

# Partial Derivatives of Regular Expressions and Finite Automata Constructions

Valentin Antimirov

CRIN (CNRS) & INRIA-Lorraine,  
BP 239, F54506, Vandœuvre-lès-Nancy Cedex, FRANCE  
e-mail: `Valentin.Antimirov@loria.fr`

**Abstract.** We introduce a notion of a *partial derivative* of a regular expression. It is a generalization to the non-deterministic case of the known notion of a *derivative* invented by Brzozowski. We give a constructive definition of partial derivatives, study their properties, and employ them to develop a new algorithm for turning regular expressions into relatively small NFA and to provide certain improvements to Brzozowski's algorithm constructing DFA. We report on a prototype implementation of our algorithm constructing NFA and present some examples.

## Introduction

In 1964 Janusz Brzozowski introduced *word derivatives* of regular expressions and suggested an elegant algorithm turning a regular expression  $r$  into a deterministic finite automata (DFA); the main point of the algorithm is that the word derivatives of  $r$  serve as states of the resulting DFA [5].

In the following years derivatives were recognized as a quite useful and productive tool. Conway [8] uses derivatives to present various computational procedures in the algebra of regular expressions and to investigate some logical properties of this algebra. Krob [12] extends this differential calculus to a more general algebra of  $K$ -rational expressions. Brzozowski and Leiss [6] employ the idea of derivatives to ascertain relations between regular expressions, finite automata and boolean networks. Berry and Sethi [4] give a solid theoretical background for McNaughton and Yamada's algorithm [14] through the notion of *continuation* which is a particular kind of derivative. Ginzburg [9] uses derivatives to develop a procedure for proving equivalence of regular expressions; further development of this procedure is provided by Mizoguchi *et al* [15]. Yet another procedure for proving equivalence of extended regular expressions is suggested by the present author and Mosses [2, 3] and is also based on some constructions closely related to derivatives.

In the present paper we come up with a new notion of a *partial derivative* which is, in a sense, a non-deterministic generalization of the notion of derivative: likewise derivatives are related to DFA, partial derivatives are in the same natural way related to non-deterministic finite automata (NFA).

In Sect.2 we introduce partial derivatives which are regular expressions appearing as components of so-called *non-deterministic linear forms*. We give a set

of recursive equations for computing linear forms and partial derivatives. Some basic properties of partial derivatives are established.

In Sect.3 we present two theorems which are the main theoretical results of the paper. These show that the set of all syntactically distinct partial derivatives of any regular expression  $r$  is finite<sup>1</sup> and its cardinality is quite small – less than or equal to one plus the number of occurrences of alphabet letters appearing in  $r$ ; moreover, each partial derivative (and so the set of them) can be represented quite compactly.

Sect.4 is devoted to an application of the above theoretical results to finite automata constructions. First, we present a new top-down algorithm for turning a regular expression  $r$  into an NFA; the set of partial derivatives of  $r$  forms the set of states of the NFA. This implies that the above upper bound for the cardinality of the set of partial derivatives holds as an upper bound for the number of states of NFA produced by our algorithm. This upper bound coincides with the number of states of NFA produced by McNaughton and Yamada’s algorithm [14] and by its improvement due to Berry and Sethi [4]. However, in many cases our NFA have actually fewer states than this upper bound. Moreover, there are examples when our NFA turn out to be smaller than those produced by a tricky Chang and Paige’s algorithm [7] (which involves several non-trivial optimizations of the representation of NFA). The second procedure in this section provides several improvements to Brzozowski’s algorithm [5] due to the use of partial derivatives.

We have implemented our algorithm turning regular expressions into NFA as an algebraic program in OBJ3 (see [11] for a description of the language). In Sect.5 we present several examples of NFA constructed by our program.

In this version of the paper we omit all the proofs as well as some auxiliary propositions and examples; see [1] for a more complete presentation.

## 1 Preliminaries

Given a set  $X$ , we denote its cardinality by  $|X|$ , its powerset (the set of all subsets of  $X$ ) by  $\mathcal{P}(X)$ , and the set of all finite subsets of  $X$  by  $\mathbf{Set}[X]$ .

An *idempotent semiring* is an algebra on the signature including constants  $\emptyset$  (called zero),  $\lambda$  (unit) and binary operations  $+$  (union or join) and  $\cdot$  (concatenation) such that these satisfy the following equational axioms:

$$a + (b + c) = (a + b) + c \tag{1}$$

$$a + b = b + a \tag{2}$$

$$a + a = a \tag{3}$$

$$a + \emptyset = a \tag{4}$$

$$(a \cdot b) \cdot c = a \cdot (b \cdot c) \tag{5}$$

$$a \cdot \lambda = \lambda \cdot a = a \tag{6}$$

---

<sup>1</sup> Note that derivatives do not enjoy this property.

$$a \cdot \emptyset = \emptyset \cdot a = \emptyset \quad (7)$$

$$a \cdot (b + c) = a \cdot b + a \cdot c \quad (8)$$

$$(a + b) \cdot c = a \cdot c + b \cdot c \quad (9)$$

Thus, the algebra is simultaneously an *upper semilattice with the bottom* (w.r.t.  $\emptyset$  and  $+$ ) and a monoid (w.r.t.  $\lambda$  and  $\cdot$ ). We shall refer to the set of the equations (1–9) as to *SR-axioms*; the least congruence generated by these (on some appropriate algebra) will be called the *SR-congruence* and denoted by  $\mathcal{E}(SR)$ . Similarly, the set of equations (1–4) is called *ACIZ-axioms* and its subset (1–3) is called *ACI-axioms*; corresponding least congruences are denoted by  $\mathcal{E}(ACIZ)$  and  $\mathcal{E}(ACI)$ . Note that the set  $\mathbf{Set}[X]$  forms an upper semilattice with the join  $\cup$  and the empty set  $\emptyset$  as the bottom.

