ELSEVIER

Theoretical Computer Science 155 (1996) 291–319

Theoretical Computer Science

Partial derivatives of regular expressions and finite automaton constructions [1]

Valentin Antimirov *

CRIN (CNRS) & INRIA-Lorraine Campus Scientifique, BP 239 F-54506, Vandœuvre-lès-Nancy Cedex, France

Received September 1995
Communicated by M. Nivat

Abstract

We introduce a notion of *partial derivative* of a regular expression and apply it to finite automaton constructions. The notion is a generalization of the known notion of *word derivative* due to Brzozowski: partial derivatives are related to non-deterministic finite automata (NFA's) in the same natural way as derivatives are related to deterministic ones (DFA's). We give a constructive definition of partial derivatives and prove several facts, in particular: (1) any derivative of a regular expression r can be represented by a finite set of partial derivatives of r; (2) the set of all partial derivatives of r is finite and its cardinality is less than or equal to one plus the number of occurrences of letters from $\mathscr{A}$ appearing in r; (3) any partial derivative of r is either a regular unit, or a subterm of r, or a concatenation of several such subterms. These theoretical results lead us to a new algorithm for turning regular expressions into relatively small NFA's and allow us to provide certain improvements to Brzozowski's algorithm for constructing DFA's. We also report on a prototype implementation of our NFA construction and present several examples.

0. Introduction

In 1964 Janusz Brzozowski introduced *word derivatives* of regular expressions and suggested an elegant algorithm for turning a regular expression r into a deterministic finite automaton (DFA) whose states are represented by derivatives of r [8].

Since then, derivatives of regular expressions have been recognized as a useful and productive concept. Conway [11] uses derivatives to develop various computational

[1] A short version of this paper [2] was presented at the 12th Annual Symposium on Theoretical Aspects of Computer Science, Munich, Germany, March 1995.

* Valentin Antimirov passed away in May 1995. This paper was prepared for its posthumous publication by his colleagues of the EURÉCA and PROTHÉO groups of CRIN and INRIA-Lorraine. Correspondence to: G. Kucherov, CRIN & INRIA Lorraine, Campus Scientifique, B.P. 239, 54506 Vandœvre-lès-Nancy Cedex, France. Email: kucherov@loria.fr.

SSDI 0304-3975(95)00182-4

procedures in the algebra of regular expressions and to investigate logical properties of this algebra. Krob [19] extends this differential calculus to a more general algebra of *K-rational* expressions. Brzozowski and Leiss [9] employ the idea of derivatives (in a slightly different form of *left quotients of language equations*) to describe relations between regular languages, boolean automata, and sequential networks. Using derivatives of a particular kind, so-called *continuations*, Berry and Sethi [5] provide a solid theoretical background for the McNaughton and Yamada algorithm [22] (a similar algorithm is due to Glushkov [14]) which turns a regular expression into a non-deterministic finite automaton (NFA). Ginzburg [13] uses derivatives to develop a procedure for proving equivalence of regular expressions; further development of this procedure is provided by Mizoguchi et al. [23]. Yet another procedure for proving equivalence of *extended* regular expressions is suggested by this author and Mosses [3] and is also based on some constructions closely related to derivatives.

In the present paper we come up with a new notion of *partial derivative* which is, in a sense, a "non-deterministic generalization" of that of a derivative: we demonstrate that partial derivatives are related to NFA's in the same natural way as derivatives are related to DFA's.

In Section 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 to compute linear forms and partial derivatives of a given regular expression. Then some basic properties of partial derivatives are established; in particular, we prove that any derivative of r can be represented by a finite set of some partial derivatives of r.

In Section 3 we study further properties of partial derivatives and prove two theorems which are the main theoretical results of this paper. The first theorem demonstrates that the set of all (syntactically distinct) partial derivatives of any regular expression r is finite[2] and its cardinality is quite small – less than or equal to one plus the number of occurrences of alphabet letters appearing in r. The second theorem clarifies the internal structure of partial derivatives. It turns out that any partial derivative of r is either a regular unit, or a subterm of r, or a concatenation of several such subterms. We also discuss some direct consequences of these two theorems.

In Section 4 we apply the above theoretical results to a classical problem of turning regular expressions into finite automata. It should be noted that there exist several classical algorithms performing this task [8, 22, 14, 31]. Nevertheless, new algorithms, aimed at reducing sizes of resulting automata, improving their performance, etc., keep appearing [5, 7, 10, 17] (cf. also [32] for a survey).

We present two new finite automaton constructions based on partial derivatives. The main point of the first one is that it turns a regular expression r into an NFA whose states are represented by partial derivatives of r. This implies that the above upper bound on the cardinality of the set of partial derivatives holds as an upper bound on the number of states of our NFA's. Note that this upper bound is exactly the number

[2] Note that derivatives do not enjoy this property.

of states of NFA's produced by a classical algorithm by McNaughton and Yamada [22] and Glushkov [14], as well as by its improvements due to Berry and Sethi [5] and Brüggemann-Klein [7]. In many cases our NFA's have actually fewer (in some cases, *much* fewer) states than this upper bound. Moreover, there are examples when our NFA's turn out to be smaller than those produced by a tricky Chang and Paige's algorithm [10] which uses a non-trivial representation of NFA's and involves several optimizations.

The second construction presented in Section 4 is a modification of Brzozowski's one [8]: it turns a regular expression r into a DFA whose states are represented by finite sets of partial derivatives. We discuss the advantages which are due to the use of partial derivatives (rather than derivatives) in this construction.

We have implemented our algorithm for turning regular expressions into NFA as an algebraic program in OBJ3 (cf. [16] for the description of the language). In Section 5 we briefly describe this implementation and present and discuss several examples.

