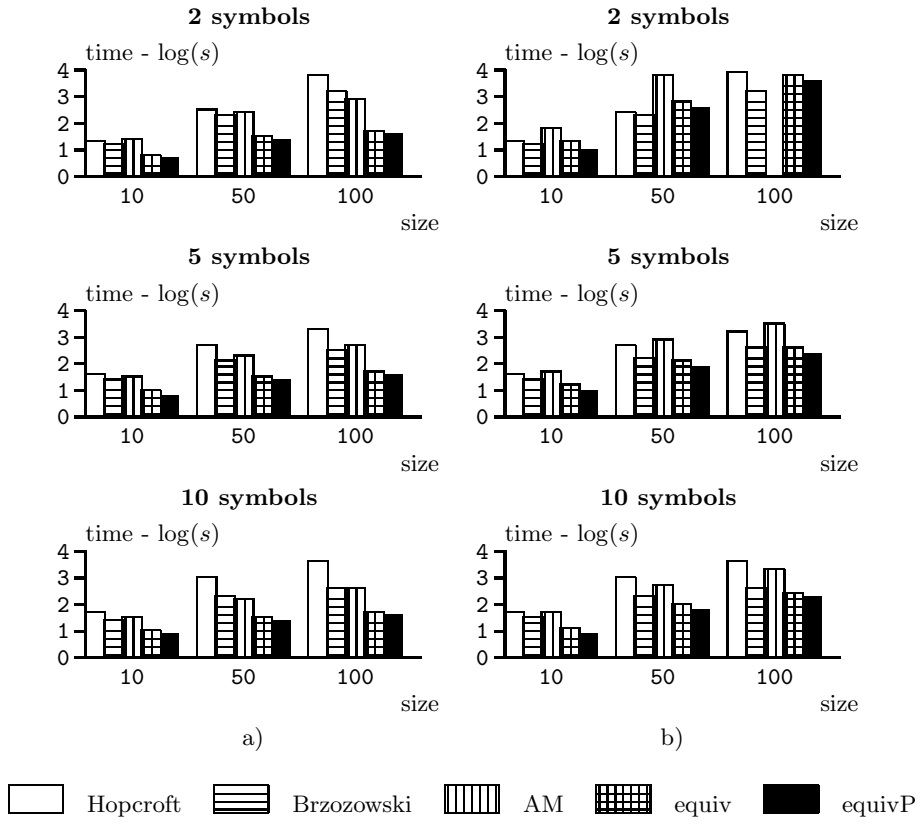

Fig. 1. Running times of three different methods for checking the equivalence of *re*. a) 10.000 pairs of random *re*; b) 10.000 pairs of syntactically equal random *re*. The missing column corresponds to a larger than reasonable observed runtime.

(**equiv**). Because the direct comparison methods try to compute a refutation, we performed a set of tests for the worst case scenario of these algorithms: the equivalence of two syntactically equal regular expressions.

As shown in Figure 1 (a), when comparing randomly generated *re*, any of the direct methods is always the fastest. Note also that Hopcroft's algorithm never achieves shorter running times than Brzozowski's. Figure 1 (b) shows the results of the application of each algorithm to pairs of syntactically equal random *re*. Except for the samples of *re* with size 50 or 100, over an alphabet of 2 symbols, the direct *re* comparison methods are still the fastest. Again, Brzozowki's algorithm always presents better running times than Hopcroft's. Among the direct comparison methods, **equivP** always performs better, with a speedup of $20\% - 30\%$. It is important to state that, asymptotically, when using the minimal DFA approach, the minimization algorithm is the bottleneck of the entire process. It always takes over 50% of the total amount of time when the size of the *re* and/or the alphabet grows . To ensure the fairness of the comparison for

the method using NFAs, we tried several algorithms for computing (small) NFAs from *re* (c.f Ilie and Yu [IY03]), but the size of the NFAs seems not to affect significantly the overall performance.

5 Conclusion

We presented a variant method based on a rewrite system for testing the equivalence of two *re*, that attempts to refute its equivalence by finding a pair of derivatives that disagree in their constant parts. While a good behaviour was expected for some non-equivalent *re*, experimental results point to a good average-case performance for this method, even when feeded with equivalent *re*. Some improvement was also achieved by using partial derivatives. Given the spread of multi-cores and grid computer systems, a parallel execution of the better behavioured classic method and our direct comparison method can lead to an optimized framework for testing *re* equivalence. A better theoretical understanding of relationships between the two approaches would be helpful towards the characterization of their average-case complexity. In particular, it will be interesting to compare our method with the one by Hopcroft and Karp [HK71].

References

[AM94] Antimirov, V.M., Mosses, P.D.: Rewriting extended regular expressions. In: Rozenberg, G., Salomaa, A. (eds.) Developments in Language Theory, pp. 195–209. World Scientific, Singapore (1994)

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

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

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

[ESW02] Ellul, K., Shallit, J., Wang, M.: Regular expressions: New results and open problems. Talk at the DCFS 2002 conference, London, Ontario (2002)

[HK71] Hopcroft, J., Karp, R.M.: A linear algorithm for testing equivalence of finite automata. Technical Report TR 71 -114, University of California, Berkeley, California (1971)

[HMU00] Hopcroft, J., Motwani, R., Ullman, J.D.: Introduction to Automata Theory, Languages and Computation. Addison-Wesley, Reading (2000)

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

[Koz94] Kozen, D.C.: A completeness theorem for Kleene algebras and the algebra of regular events. Infor. and Comput. 110(2), 366–390 (1994)

[Koz97] Kozen, D.C.: Automata and Computability. Undergrad. Texts in Computer Science. Springer, Heidelberg (1997)

[KS86] Kuich, W., Salomaa, A.: Semirings, Automata, Languages, vol. 5. Springer, Heidelberg (1986)

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

[RMA05] Reis, R., Moreira, N., Almeida, M.: On the representation of finite automata. In: Mereghetti, C., Palano, B., Pighizzini, G., Wotschke, D. (eds.) Proc. of DCFS 2005, Como, Italy, pp. 269–276 (2005)

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

[Sha04] Shallit, J.: Regular expressions, enumeration and state complexity. In: Domaratzki, M., Okhotin, A., Salomaa, K., Yu, S. (eds.) CIAA 2004. LNCS, vol. 3317. Springer, Heidelberg (2005)

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