Given an alphabet (a finite set of letters)  $\mathcal{A}$ , let  $\mathcal{A}^*$  be the set (and the free monoid) of words on  $\mathcal{A}$ ,  $\mathbf{Reg}[\mathcal{A}]$  be the set (and the algebra) of *regular* (or *rational*) languages on the alphabet  $\mathcal{A}$  with the standard operations – concatenation  $L_1 \cdot L_2$ , union  $L_1 \cup L_2$ , and iteration  $L^*$ . Let  $\mathbf{Reg1}[\mathcal{A}]$  be the subset of  $\mathbf{Reg}[\mathcal{A}]$  consisting of all the regular languages containing the empty word  $\lambda$ ; the complement of this subset is denoted by  $\mathbf{Reg0}[\mathcal{A}]$ . Let  $\mathcal{A}^+ = \mathcal{A} \cdot \mathcal{A}^*$ .

Given a regular language  $L$ , a *left quotient of  $L$  w.r.t. a word  $w$* , written  $w \setminus L$ , is the language  $\{ u \in \mathcal{A}^* \mid w \cdot u \in L \}$ . Note that the membership  $w \in L$  is equivalent to  $\lambda \in w \setminus L$  and that  $(w \cdot x) \setminus L = x \setminus (w \setminus L)$ .

We consider a (*non-deterministic*) *finite automaton*  $\mathbf{M}$  on  $\mathcal{A}$  as a quadruple  $\langle M, \tau, \mu_0, F \rangle$  where  $M$  is a set of states,  $\tau : M \times \mathcal{A} \rightarrow \mathbf{Set}[M]$  is a transition function (which can also be represented as a relation  $\tau \subset M \times \mathcal{A} \times M$ ),  $\mu_0 \in M$  is an initial state, and  $F \subset M$  is a set of final states.<sup>2</sup> The automaton is *deterministic* if  $|\tau(\mu, x)| \leq 1$  for all  $\mu \in M, x \in \mathcal{A}$ ; in this case the transition function is represented as a partial one,  $\tau : M \times \mathcal{A} \dashrightarrow M$ , returning a state or nothing. The function can always be completed to a total one by adding one “sink” state  $\emptyset$  to  $M$  such that  $\tau(\emptyset, x) = \emptyset$  for all  $x \in \mathcal{A}$ . In any of the above cases, the extension  $\tau(\mu, w)$  of the transition function to words  $w \in \mathcal{A}^*$  is defined in the usual way. A word  $w \in \mathcal{A}^*$  is said to be accepted by a state  $\mu \in M$  if  $\tau(\mu, w) \cap F \neq \emptyset$ . The set of all words accepted by  $\mu_0$  is the language *recognized* by  $\mathbf{M}$ . Two automata are *equivalent* if they recognize the same language.

*Regular* (or *rational*) expressions are terms on the signature of the regular algebra  $\mathbf{Reg}[\mathcal{A}]$ . Actually, there exist different ways to choose the signature and to formalize the algebra. In this paper we follow the idea of [2] that  $\mathbf{Reg}[\mathcal{A}]$  should be regarded as an *order-sorted* algebra [10] having a sort  $\mathcal{A}$  for an alphabet which is a subsort of a sort *Reg* for all the regular expressions. Here we also introduce two further subsorts of *Reg*, namely *Reg0* and *Reg1*, to distinguish regular expressions denoting elements of  $\mathbf{Reg0}[\mathcal{A}]$  and  $\mathbf{Reg1}[\mathcal{A}]$  correspondingly. To sum up, the order-sorted signature *REG* on the alphabet  $\mathcal{A} = \{\alpha_1, \alpha_2, \dots, \alpha_k\}$  consists of the following components:<sup>3</sup>

<sup>2</sup> This is obviously not the most general definition, but we shall need just this particular kind of NFA.

<sup>3</sup> Argument places of operations are indicated by the underbar character “\_”.

<b>sorts</b>	$\mathcal{A}, \text{Reg}, \text{Reg0}, \text{Reg1}.$	
<b>subsorts</b>	$\mathcal{A} \leq \text{Reg0} \leq \text{Reg}, \text{Reg1} \leq \text{Reg}.$	
<b>constants</b>	$\emptyset : \text{Reg0}; \lambda : \text{Reg1}; \alpha_1, \alpha_2, \dots, \alpha_k : \mathcal{A}.$	
<b>operations</b>	$- + - : \text{Reg Reg} \rightarrow \text{Reg}$	$- \cdot - : \text{Reg Reg} \rightarrow \text{Reg}$
	$- + - : \text{Reg1 Reg} \rightarrow \text{Reg1}$	$- \cdot - : \text{Reg0 Reg} \rightarrow \text{Reg0}$
	$- + - : \text{Reg Reg1} \rightarrow \text{Reg1}$	$- \cdot - : \text{Reg Reg0} \rightarrow \text{Reg0}$
	$- + - : \text{Reg0 Reg0} \rightarrow \text{Reg0}$	$- \cdot - : \text{Reg1 Reg1} \rightarrow \text{Reg1}$
	$-^* : \text{Reg} \rightarrow \text{Reg1}.$	

Sets of ground terms on the signature  $REG$  of the sorts  $\text{Reg}$ ,  $\text{Reg0}$ , and  $\text{Reg1}$  are defined in the usual way and denoted by  $\mathcal{T}_{\text{Reg}}$ ,  $\mathcal{T}_{\text{Reg0}}$ , and  $\mathcal{T}_{\text{Reg1}}$  correspondingly. In what follows we call the elements of  $\mathcal{T}_{\text{Reg}}$  *regular terms*.