Finally, in Section 6 we discuss the results of the present work and point out some directions for further research on partial derivatives.

To make the paper self-contained, we start with basic notions and notation. We assume the reader to be familiar with basic notions of universal algebra such as signatures, homomorphisms, congruences, and quotients.

1. Preliminaries

The purpose of this section is to recall some definitions and facts concerning regular languages, finite automata, regular expressions, and derivatives.

1.1. Semilattices, monoids, and semirings

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

An *upper semilattice* is an algebra with a binary operation $+$, called *join*, which is associative, commutative, and idempotent, i.e. satisfies the following set of axioms, which will be called *ACI-axioms*:

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

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

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

We denote by $\mathscr{E}(ACI)$ the least congruence generated by these equations (on some algebra having join in its signature) and call it an *ACI-congruence*. If the join has a neutral element 0 (which we call *zero*), one obtains an *upper semilattice with the bottom* satisfying an additional *Z-axiom*:

$$a+0=a. \tag{4}$$

We shall refer to the set of equations (1)–(4) as to *ACIZ-axioms*; a corresponding least congruence $\mathscr{E}(ACIZ)$ (on an algebra with an appropriate signature) is called an *ACIZ-congruence*. Note that the set $\mathbf{Set}[X]$ forms an upper semilattice with the union $\_\cup\_$ as the join and the empty set $\emptyset$ as the bottom.

Let $\mathscr{A}$ be a finite alphabet (a set of *letters*). Then $\mathscr{A}^*$ denotes the set of all finite words over $\mathscr{A}$ and also the *free monoid* over $\mathscr{A}$ with concatenation of words $\_\cdot\_$ as multiplication, and the empty word λ as the neutral element. Like any monoid, it satisfies the following axioms:

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

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

An *idempotent semiring* is an algebra with constants 0, λ and two binary operations $+$ and $\cdot$ such that it is simultaneously an upper semilattice (w.r.t. 0 and $+$) and a monoid (w.r.t. λ and $\_\cdot\_$) and, in addition, satisfies the following equational axioms:[3]

$$a \cdot 0 = 0 \cdot a = 0, \tag{7}$$

$$a \cdot (b + c) = a \cdot b + a \cdot c \tag{8}$$

$$(a + b) \cdot c = a \cdot c + b \cdot c \tag{9}$$

We shall refer to the set of all the Eqs. (1)–(9) as to *SR-axioms*; the least congruence generated by these (on an algebra with an appropriate signature) is called *SR-congruence* and denoted by $\mathscr{E}(SR)$.

1.2. Regular languages and finite automata

A *language* L over an alphabet $\mathscr{A}$ is a subset of $\mathscr{A}^*$. The set $\mathbf{Reg}[\mathscr{A}]$ of all *regular* (also called rational) languages over $\mathscr{A}$ is the least subset of the powerset $\mathscr{P}(\mathscr{A}^*)$ which includes the empty set $\emptyset$, the singletons $\{\lambda\}$ and $\{\alpha\}$ for all $\alpha \in \mathscr{A}$ and is closed under standard regular (rational) operations on languages – concatenation $L_1 \cdot L_2$, union $L_1 \cup L_2$, and iteration (Kleene star) L^* – all defined in the usual way.

The set $\mathbf{Reg}[\mathscr{A}]$, together with the regular operations, forms an algebra of regular languages (or a *regular algebra*, for short). Let $\mathbf{Reg1}[\mathscr{A}]$ be a subset of $\mathbf{Reg}[\mathscr{A}]$ consisting of all the regular languages containing the empty word λ. The complement of this subset is denoted by $\mathbf{Reg0}[\mathscr{A}]$ and consists of regular languages that are subsets of the set $\mathscr{A}^+ = \mathscr{A} \cdot \mathscr{A}^*$ of all non-empty words over $\mathscr{A}$

Given a language L, a *left quotient of* L *w.r.t. a word* w, written $w^{-1}L$, is the language $\{u \in \mathscr{A}^* \mid w \cdot u \in L\}$. It follows, for any words w, $u \in \mathscr{A}^*$, that membership $w \in L$ is equivalent to $\lambda \in w^{-1}L$ and that the left quotient $(w \cdot u)^{-1}L$ is equal to $u^{-1}(w^{-1}L)$. Any left quotient of a regular language L is a regular language too and the set $\{w^{-1}L \mid w \in \mathscr{A}^*\}$ of all the left quotients of L w.r.t. the words over $\mathscr{A}$ is finite.

[3] Concatenation has higher priority than join. When appropriate, we omit the concatenation symbol from expressions.

We consider a *(non-deterministic) finite automaton* **M** over an alphabet $\mathscr{A}$ as a quadruple $\mathbf{M} = \langle M, \tau, \mu_0, F\rangle$ where M is a *set of states*, $\tau : M \times \mathscr{A} \to \mathbf{Set}[M]$ is a *transition function*, (which can also be represented as a relation $\tau \subseteq M \times \mathscr{A} \times M$), $\mu_0 \in M$ is an *initial state*, and $F \subseteq M$ is a set of *final* (terminal) states.[4] The automaton is called *deterministic* if $|\tau(\mu, x)| \leqslant 1$ for all $\mu \in M$, $x \in \mathscr{A}$; in this case the transition function is represented as a partial one, $\tau : M \times \mathscr{A} \tilde{\to} M$, returning a state or nothing. The function can always be completed to a total one by adding one "sink" state 0 to M such that $\tau(0, x) = 0$ for all $x \in \mathscr{A}$. In any of the above cases, the extension $\tau(\mu, w)$ of the transition function to the words $w \in \mathscr{A}^*$ is defined in the usual way (by recursion on w). A word $w \in \mathscr{A}^*$ 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 a state μ is denoted by $\mathscr{L}_M(\mu)$; then the language $\mathscr{L}_M(\mu_0)$ is said to be *recognized* by **M**.