Given a regular term  $t$ , let  $\|t\|$  denote its *alphabetic width* – the number of all the occurrences of letters from the alphabet  $\mathcal{A}$  appearing in  $t$ .

A regular term  $t$  denotes a regular language  $\mathcal{L}(t)$ ; this interpretation is determined by the homomorphism  $\mathcal{L}(\_)$  from the absolutely free algebra of regular terms  $\mathcal{T}_{\text{Reg}}$  to  $\mathbf{Reg}[\mathcal{A}]$ . Let  $\mathcal{E}(\mathbf{Reg})$  be the kernel of this homomorphism, i.e. the congruence on  $\mathcal{T}_{\text{Reg}}$  consisting of all the pairs  $\langle t_1, t_2 \rangle$  such that  $\mathcal{L}(t_1) = \mathcal{L}(t_2)$ .

Recall that  $\mathbf{Reg}[\mathcal{A}]$  is an idempotent semiring, i.e. satisfies the SR-axioms (1–9). There are further axioms concerning Kleene star (see e.g. [8] or [16]).

Thus, we have the following chain of quotients of  $\mathcal{T}_{\text{Reg}}$  related by surjective homomorphisms:

$$\mathcal{T}_{\text{Reg}} \rightarrow \mathcal{T}_{\text{Reg}}/\mathcal{E}(\text{ACI}) \rightarrow \mathcal{T}_{\text{Reg}}/\mathcal{E}(\text{ACIZ}) \rightarrow \mathcal{T}_{\text{Reg}}/\mathcal{E}(\text{SR}) \rightarrow \mathcal{T}_{\text{Reg}}/\mathcal{E}(\mathbf{Reg}) \quad (10)$$

where the last quotient is isomorphic to  $\mathbf{Reg}[\mathcal{A}]$ .

A *constant part* of a regular term  $t$  is equal to  $o(t)$  where the function  $o(\_)$  is defined on  $\mathcal{T}_{\text{Reg}}$  as follows:  $o(t) = \mathbf{if } t \in \mathcal{T}_{\text{Reg1}} \mathbf{ then } \lambda \mathbf{ else } \emptyset \mathbf{ fi}$ .

**Definition 1. (Derivatives)** For any letter  $x \in \mathcal{A}$  and word  $w \in \mathcal{A}^*$  the functions  $x^{-1}(\_)$  and  $w^{-1}(\_)$  on  $\mathcal{T}_{\text{Reg}}$  computing (*word*) *derivatives* of regular terms are defined recursively by the following equations for all  $y \in \mathcal{A}$ ,  $a, b \in \mathcal{T}_{\text{Reg}}$  [5]:

$$\begin{aligned} x^{-1}\emptyset &= x^{-1}\lambda = \emptyset, & x^{-1}(a^*) &= (x^{-1}a) \cdot a^*, \\ x^{-1}y &= \mathbf{if } x = y \mathbf{ then } \lambda \mathbf{ else } \emptyset \mathbf{ fi}, & \lambda^{-1}a &= a, \\ x^{-1}(a + b) &= x^{-1}a + x^{-1}b, & (w \cdot x)^{-1}a &= x^{-1}(w^{-1}a). \\ x^{-1}(a \cdot b) &= (x^{-1}a) \cdot b + o(a) \cdot x^{-1}b, & & \square \end{aligned}$$

These equations are stable w.r.t. the congruences on  $\mathcal{T}_{\text{Reg}}$  mentioned in (10), therefore  $x^{-1}(\_)$  and  $w^{-1}(\_)$  are correctly defined on the corresponding quotients of  $\mathcal{T}_{\text{Reg}}$ . It is known that  $\mathcal{L}(w^{-1}r) = w \setminus \mathcal{L}(r)$ .

Given a set of equations  $E$  on the signature  $REG$ , two derivatives are said to be *E-similar* if they are equivalent modulo  $E$ . The set  $\mathcal{D}_E(t)$  of all *E-dissimilar* word derivatives of a regular term  $t$  is obtained as a set of representatives of the equivalence classes modulo  $E$  of terms  $w^{-1}t$  for all  $w \in \mathcal{A}^*$ . It was proved in [5] that the set  $\mathcal{D}_E(t)$  may be infinite for some regular term  $t$  when  $E = \emptyset$ , but it becomes finite (for all  $t$ ) as soon as  $E$  includes the ACI-axioms (1–3). In the latter case the following fact holds which presents Brzozowski's method for constructing DFA:

**Proposition 2.** *Given a regular term  $t$ , consider the DFA  $\mathbf{M}$  with the set of states  $M = \mathcal{D}_E(t)$ , the initial state  $\mu_0 = t$ , the transition function defined by  $\tau(r, x) = x^{-1}r$  for all  $x \in \mathcal{A}$ ,  $r \in M$ , and the set of final states  $F = \{r \in M \mid o(r) = \lambda\}$ . Then  $\mathbf{M}$  recognizes the language  $\mathcal{L}(t)$ .*

Note that to practically implement this construction, one needs to compute the set  $\mathcal{D}_E(t)$  that involves testing equivalence of regular expressions modulo  $E$ . Another technical problem is that for some  $t$  the cardinality of  $\mathcal{D}_E(t)$  is an exponent in the size of  $t$ .