Given a deterministic automaton **M**, the following important equation holds:

$$\mathscr{L}_M(\tau(\mu, w)) = w^{-1}\mathscr{L}_M(\mu) \tag{10}$$

for all states $\mu \in M$ and words $w \in \mathscr{A}^*$. In particular, the set of words $\mathscr{L}_M(\mu)$ accepted by a state $\mu = \tau(\mu_0, w)$ is exactly the left quotient $w^{-1}\mathscr{L}_M(\mu_0)$ of the language recognized by **M**.

This leads to the following abstract construction of a DFA recognizing a given regular language L: take the set of states M to be the set of left quotients $\{w^{-1}L \mid w \in \mathscr{A}^*\}$ with the initial state $\mu_0 = \lambda^{-1}L = L$ and the set of final states $F = \{\mu \in M \mid \lambda \in \mu\}$; then define the transition function by $\tau(\mu, x) = x^{-1}\mu$. This construction comes from the well-known Myhill – Nerode theorem [24, 25, 27] (cf. also [26, p. 9]).

1.3. Regular expressions

Regular (also called rational) expressions are terms on the signature of the regular algebra $\mathbf{Reg}[\mathscr{A}]$. Actually, there exist different ways of choosing the signature and of formalizing the algebra. In this paper we follow the idea of [3] that $\mathbf{Reg}[\mathscr{A}]$ should be regarded as an *order-sorted* algebra [15] having a sort $\mathscr{A}$ for an alphabet which is a subsort of a sort ***Reg*** for all the regular expressions. Here we also introduce two subsorts of ***Reg***, namely *Reg0* and *Reg1*, to distinguish regular expressions denoting elements of $\mathbf{Reg0}[\mathscr{A}]$ and $\mathbf{Reg1}[\mathscr{A}]$, respectively.

To sum up, the order-sorted signature *REG* over the alphabet $\mathscr{A} = \{x_1, x_2, \ldots, x_k\}$ is presented in Table 1. (The argument places of operations are indicated by the underbar character "_". Kleene star has the highest priority among all the operations on regular expressions.)

Sets of ground terms on the signature *REG* of the sorts ***Reg***, *Reg0*, and *Reg1* are defined in the usual way and denoted by $\mathscr{T}_{Reg}$, $\mathscr{T}_{Reg0}$, and $\mathscr{T}_{Reg1}$ respectively. Note that $\mathscr{T}_{Reg}$ is a disjoint union of $\mathscr{T}_{Reg0}$ and $\mathscr{T}_{Reg1}$. In what follows we call the elements of $\mathscr{T}_{Reg}$ *regular terms*.

[4] This is not the most general definition, since it does not allow λ-transitions; but in this paper we need only this particular kind of NFAs.

Table 1
Signature *REG*

Sorts	$\mathscr{A}$, ***Reg***, *Reg0*, *Reg1*.
Subsorts	$\mathscr{A} \leqslant$ *Reg0* $\leqslant$ ***Reg***, *Reg1* $\leqslant$ ***Reg***.
Constants	0 : *Reg0*; λ : *Reg1*; $x_1, x_2, \ldots, x_k : \mathscr{A}$.
Operations	$\_+\_$: ***Reg*** ***Reg*** $\to$ ***Reg*** $\quad \_\cdot\_$: ***Reg*** ***Reg*** $\to$ ***Reg***
	$\_+\_$: *Reg1* ***Reg*** $\to$ *Reg1* $\quad \_\cdot\_$: *Reg0* ***Reg*** $\to$ *Reg0*
	$\_+\_$: ***Reg*** *Reg1* $\to$ *Reg1* $\quad \_\cdot\_$: ***Reg*** *Reg0* $\to$ *Reg0*
	$\_+\_$: *Reg0* *Reg0* $\to$ *Reg0* $\quad \_\cdot\_$: *Reg1* *Reg1* $\to$ *Reg1*
	$\_^*$: ***Reg*** $\to$ *Reg1*.

A regular term t denotes a regular language $\mathscr{L}(t)$ and this interpretation is determined by the following homomorphism $\mathscr{L}(\_)$ from the absolutely free algebra of regular terms $\mathscr{T}_{Reg}$ to $\mathbf{Reg}[\mathscr{A}]$:

$$\mathscr{L}(0) = \emptyset, \qquad \mathscr{L}(\lambda) = \{\lambda\}, \qquad \mathscr{L}(\alpha) = \{\alpha\}, \qquad \mathscr{L}(\mathrm{r} \cdot \mathrm{t}) = \mathscr{L}(\mathrm{r}) \cdot \mathscr{L}(\mathrm{t}),$$
$$\mathscr{L}(r + t) = \mathscr{L}(r) \cup \mathscr{L}(t), \qquad \mathscr{L}(t^*) = \mathscr{L}(t)^*$$

for all $\alpha \in \mathscr{A}$, $r, t \in \mathscr{T}_{Reg}$. It can be checked that the homomorphism maps $\mathscr{T}_{Reg0}$ and $\mathscr{T}_{Reg1}$ to $\mathbf{Reg0}[\mathrm{A}]$ and $\mathbf{Reg1}[\mathrm{A}]$, respectively – as expected according to the order-sorted structure of the algebra. Let $\mathscr{E}(\mathbf{Reg})$ be the kernel of this homomorphism, i.e. the congruence on $\mathscr{T}_{Reg}$ consisting of all the pairs $\langle t_1, t_2 \rangle$ such that $\mathscr{L}(t_1) = \mathscr{L}(t_2)$ (this congruence can also be considered as a set of all ground equations $t_1 = t_2$ valid in $\mathbf{Reg}[\mathscr{A}]$).

Recall that $\mathbf{Reg}[\mathscr{A}]$ is an idempotent semiring, i.e. satisfies all the SR-axioms (1)–(9). There are further axioms involving Kleene star, e.g.

$$\lambda + a^* \cdot a = \lambda + a \cdot a^* = a^*, \tag{11}$$

$$(a + \lambda)^* = a^*, \tag{12}$$

and many others. (It is known that the whole set of equational properties of Kleene star cannot be captured by a finite set of equational axioms – the ground equational theory of $\mathbf{Reg}[\mathscr{A}]$ is not finitely based [28, 11, 29]. But for the purposes of this paper we shall need only SR-axioms and the simplest properties of the star.)

Thus, we have the following chain of congruences

$$\emptyset \subset \mathscr{E}(ACI) \subset \mathscr{E}(ACIZ) \subset \mathscr{E}(SR) \subset \mathscr{E}(\mathbf{Reg}) \tag{13}$$

and the corresponding chain of quotients of $\mathscr{T}_{Reg}$ related by surjective homomorphisms:

$$\mathscr{T}_{Reg} \to \mathscr{T}_{Reg}/_{\mathscr{E}(ACI)} \to \mathscr{T}_{Reg}/_{\mathscr{E}(ACIZ)} \to \mathscr{T}_{Reg}/_{\mathscr{E}(SR)} \to \mathscr{T}_{Reg}/_{\mathscr{E}(\mathbf{Reg})} \tag{14}$$

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

1.4. Derivatives

The *constant part* (also called the *output*, cf. [11]) of a regular term t, written $o(t)$, is defined as follows:

$$o(t) = \textbf{if}\ \ \lambda \in \mathscr{L}(t)\ \ \textbf{then}\ \ \lambda\ \textbf{else}\ \ 0\ \textbf{fi}. \tag{15}$$

Note that $o(a_0) = 0$ for all $a_0 \in \mathscr{T}_{Reg0}$ and $o(a_1) = \lambda$ for all $a_1 \in \mathscr{T}_{Reg1}$.

Definition 1.1 (Derivatives [8]). For any letter $x \in \mathscr{A}$ and word $w \in \mathscr{A}^*$, the functions $x^{-1}(\_)$ and $w^{-1}(\_)$ on $\mathscr{T}_{Reg}$ computing *(word) derivatives* of regular terms (w.r.t. x and w, respectively) are defined recursively by the following equations:[5]

$$x^{-1}0 = 0, \tag{16}$$

$$x^{-1}\lambda = 0, \tag{17}$$

$$x^{-1}y = \textbf{if}\ \ x = y\ \ \textbf{then}\ \ \lambda\ \textbf{else}\ \ 0\ \textbf{fi}, \tag{18}$$

$$x^{-1}(r + t) = x^{-1}r + x^{-1}t, \tag{19}$$

$$x^{-1}(r^*) = (x^{-1}r) \cdot r^*, \tag{20}$$

$$x^{-1}(r \cdot t) = \textbf{if}\ \ o(r) = 0\ \ \textbf{then}\ \ (x^{-1}r) \cdot t\ \textbf{else}\ \ (x^{-1}r) \cdot t + x^{-1}t\ \textbf{fi}, \tag{21}$$

$$\lambda^{-1}r = r, \tag{22}$$

$$(w \cdot x)^{-1}r = x^{-1}(w^{-1}r). \tag{23}$$

for all $y \in \mathscr{A}$, $r, t \in \mathscr{T}_{Reg}$.

All the equations in Definition 1.1 are stable w.r.t. congruences on $\mathscr{T}_{Reg}$ mentioned in (13), therefore the functions $x^{-1}(\_)$ and $w^{-1}(\_)$ are correctly defined on corresponding quotients of $\mathscr{T}_{Reg}$.

It is not difficult to check that the equation $\mathscr{L}(w^{-1}r) = w^{-1}\mathscr{L}(r)$ holds for any regular term r and word $w \in \mathscr{A}^*$, i.e., any derivative of r denotes a corresponding left quotient of $\mathscr{L}(r)$. Hence the membership $w \in \mathscr{L}(r)$ is equivalent to $o(w^{-1}r) = \lambda$.

This correspondence between derivatives (considered as terms) and left quotients is not one-to-one: given two different words w_1, w_2, the two regular terms ${w_1}^{-1}r$ and ${w_2}^{-1}r$ may be syntactically distinct, but denote the same language ${w_1}^{-1}\mathscr{L}(r) = {w_2}^{-1}\mathscr{L}(r)$. Moreover, for some regular terms, Definition 1.1 gives an infinite set of (syntactically distinct) word derivatives denoting the same language. Therefore, in order to use derivatives as states of a DFA, one has to consider them modulo some equivalence relation on $\mathscr{T}_{Reg}$. For this purpose, the notion of *similarity* of derivatives was introduced in [8]. We slightly generalize this notion below.

Given a set E of equations on the signature *REG*, two derivatives are said to be *E-similar* if they are equivalent modulo E (i.e. modulo the least congruence $\mathscr{E}(E)$ generated by E on $\mathscr{T}_{Reg}$). Then the set $\mathscr{D}_E(r)$ of all *E-dissimilar* derivatives of a regular term r is defined to be a set of representatives of the equivalence classes modulo E of the terms $w^{-1}r$ for all $w \in \mathscr{A}^*$.