## 2 Introducing Partial Derivatives

It is a folk knowledge that any regular expression  $r$  on an alphabet  $\mathcal{A} = \{x_1, \dots, x_n\}$  can be represented in the following “linear” form:

$$r = o(r) + x_1 \cdot r_1 + \dots + x_n \cdot r_n \quad (11)$$

where all the  $r_i$  are some regular expressions (see [5, 18, 8]). In particular, one can take each  $r_i$  to be the derivative  $x_i^{-1}r$ . We are going to generalize this linear factorization in several ways to make it *non-deterministic* in a sense; this will lead us to partial derivatives. First we need to introduce some auxiliary notions.

**Definition 3.** Let **SetReg** be the upper semilattice  $\mathbf{Set}[\mathcal{T}_{Reg} \setminus \{\emptyset\}]$  of finite sets of non-zero regular terms. We define a function  $\rho : \mathcal{T}_{Reg} \rightarrow \mathbf{SetReg}$  which satisfies the conditions  $\rho(\emptyset) = \emptyset$ ,  $\rho(t_1 + t_2) = \rho(t_1) \cup \rho(t_2)$  for all  $t_1, t_2 \in \mathcal{T}_{Reg}$ , and maps any other regular term  $t$  to the singleton  $\{t\}$ . We call  $\rho(t)$  a *set representation* of  $t$ . Two regular terms  $t_1, t_2$  are said to be *weakly similar* if  $\rho(t_1) = \rho(t_2)$ . We denote this equivalence relation (which is a kernel of  $\rho$ ) by  $\sim_{ws}$ . Finally, let  $\mathcal{L}(R) = \bigcup_{r \in R} \mathcal{L}(r)$  for any  $R \subset \mathcal{T}_{Reg}$ .  $\square$

The idea behind this construction is that it allows to take into account the ACIZ-properties of only those occurrences of “+” in regular terms which appear at the very upper level. E.g., the term  $a + (b + c)^* + \emptyset$  is weakly similar to  $(b + c)^* + a$ , but not to  $a + (c + b)^*$ . Note that the equivalence relation  $\sim_{ws}$  is weaker than  $\mathcal{E}(ACIZ)$  on  $\mathcal{T}_{Reg}$ ; and on the subset of regular terms not having occurrences of  $\emptyset$  it is also weaker than  $\mathcal{E}(ACI)$ .

To relate finite sets of regular terms with corresponding regular terms modulo  $\sim_{ws}$ , we introduce the function  $\sum_- : \mathbf{SetReg} \rightarrow \mathcal{T}_{Reg}/\sim_{ws}$  which maps any singleton  $\{t\}$  to the (equivalence class of the) regular term  $t$  and satisfies the conditions

$$\sum \emptyset = \emptyset, \quad \sum(R_1 \cup R_2) = \sum R_1 + \sum R_2$$

for all  $R_1, R_2 \in \mathbf{SetReg}$ . Note that  $\mathcal{L}(\sum R) = \mathcal{L}(R)$  for any  $R \in \mathbf{SetReg}$ .

We shall also need an extension of the concatenation operation defined by the following equations for all  $t \in \mathcal{T}_{Reg} \setminus \{\emptyset, \lambda\}$ ,  $R \in \mathbf{SetReg}$ :

$$\begin{aligned} R \cdot \emptyset &= \emptyset, & R \cdot \lambda &= R, \\ R \cdot t &= \mathbf{if} \ \lambda \in R \ \mathbf{then} \ \{t\} \cup \{r \cdot t \mid r \in R \setminus \{\lambda\}\} \ \mathbf{else} \ \{r \cdot t \mid r \in R\} \ \mathbf{fi}. \end{aligned}$$

**Definition 4. (Linear forms)** Given a letter  $x \in \mathcal{A}$  and a term  $t \in \mathcal{T}_{Reg}$ , we call the pair  $\langle x, t \rangle$  a *monomial*. A (*non-deterministic*) *linear form* is a finite set of monomials. Let  $\mathbf{Lin}$  denote the semilattice  $\mathbf{Set}[\mathcal{A} \times \mathcal{T}_{Reg}]$  of linear forms. The function  $lf(-) : \mathcal{T}_{Reg} \rightarrow \mathbf{Lin}$ , returning a linear form of its argument, is defined recursively by the following equations:

$$\begin{aligned} lf(\emptyset) &= \emptyset, & lf(a^*) &= lf(a) \odot a^*, \\ lf(\lambda) &= \emptyset, & lf(a_0 \cdot b) &= lf(a_0) \odot b, \\ lf(x) &= \{\langle x, \lambda \rangle\}, & lf(a_1 \cdot b) &= lf(a_1) \odot b \cup lf(b) \\ lf(a + b) &= lf(a) \cup lf(b), \end{aligned}$$

for all  $x \in \mathcal{A}$ ,  $a, b \in \mathcal{T}_{Reg}$ ,  $a_0 \in \mathcal{T}_{Reg0}$ ,  $a_1 \in \mathcal{T}_{Reg1}$ . These equations involve an extension of concatenation,  $\odot : \mathbf{Lin} \times \mathcal{T}_{Reg} \rightarrow \mathbf{Lin}$ , defined as follows:

$$\begin{aligned} l \odot t &= \mathbf{if} \ t = \emptyset \ \mathbf{then} \ \emptyset \\ &\quad \mathbf{else} \ \mathbf{if} \ t = \lambda \ \mathbf{then} \ l \\ &\quad \quad \mathbf{else} \ \{ \langle x, p \cdot t \rangle \mid \langle x, p \rangle \in l \wedge p \neq \lambda \} \cup \{ \langle x, t \rangle \mid \langle x, \lambda \rangle \in l \} \\ &\quad \mathbf{fi} \ \mathbf{fi}. \end{aligned}$$

for all  $l \in \mathbf{Lin}$ ,  $t \in \mathcal{T}_{Reg}$ . □

Regarding a monomial  $\langle x, t \rangle$  as representing a regular term  $x \cdot t$ , the algebra  $\mathbf{Lin}$  is isomorphic to a subalgebra of  $\mathbf{SetReg}$ . This allows to apply the function  $\sum_-$  defined above to translate a linear form  $l$  into a regular term  $\sum l$  modulo weak similarity. The following proposition ensures that the function  $lf(-)$  provides a correct linear factorization of regular terms.

**Proposition 5.** *For any term  $t \in \mathcal{T}_{Reg}$  the following equation holds in the algebra  $\mathbf{Reg}[\mathcal{A}]$ :*

$$t = o(t) + \sum lf(t). \quad (12)$$

*Remark.* In general, a regular term  $t$  may have several linear forms which are distinct modulo  $\mathcal{E}(SR)$ , but all satisfy (12). Thus, the function  $lf(t)$  returns a *particular* linear form of  $t$ . Note that it can be computed by one pass over  $t$ . The definition of  $lf$  can be extended by further equations which provide more compact linear forms for some terms (e.g.,  $lf(a_1 \cdot a_1) = lf(a_1) \cdot a_1$ ), but make computations more expensive (cf. [1]). □

Now we come to the central definition of this paper.

**Definition 6. (Partial derivatives)** Given a regular term  $t$  and a letter  $x \in \mathcal{A}$ , a regular term  $p$  is called a *partial derivative of  $t$  w.r.t.  $x$*  if the linear form  $lf(t)$  contains a monomial  $\langle x, p \rangle$ . We define a function  $\partial_x : \mathcal{T}_{Reg} \rightarrow \mathbf{SetReg}$ , which returns a set of all non-zero partial derivatives of its argument w.r.t.  $x$ , as follows:

$$\partial_x(t) = \{ p \in \mathcal{T}_{Reg} \setminus \{\emptyset\} \mid \langle x, p \rangle \in lf(t) \} \quad (13)$$

The following equations extend this function allowing any word  $w \in \mathcal{A}^*$  and set of words  $W \subset \mathcal{A}^*$  at the place of  $x$  and any set of regular terms  $R \subset \mathcal{T}_{Reg}$  at the place of  $t$ :

$$\begin{aligned} \partial_\lambda(\emptyset) &= \emptyset, & \partial_w(R) &= \bigcup_{r \in R} \partial_w(r), \\ \partial_\lambda(t) &= \{t\} \text{ if } t \neq \emptyset, & \partial_W(t) &= \bigcup_{w \in W} \partial_w(t), \\ \partial_{w \cdot x}(t) &= \partial_x(\partial_w(t)), \end{aligned}$$

An element of the set  $\partial_w(t)$  is called a *partial (word) derivative of  $t$  w.r.t.  $w$* .  $\square$

*Example 1.* Let's compute partial derivatives of the term  $t = x^* \cdot (x \cdot x + y)^*$ . Let  $r$  stand for  $(x \cdot x + y)^*$ . Using Def. 4, we obtain:

$$If(t) = \{\langle x, t \rangle, \langle x, x \cdot r \rangle, \langle y, r \rangle\}, \quad If(x \cdot r) = \{\langle x, r \rangle\}, \quad If(r) = \{\langle x, x \cdot r \rangle, \langle y, r \rangle\},$$

hence

$$\begin{aligned} \partial_x(t) &= \{t, x \cdot r\}, & \partial_y(t) &= \{r\}, \\ \partial_{xx}(t) &= \partial_x(\{t, x \cdot r\}) = \{t, x \cdot r, r\}, & \partial_{yx}(t) &= \partial_x(\{r\}) = \{x \cdot r\}, \\ \partial_{xy}(t) &= \partial_y(\{t, x \cdot r\}) = \{r\}, & \partial_{yy}(t) &= \partial_y(\{r\}) = \{r\}, \end{aligned}$$

etc.  $\square$

The following facts explain semantics of partial derivatives and relate them to derivatives.

**Proposition 7.**  $\mathcal{L}(\partial_w(t)) = w \setminus \mathcal{L}(t)$  for any  $t \in \mathcal{T}_{Reg}$ ,  $w \in \mathcal{A}^*$ . In particular, any partial derivative  $p \in \partial_w(t)$  denotes a subset of the left quotient  $w \setminus \mathcal{L}(t)$ .

**Corollary 8.** For any  $t \in \mathcal{T}_{Reg}$ ,  $w \in \mathcal{A}^*$  the equation  $w^{-1}t = \sum \partial_w(t)$  holds in the algebra  $\mathbf{Reg}[\mathcal{A}]$ .  $\square$

Thus, partial derivatives in  $\partial_w(t)$  represents “parts” of the derivative  $w^{-1}t$  (that justifies their name).

### 3 Properties of Partial Derivatives.

Let  $\mathcal{PD}(t)$  stand for the set  $\partial_{\mathcal{A}^*}(t)$  of all (syntactically distinct) partial word derivatives of  $t$ . The next two theorems present important properties of  $\mathcal{PD}(t)$ .

The first theorem shows that  $\mathcal{PD}(t)$  is finite and gives a nice upper bound for its cardinality.