[5] There is a slight deviation in one of these equations from the original one in [8]; the reason for this is explained in a remark below. Note also that our notation for derivatives differs from [8], but is compatible with [26].

Suppose we have some E such that the set $\mathscr{D}_E(r)$ is finite for any r. Then the following proposition, which presents Brzozowski's method for constructing DFA, holds:

Proposition 1.2. *Given a regular term t, consider a deterministic finite automaton* $\mathbf{M}$ *which has the set of states $M = \mathscr{D}_E(t)$, the initial state $\mu_0 = t$, the transition function defined by $\tau(r, x) = x^{-1}r$ for all $x \in \mathscr{A}$, $r \in M$, and the set of final states $F = \{ r \in M \mid o(r) = \lambda \}$. Then* $\mathbf{M}$ *recognizes the language $\mathscr{L}(t)$.*

Provided the equivalence modulo E is decidable, the proposition gives an elegant and conceptually simple algorithm which is a constructive version of the above-mentioned abstract DFA-construction based on the Myhill–Nerode theorem. An important question is how to choose the set E in order to provide finiteness of $\mathscr{D}_E(r)$ for any r.

It is known that the set of all $\mathscr{E}(\mathbf{Reg})$-dissimilar derivatives of any regular term r is finite[6] (cf. [8, Theorem 4.3]). However, it is not easy to check (non)equivalence of regular terms modulo $\mathscr{E}(\mathbf{Reg})$, so a finer congruence on $\mathscr{T}_{Reg}$ is needed to make the construction more practical.

Several candidates for such a congruence have been proposed in the literature. In [11, 26] derivatives are considered modulo all SR-axioms and it is proved that the set $\mathscr{D}_{SR}(r)$ is finite. A slightly finer congruence $\mathscr{E}(E)$, where E is the set of SR-axioms without distributivity laws (8) (9), is used in [6, 5]. But the finest solution is due to Brzozowski: he has proved that the set of all ACI-dissimilar word derivatives of any regular term is finite [8, Theorem 5.2].

Remark 1.3. We note that the original definition of derivatives in [8] uses the equation

$$x^{-1}(r \cdot t) = (x^{-1}r) \cdot t + o(r) \cdot x^{-1}t \tag{24}$$

instead of (21). However, computing with this equation the derivatives of the expression $(x + y)^*$ w.r.t. the words x, xx, xxx, etc., we obtain an infinite sequence of regular terms non-equivalent modulo $\mathscr{E}(ACI)$. Therefore, to save the nice result on finiteness of the set of ACI-dissimilar derivatives, we have changed Eq. (24) to (21). One can check that with this modification the original proof of the result becomes correct, cf. [8, p. 493].

Of course, it is much easier to check equivalence of regular terms modulo $\mathscr{E}(ACI)$, than modulo $\mathscr{E}(\mathbf{Reg})$. However, even with this possibility, the Brzozowski construction has been considered not sufficiently efficient to use in practice. Indeed, derivatives are regular terms which may have relatively big size, so it may take much time to compute and much space to represent the set $\mathscr{D}_E(r)$. One of the applications of the technique which we develop in this paper will be a much more economical representation of derivatives; this will help us to improve the efficiency of this construction to some extent.

[6] Actually, this set is in one-to-one correspondence to the set of all left quotients of $\mathscr{L}(r)$.

2. Introducing partial derivatives

Our definition of partial derivatives involves some auxiliary technical constructions and hence, we prefer to outline first basic ideas that lead to this notion.

2.1. Basic ideas

It is a folk knowledge that any regular expression r over an alphabet $\mathscr{A} = \{x_1, \ldots, x_k\}$ can be represented in the following form:

$$r = o(r) + x_1 \cdot r_1 + \cdots + x_k \cdot r_k, \tag{25}$$

where all the r_i are some regular expressions (see, e.g., [8, 11, 29]). In particular, one can take each r_i to be the derivative $x_i^{-1}r$.

This representation is closely related to a Brzozowski's DFA for r: each summand $x_i \cdot r_i$ corresponds to the transition from the state r to the state r_i on the letter x_i. For this reason, the right-hand side of (25) can be called a *deterministic linear factorization* of r. What makes it *deterministic*, is the following two requirements:

- each letter $x_i \in \mathscr{A}$ occurs as the head of a summand $x_i \cdot r_i$;
- the heads of different summands are different letters.

Now the idea is to define a more general kind of *non-deterministic* linear factorizations (n.l.f., for short) by dropping these requirements. For example, the regular expression x^*xy over the alphabet $\mathscr{A} = \{x, y\}$ can be factorized as follows:

$$x^*xy = x \cdot x^*xy + x \cdot y. \tag{26}$$

There are two technical problems here. The first one comes from the fact that a regular expression may have several essentially different n.l.f. For instance, one can check that the equation

$$x^*xy = x \cdot (xx)^*xy + x \cdot x(xx)^*xy + x \cdot y \tag{27}$$

is valid in $\mathbf{Reg}[\mathscr{A}]$ and therefore provides yet another n.l.f. for x^*xy. This means that we have to be more specific and define a particular n.l.f. for each regular term.

Our intention to make such a definition computationally efficient raises the second problem. In order to achieve this goal, we would like to treat regular expressions as free terms, i.e. elements of $\mathscr{T}_{Reg}$. On the other hand, we do have to consider regular expressions modulo some equivalence relation: e.g. the term $x \cdot y + x \cdot x^*xy$ is syntactically different from the right-hand side of (26), but represents virtually the same n.l.f.

To cope with this problem, we are going to work with *finite sets* of regular terms. The corresponding algebraic constructions are introduced in the next subsection. They will provide us with an appropriate equivalence relation on $\mathscr{T}_{Reg}$ and allow to give computationally efficient definitions.