**Theorem 9.**  $|\partial_{\mathcal{A}^+}(t)| \leq \|t\|$  and  $|\mathcal{PD}(t)| \leq \|t\| + 1$  hold for any  $t \in \mathcal{T}_{Reg}$ .

**Corollary 10.**  $|\partial_W(t)| \leq \|t\| + 1$  holds for any  $W \subset \mathcal{A}^*$ .  $\square$

*Remark.* It follows from Def. 6 and Prop. 7 that the term  $\sum \partial_w(t)$  represents an *event derivative of  $t$  w.r.t.  $W$*  (cf. [8]). Thus, Corollary 10 implies Theorem 3 from [8, chapt.5] (which proves a half of Kleene's main theorem).  $\square$

*Example 2.* In notation of Example 1, we have  $\|t\| = 4$  and  $\mathcal{PD}(t) = \{t, x \cdot r, r\}$ .  $\square$

The second theorem clarifies the internal structure of partial derivatives.

**Theorem 11.** *Given a regular term  $t \in \mathcal{T}_{Reg}$ , any partial derivative of  $t$  is either  $\lambda$ , or a subterm of  $t$ , or a concatenation  $t_0 \cdot t_1 \cdot \dots \cdot t_n$  of several such subterms where  $n$  is not more than the number of occurrences of concatenation and Kleene star appearing in  $t$ .*  $\square$

It follows that the set  $\mathcal{PD}(t)$  can be represented by a data structure of a relatively small size: each partial derivative of  $t$  is just a (possibly empty) list of references to subterms of  $t$  and there are not more than  $\|t\| + 1$  such lists. In the next section it will be made clear that this data structure is virtually a set of states of an NFA recognizing the language  $\mathcal{L}(t)$  and that it can also serve as a basis for compact representation of the set of all  $\sim_{ws}$ -dissimilar derivatives of  $t$ .

## 4 Finite Automata Constructions Using Partial Derivatives

In this section we apply partial derivatives to a classical problem of turning regular expressions into finite automata. There are several well-known algorithms performing this task [14, 19, 4]. Nevertheless, new algorithms, aimed at reducing sizes of resulting automata, improving their performance, etc., keep appearing (see e.g. a survey [20]). Using partial derivatives, we get yet another new algorithms.

### 4.1 From regular expressions to small NFA.

In this subsection we describe a new algorithm turning a regular term  $t$  into an NFA having not more than  $\|t\| + 1$  states. The following theorem presents our construction.

**Theorem 12.** *Given a regular term  $t$  on an alphabet  $\mathcal{A}$ , let an automaton  $\mathbf{M}$  on  $\mathcal{A}$  have the set of states  $M = \mathcal{PD}(t)$ , the initial state  $\mu_0 = t$ , the transition function  $\tau$  defined by  $\tau(p, x) = \partial_x(p)$  for all  $p \in \mathcal{PD}(t)$ ,  $x \in \mathcal{A}$ , and the set of final states  $F = \{p \in \mathcal{PD}(t) \mid o(p) = \lambda\}$ . Then  $\mathbf{M}$  recognizes  $\mathcal{L}(t)$ .*  $\square$

To practically implement this construction, one needs to compute the set  $\mathcal{PD}(t)$  and the function  $\tau$  (the set  $F$  can be obtained in the obvious way). This can be done through the following iterative process

$$\langle \mathcal{PD}_0, \Delta_0, \tau_0 \rangle := \langle \emptyset, \{t\}, \emptyset \rangle \quad (14)$$

$$\mathcal{PD}_{i+1} := \mathcal{PD}_i \cup \Delta_i \quad (15)$$

$$\Delta_{i+1} := \bigcup_{p \in \Delta_i} \{q \mid \langle x, q \rangle \in \text{If}(p) \wedge q \notin \mathcal{PD}_{i+1}\} \quad (16)$$

$$\tau_{i+1} := \tau_i \cup \{ \langle p, x, q \rangle \mid p \in \Delta_i \wedge \langle x, q \rangle \in \text{If}(p) \} \quad (17)$$

for  $i = 0, 1, \dots$ . Here  $\tau$  is represented as a finite subset of  $M \times \mathcal{A} \times M$  (i.e., a transition relation). The set  $\Delta_i$  accumulates new partial derivatives appearing at each step. In not more than  $\|t\|$  steps  $\Delta_i$  becomes empty – then  $\mathcal{PD}_i$  and  $\tau_i$  contain the needed results. All the basic operations involved into this construction can be computed in time between  $O(n)$  and  $O(n^2)$ , hence it can be implemented as a respectably efficient program.

## 4.2 From regular expressions to DFA: improvements to Brzowski's algorithm

Our construction of NFA presented in Theorem 12 can easily be modified into a procedure constructing DFA: the set  $\mathcal{DD}(t) = \{ \partial_w(t) \mid w \in \mathcal{A}^* \}$  is to be taken as the set of states of the DFA, the initial state is the singleton  $\{t\}$ , the transition function is defined by  $\tau(P, x) = \partial_x(P)$  for all  $P \in \mathcal{DD}(t)$ ,  $x \in \mathcal{A}$ , and the set of final states is  $F = \{ P \in \mathcal{DD}(t) \mid o(\sum P) = \lambda \}$ .

**Proposition 13.** *The automaton  $\langle \mathcal{DD}(t), \tau, \{t\}, F \rangle$  presented above recognizes the language  $\mathcal{L}(t)$ .  $\square$*

The relation between the sets  $\partial_w(t)$  and the derivatives  $w^{-1}t$  given by Prop.7 readily demonstrates that this construction is just a modification of Brzowski's algorithm where each derivative  $w^{-1}t$  is substituted by a corresponding set  $\partial_w(t)$  of partial derivatives. However, the use of partial derivatives leads to several advantages:

1. Rather than to compute separately and to keep in memory all *ACI*-dissimilar derivatives of  $t$ , one can compute  $\mathcal{PD}(t)$  and represent each deterministic state  $\partial_w(t)$  as a set of *references* to corresponding elements in  $\mathcal{PD}(t)$ . Thus,  $\mathcal{PD}(t)$  serves as a relatively small *basis* for the set  $\mathcal{DD}(t)$  (and so for the set of all derivatives of  $t$ ).
2. Computing the set  $\mathcal{DD}(t)$  represented as suggested above, one compares its elements just as sets of references (that can be performed in  $O(n \log n)$  time), rather than checks equivalence of derivatives modulo  $\mathcal{E}(ACI)$ , or any other non-trivial congruence.
3. Components of the transition function  $\tau$  can be computed through the function  $lf(-)$  which gives a whole tuple of transitions  $\{ \langle x, \partial_x(P) \rangle \mid x \in \mathcal{A} \}$  by one pass over  $P$ . This is more efficient than to compute separately each derivative w.r.t.  $x \in \mathcal{A}$  (that requires one pass for each  $x$ ). The bigger the alphabet, the more one gains from this optimization.

These improvements to the original algorithm by Brzowski provide a more efficient programming implementation.<sup>4</sup>

*Remark.* The automata constructions presented above demonstrate that the relation between partial derivatives and derivatives is similar to the well known relation between

---

<sup>4</sup> Of course, one should bear in mind that the output of this procedure can have an exponential size.

NFA and DFA provided by the classical subset construction [17]. Really, suppose a regular term  $t$  is turned into an NFA as described in Theorem 12. This NFA can be transformed into an equivalent DFA by the subset construction; the states of the DFA will be represented by sets of states of the original NFA, i.e. by subsets of  $\mathcal{PD}(t)$ . On the other hand, the same DFA – with the set of states  $\mathcal{DD}(t)$  – can be obtained directly from  $t$  as described in Prop. 13. Note that this gives a new algebraic interpretation of the subset construction. Also, taking into account Theorem 11, we come to an interesting conclusion that *states of a DFA recognizing  $\mathcal{L}(t)$  are virtually finite sets of certain lists of subterms of  $t$ .*

## 5 Implementation and Examples

We have used the algebraic programming language OBJ3 [11] to develop a prototype implementation of the algorithms for computing partial derivatives of regular expressions and constructing NFA.

Recall that a finite automaton  $\mathbf{M} = \langle M, \tau, \mu_0, F \rangle$  can be represented by a finite system of *state equations* of the form

$$\mu := o(\mu) + x_1 \cdot \mu_1 + \dots + x_k \cdot \mu_k \quad (18)$$

for each state  $\mu \in M$  where  $x_i \in \mathcal{A}$  and  $\mu_i \in \tau(\mu, x_i)$ ,  $i = 1 \dots k$  (see e.g. [6]). Here  $o(\mu)$  is  $\lambda$  if  $\mu \in F$ , or  $\emptyset$  otherwise – in the latter case it is omitted from the sum. The components  $x_i \cdot \emptyset$  (if any) can also be omitted from the right-hand sides of (18), so that the resulting set of equations represents in general a non-complete NFA – without the sink state  $\emptyset$ .

Our program consists of several order-sorted term-rewriting systems implementing, in particular, Def.4 and the iterative process defined by the equations (14–17). It takes a regular term as an input and rewrites it into a set of state equations representing a corresponding NFA, and a set of partial derivatives corresponding to the states of the automaton. Below we present some examples obtained with the help of this program. We consider regular terms on the alphabet  $\mathcal{A} = \{a, b, c, \dots\}$ . The concatenation sign is omitted from the terms.

*Example 3.* This is a working example from [7]: the regular expression

$$t = (a + b)^* abb.$$

Our algorithm turns it into the following NFA with 4 states and 5 edges:

State equations	Partial derivatives
$S_1 := a \cdot S_1 + b \cdot S_1 + a \cdot S_2$	$(a + b)^* abb$
$S_2 := b \cdot S_3$	$bb$
$S_3 := b \cdot S_4$	$b$
$S_4 := \lambda$	$\lambda$

In [7] the expression was first turned into an NFA with 6 states and 11 edges by Berry and Sethi's algorithm. Then this NFA was transformed into a so-called

*compressed normalized NFA* with 5 states and 6 edges through several non-trivial optimizations. It is remarkable that our algorithm gives a smaller NFA without any additional optimization.  $\square$

*Example 4.* Given a natural constant  $n \geq 2$ , consider the following regular expression<sup>5</sup>

$$t_n = (\lambda + a + a^2 + \dots + a^{n-1})(a^n)^*.$$

One can see that  $\|t_n\| = n(n+1)/2$ . However, our algorithm turns  $t_n$  into an NFA with only  $n+1$  states. E.g., for  $n=4$  the NFA is as follows:

State equations	Partial derivatives
$S_1 := \lambda + a \cdot S_2 + a \cdot S_3 + a \cdot S_4 + a \cdot S_5$	$(\lambda + a + aa + aaa)(aaaa)^*$
$S_2 := \lambda + a \cdot S_5$	$(aaaa)^*$
$S_3 := a \cdot S_2$	$a(aaaa)^*$
$S_4 := a \cdot S_3$	$aa(aaaa)^*$
$S_5 := a \cdot S_4$	$aaa(aaaa)^*$