2.2. Set representation and fine similarity of regular terms

Let **SetReg** be the upper semilattice $\mathbf{Set}[\mathscr{T}_{Reg}\backslash\{0\}]$ of finite sets of non-zero regular terms. The interpretation function $\mathscr{L} : \mathscr{T}_{Reg} \to \mathbf{Reg}[\mathscr{A}]$ extends to **SetReg** as follows:

$$\mathscr{L}(R) = \bigcup_{t\in R} \mathscr{L}(t)$$

for all $R \in \mathbf{SetReg}$.

Definition 2.1. Let a function $\rho : \mathscr{T}_{Reg} \to \mathbf{SetReg}$ satisfy the equations

$$\rho(0) = \emptyset, \tag{28}$$

$$\rho(t_1 + t_2) = \rho(t_1) \cup \rho(t_2) \tag{29}$$

for all $t_1, t_2 \in \mathscr{T}_{Reg}$ and map any other regular term t into the singleton $\{t\}$. Two regular terms t_1, t_2 are said to be *finely similar* if $\rho(t_1) = \rho(t_2)$. We denote this equivalence relation (which is a kernel of ρ) by $\sim_\rho$.

Observe that this construction allows us to take into account the ACIZ-properties of only *some* of the occurrences of $\_ + \_$ in a regular term r, namely of those which appear at the very upper level of r. For example, the term $a + (b + c)^* + a + 0$ is finely similar to $(b+c)^* + a$, but not to $a+(c+b)^*$. Note that the equivalence relation $\sim_\rho$ on $\mathscr{T}_{Reg}$ is finer than $\mathscr{E}(ACIZ)$; and it is also finer than $\mathscr{E}(ACI)$ on the subset of regular terms without occurrences of 0.

Let $\mathscr{T}_{Reg}/_{\sim_\rho}$ denote the factor set of $\mathscr{T}_{Reg}$ by the equivalence relation $\sim_\rho$.[7] The next definition relates elements of **SetReg** to regular terms modulo fine similarity.

Definition 2.2. Let the function $\Sigma\_ : \mathbf{SetReg} \to \mathscr{T}_{Reg}/_{\sim_\rho}$ be defined as follows: it maps any non-empty set $\{t_1, \ldots, t_k\}$ to the equivalence class of the term $t_1 + \cdots + t_k$ and the empty set $\emptyset$ to (the equivalence class of) 0. Any term from an equivalence class ΣR denotes the same regular language $\mathscr{L}(R)$, so we write $\mathscr{L}(\Sigma R)$ and $o(\Sigma R)$ without ambiguity. We often say "the term" ΣR when it makes no difference which of the representatives of this equivalence class is considered.

We shall also need an extension of the concatenation operation,

$$\_ \cdot \_ : \mathbf{SetReg} \times \mathscr{T}_{Reg} \to \mathbf{SetReg},$$

defined recursively by the following equations

$$R \cdot 0 = \emptyset, \tag{30}$$

$$R \cdot \lambda = R, \tag{31}$$

$$\{r\} \cdot t = \textbf{if } r = \lambda \textbf{ then } t \textbf{ else } r \cdot t \textbf{ fi}, \tag{32}$$

[7] Note that $\sim_\rho$ is not a congruence on $\mathscr{T}_{Reg}$. E.g., $(a + b) \sim_\rho (b + a)$, but $(a + b)^*$ is not finely similar to $(b + a)^*$.

$$(R \cup R') \cdot t = R \cdot t \cup R' \cdot t \tag{33}$$

for all $R, R' \in \mathbf{SetReg}$, $r \in \mathscr{T}_{Reg} \setminus \{0\}$, $t \in \mathscr{T}_{Reg} \setminus \{0, \lambda\}$. Note that $\mathscr{L}(R \cdot t) = \mathscr{L}(R) \cdot \mathscr{L}(t) = \mathscr{L}((\Sigma R) \cdot t)$ and that $|R \cdot t| \leqslant |R|$.

2.3. Linear forms and partial derivatives of regular terms

Definition 2.3. Given a letter $x \in \mathscr{A}$ and a term $t \in \mathscr{T}_{Reg}$, we call the pair $\langle x, t \rangle$ a *monomial* with the *head* x and the *tail* t; then let $\mathbf{Mon} = \mathscr{A} \times \mathscr{T}_{Reg}$ denote the set of all monomials (over a given alphabet $\mathscr{A}$). A *(non-deterministic) linear form* is a finite subset of **Mon**. Let $\mathbf{Lin} = \mathbf{Set}[\mathbf{Mon}]$ be the set (and the upper semilattice) of linear forms.

Let us regard a monomial $\langle x, t \rangle$ as representing a regular term $x \cdot t$. Then **Mon** is isomorphic to a subalgebra of $\mathscr{T}_{Reg}$ and **Lin** is isomorphic to a sub-semilattice of **SetReg**. This allows us to write

$$\mathscr{L}(\langle x, t \rangle) = \mathscr{L}(x \cdot t) = \{x\} \cdot \mathscr{L}(t) \tag{34}$$

for the language denoted by a monomial $\langle x, t \rangle \in \mathbf{Mon}$, as well as

$$\mathscr{L}(l) = \bigcup_{\langle x, t \rangle \in l} \mathscr{L}(\langle x, t \rangle) = \bigcup_{\langle x, t \rangle \in l} \{x\} \cdot \mathscr{L}(t) \tag{35}$$

for the language denoted by a linear form $l \in \mathbf{Lin}$. Furthermore, we can use the function $\Sigma_{\_}$ to translate any linear form $l = \{\langle x_1, r_1 \rangle, \ldots, \langle x_k, r_k \rangle\}$ into a regular term modulo fine similarity:

$$\Sigma l = x_1 \cdot r_1 + \cdots + x_k \cdot r_k \tag{36}$$

Note that $\lambda \notin \mathscr{L}(l)$ for any l.

Our next goal is to set up a correspondence between regular terms and linear forms such that a linear form l_t, corresponding to a term t, would satisfy the equation

$$t = o(t) + \Sigma l_t \tag{37}$$

providing a non-deterministic linear factorization for t. This is implemented in the next definition.

Definition 2.4. The binary operation

$$\_ \odot \_ : \mathbf{Lin} \times \mathscr{T}_{Reg} \to \mathbf{Lin}$$

is an extension of concatenation to linear forms and is defined recursively by the following equations:

$$l \odot 0 = \varnothing \tag{38}$$

$$l \odot \lambda = l \tag{39}$$

$$\emptyset \odot t = \emptyset \tag{40}$$

$$\{\langle x,\ 0\rangle\} \odot t = \{\langle x,\ 0\rangle\} \tag{41}$$

$$\{\langle x,\ \lambda\rangle\} \odot t = \{\langle x,\ t\rangle\} \tag{42}$$

$$\{\langle x,\ p\rangle\} \odot t = \{\langle x,\ p \cdot t\rangle\} \tag{43}$$

$$(l \cup l') \odot t = (l \odot t) \cup (l' \odot t) \tag{44}$$

for all $l,\ l' \in \mathbf{Lin}$, $t \in \mathscr{T}_{Reg} \setminus \{0,\ \lambda\}$, $p \in \mathscr{T}_{Reg} \setminus \{0,\ \lambda\}$. The function

$$lf(\_) : \mathscr{T}_{Reg} \to \mathbf{Lin},$$

returning a linear form of its argument, is defined recursively by the following equations:

$$lf(0) = \emptyset \tag{45}$$

$$lf(\lambda) = \emptyset \tag{46}$$

$$lf(x) = \{\langle x,\ \lambda\rangle\} \tag{47}$$

$$lf(r + t) = lf(r) \cup lf(t) \tag{48}$$

$$lf(r^*) = lf(r) \odot r^* \tag{49}$$

$$lf(r_0 \cdot t) = lf(r_0) \odot t \tag{50}$$

$$lf(r_1 \cdot t) = lf(r_1) \odot t \cup lf(t) \tag{51}$$

for all $x \in \mathscr{A}$, $r, t \in \mathscr{T}_{Reg}$, $r_0 \in \mathscr{T}_{Reg0}$, $r_1 \in \mathscr{T}_{Reg1}$.

It is easy to check that the language $\mathscr{L}(l \odot t)$ is equal to $\mathscr{L}(l) \cdot \mathscr{L}(t)$; hence the equation $\Sigma(l \odot t) = (\Sigma l) \cdot t$ holds in $\mathbf{Reg}[\mathscr{A}]$.

The following proposition ensures correctness of the definition of $lf(\_)$.

Proposition 2.5. *For any term $t \in \mathscr{T}_{Reg}$ the following equation holds in the algebra* $\mathbf{Reg}[\mathscr{A}]$:

$$t = o(t) + \Sigma lf(t). \tag{52}$$

Proof. We have to show that both sides in (52) denote the same language, that is to prove the equation $\mathscr{L}(t) = \mathscr{L}(o(t)) \cup \mathscr{L}(lf(t))$. Since $\lambda \notin \mathscr{L}(lf(t))$, this is equivalent to the equation

$$\mathscr{L}(lf(t)) = \mathscr{L}(t) \setminus \{\lambda\}. \tag{53}$$

We prove the latter by induction on the structure of t.

The base cases, when t is 0, or λ, or a letter $x \in \mathscr{A}$, are obvious. Assuming that (53) is valid for $a, b \in \mathscr{T}_{Reg}$, direct computations show that it holds valid for $t = a + b$, $t = a^*$, and $t = a \cdot b$. For instance, for $t = a^*$ the computations go as follows:

$$\begin{aligned}\mathscr{L}(lf(a^*)) = \mathscr{L}(lf(a)\odot a^*)) &= \mathscr{L}(lf(a))\cdot\mathscr{L}(a^*) = (\mathscr{L}(a)\setminus\{\lambda\})\cdot\mathscr{L}(a)^*\\ &= (\mathscr{L}(a)\setminus\{\lambda\})\cdot(\mathscr{L}(a)\setminus\{\lambda\})^* = (\mathscr{L}(a)\setminus\{\lambda\})^*\setminus\{\lambda\}\\ &= \mathscr{L}(a)^*\setminus\{\lambda\} = \mathscr{L}(a^*)\setminus\{\lambda\},\end{aligned}$$

since $(L\setminus\{\lambda\})^* = L^*$ for any $L \in \mathbf{Reg}[\mathscr{A}]$ and $L_0\cdot L_0^* = L_0^*\setminus\{\lambda\}$ for any $L_0 \in \mathbf{Reg0}[\mathscr{A}]$. The other cases are treated in a similar way. □