This demonstrates that in some cases our NFA can be an order of magnitude smaller than McNaughton and Yamada's or Berry and Sethi's ones.  $\square$

*Example 5.* This example is due to Gregory Kucherov: he proposed us to construct an automaton for the following regular expression (which is an example of those appearing in the study of word-rewriting systems with variables [13]):

$$t = (a+b)^*(babab(a+b)^*bab + bba(a+b)^*bab)(a+b)^*$$

Our algorithm turns this expression into the following NFA with 11 states:

State equations	Partial derivatives
$S_1 := a \cdot S_1 + b \cdot S_1 + b \cdot S_2 + b \cdot S_3$	$t$
$S_2 := a \cdot S_4$	$abab(a+b)^*bab(a+b)^*$
$S_3 := b \cdot S_5$	$ba(a+b)^*bab(a+b)^*$
$S_4 := b \cdot S_6$	$bab(a+b)^*bab(a+b)^*$
$S_5 := a \cdot S_7$	$a(a+b)^*bab(a+b)^*$
$S_6 := a \cdot S_8$	$ab(a+b)^*bab(a+b)^*$
$S_7 := a \cdot S_7 + b \cdot S_7 + b \cdot S_9$	$(a+b)^*bab(a+b)^*$
$S_8 := b \cdot S_7$	$b(a+b)^*bab(a+b)^*$
$S_9 := a \cdot S_{10}$	$ab(a+b)^*$
$S_{10} := b \cdot S_{11}$	$b(a+b)^*$
$S_{11} := \lambda + a \cdot S_{11} + b \cdot S_{11}$	$(a+b)^*$

Note that  $\|t\| = 22$ , so McNaughton and Yamada's or Berry and Sethi's algorithms would turn this expression into an NFA with 23 states.  $\square$

*Acknowledgements.* The author thanks Pierre Lescanne and Gregory Kucherov for helpful discussions and comments on a draft version of this paper.

<sup>5</sup> Which comes from so-called *cyclic identities*, see e.g. [8].

## References

1. V. M. Antimirov. Partial derivatives of regular expressions and finite automata constructions. Technical report, CRIN, 1994. (Forthcoming).
2. V. M. Antimirov and P. D. Mosses. Rewriting extended regular expressions (short version). In G. Rozenberg and A. Salomaa, editors, *Developments in Language Theory - At the Crossroads of Mathematics, Computer Science and Biology*, pages 195–209. World Scientific, Singapore, 1994.
3. V. M. Antimirov and P. D. Mosses. Rewriting extended regular expressions. *Theoretical Comput. Sci.*, 141, 1995. (To appear).
4. G. Berry and R. Sethi. From regular expressions to deterministic automata. *Theoretical Comput. Sci.*, 48:117–126, 1986.
5. J. A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11:481–494, 1964.
6. J. A. Brzozowski and E. L. Leiss. On equations for regular languages, finite automata, and sequential networks. *Theoretical Comput. Sci.*, 10:19–35, 1980.
7. C.-H. Chang and R. Paige. From regular expressions to DFA's using compressed NFA's. In A. Apostolico, M. Crochemore, Z. Galil, and U. Manber, editors, *Combinatorial Pattern Matching. Proceedings.*, volume 644 of *Lecture Notes in Computer Science*, pages 88–108. Springer-Verlag, 1992.
8. J. H. Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, 1971.
9. A. Ginzburg. A procedure for checking equality of regular expressions. *J. ACM*, 14(2):355–362, 1967.
10. J. A. Goguen and J. Meseguer. Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations. *Theoretical Comput. Sci.*, 105:217–273, 1992.
11. J. A. Goguen and T. Winkler. Introducing OBJ3. Technical Report SRI-CSL-88-9, Computer Science Lab., SRI International, 1988.
12. D. Krob. Differentiation of K-rational expressions. *International Journal of Algebra and Computation*, 2(1):57–87, 1992.
13. G. Kucherov and M. Rusinowitch. On Ground-Reducibility Problem for Word Rewriting Systems with Variables. In E. Deaton and R. Wilkerson, editors, *Proceedings 1994 ACM/SIGAPP Symposium on Applied Computing*, Phoenix (USA), Mar. 1994. ACM-Press.
14. R. McNaughton and H. Yamada. Regular expressions and state graphs for automata. *IEEE Trans. on Electronic Computers*, 9(1):39–47, 1960.
15. Y. Mizoguchi, H. Ohtsuka, and Y. Kawahara. A symbolic calculus of regular expressions. *Bulletin of Informatics and Cybernetics*, 22(3–4):165–170, 1987.
16. D. Perrin. Finite automata. In J. van Leeuwen, A. Meyer, M. Nivat, M. Paterson, and D. Perrin, editors, *Handbook of Theoretical Computer Science*, volume B, chapter 1. Elsevier Science Publishers, Amsterdam; and MIT Press, 1990.
17. M. O. Rabin and D. Scott. Finite automata and their decision problems. *IBM Journal of Research and Development*, 3(2):114–125, Apr. 1959.
18. A. Salomaa. *Theory of Automata*. Pergamon, 1969.
19. K. Thompson. Regular expression search algorithms. *Communication ACM*, 11(6):419–422, 1968.
20. B. W. Watson. A taxonomy of finite automata construction algorithms. Computing Science Note 93/43, Eindhoven University of Technology, The Netherlands, 1993.

This article was processed using the L<sup>A</sup>T<sub>E</sub>X macro package with LLNCS style