Example 2.6. Let us calculate a linear form of the following regular term

$$t = x^*(xx + y)^*$$

on the alphabet $\{x,\ y\}$ (we omit the concatenation symbol). Using the equations of Definition 2.4, we obtain:

$$\begin{aligned}lf(t) &= lf(x^*(xx+y)^*) = lf(x^*)\odot(xx+y)^* \ \cup\ lf((xx+y)^*)\\ &= lf(x)\odot x^*\odot(xx+y)^* \ \cup\ lf((xx+y))\odot(xx+y)^*\\ &= \{\langle x,\ \lambda\rangle\}\odot x^*\odot(xx+y)^* \ \cup\ (lf(xx) \ \cup\ lf(y))\odot(xx+y)^*\\ &= \{\langle x,\ x^*(xx+y)^*\rangle\} \ \cup\ (lf(x)\odot x \ \cup\ \{\langle y,\ \lambda\rangle\})\odot(xx+y)^*\\ &= \{\langle x,\ x^*(xx+y)^*\rangle\} \ \cup\ \{\langle x,\ \lambda\rangle\}\odot x\odot(xx+y)^* \ \cup\ \{\langle y,\ \lambda\rangle\}\odot(xx+y)^*\\ &= \{\langle x,\ x^*(xx+y)^*\rangle,\ \langle x,\ x(xx+y)^*\rangle,\ \langle y,\ (xx+y)^*\rangle\}.\end{aligned}$$

This gives the following non-deterministic linear factorization of the term t:

$$t = o(t) + \Sigma lf(t) = \lambda + x\cdot x^*(xx+y)^* + x\cdot x(xx+y)^* + y\cdot(xx+y)^*. \ \square$$

Remark 2.7. It is worth noting several technical tricks in the definition of the function $lf(\_)$. Let us consider the set of equations (38)–(51) as a term-rewriting system (t.r.s.) [12] defined over the extension of the signature *REG* by

- new sorts – *Mon* for monomials, and *Lin* for linear forms (i.e., finite sets of monomials);
- new function symbols $lf(\_)$, $\_\odot\_$, and constructors of finite sets $\emptyset$, $\{\_\}$, and $\_\cup\_$ (as usual, the set union $\_\cup\_$ is to be considered as an associative-commutative operation and two rewrite rules $l\cup l \to l$, $l\cup\emptyset \to l$ are to be added to the t.r.s. to take into account its further properties).

Then one can see the following points.

1. The t.r.s. essentially relies on the order-sortness – see rules (50), (51).

2. The resulting linear form represents a regular term modulo fine similarity which partially captures ACIZ-properties of the regular union $\_+\_$. Moreover, some other equational properties of the regular algebra are captured by this t.r.s. – cf. the rules (38)–(42).

3. Further rules can be added to the t.r.s. to capture more equational properties and to make the resulting linear forms more compact; Proposition. 2.5 is a correctness criterion for such an extension. E.g., the following rules

$$lf(a^* \cdot a) = lf(a) \odot a^*, \tag{54}$$

$$lf(a_1 \cdot a_1^*) = lf(a_1) \odot a_1^*, \tag{55}$$

which are *specializations* of (51), can be added to the t.r.s. due to Kleene star properties. Unlike (51), the right-hand sides of these rules do not contain an application of $lf(\_)$ to the second component of concatenation in the argument; hence the resulting linear form contains fewer monomials (this point is important for reducing the size of NFA constructed by our algorithm presented in Section 4). However, these rules would increase the computational complexity of the function $lf(\_)$ (because the arguments of these rules are non-linear patterns) and for this reason we donot include them in the definition.

4. The t.r.s. involves one associative-commutative operation, $\_ \cup \_$. It does not occur in the patterns of the rules defining $lf(\_)$, but appears in one pattern in the definition of $\_ \odot \_$, as well as in the extra rules $l \cup l \to l$, $l \cup \emptyset \to l$. This does not mean, however, that one has to involve AC-matching[8] when computing $lf(t)$ for a *ground* t. One should rather take an appropriate representation of finite sets and employ the following functional definition:

$$\begin{aligned} l \odot t = \ & \textbf{if } t = 0 \textbf{ then } \emptyset \\ & \textbf{else if } t = \lambda \textbf{ then } l \\ & \quad \textbf{else } \{\langle x, p \cdot t\rangle \mid \langle x, p\rangle \in l \wedge p \neq \lambda \wedge p \neq 0\} \cup \{\langle x, t\rangle \mid \langle x, \lambda\rangle \in l\} \\ & \textbf{fi fi} \end{aligned}$$

which is a logical consequence of (38)–(44). This allows us to compute $lf(\_)$ in time $O(n^2)$; the exact complexity depends on the representation of the data involved.

Now we come to the central definition of this paper.

Definition 2.8. [Partial derivatives]

Given a regular term t and a letter $x \in \mathscr{A}$, 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 $\partial_x : \mathscr{T}_{Reg} \to \textbf{SetReg}$, which returns a set of all (non-zero) partial derivatives of its argument w.r.t. x, as follows:

$$\partial_x(t) = \{p \in \mathscr{T}_{Reg} \setminus \{0\} \mid \langle x, p\rangle \in lf(t)\} \tag{56}$$

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

$$\partial_\lambda(t) = \{t\}, \tag{57}$$

$$\partial_{wx}(t) = \partial_x(\partial_w(t)), \tag{58}$$

$$\partial_w(R) = \bigcup_{r \in R} \partial_w(r), \tag{59}$$

[8] Which is NP-complete [4].

$$\partial_W(t) = \bigcup_{w \in W} \partial_w(t) \tag{60}$$

Each element of the set $\partial_w(t)$ is called a *partial derivative of* t *w.r.t.* w. A *proper* partial derivative of t is one w.r.t. a non-empty word.

Note that the function $\partial_w(\_)$ is monotone on $\mathscr{P}(\mathscr{T}_{Reg})$ w.r.t. set inclusion.

Example 2.9. Let us compute partial derivatives of the term $t = x^*(xx + y)^*$ from Example 2.6. For the sake of clarity, we use the shorthand $r = (xx + y)^*$. Then $t = x^* \cdot r$ and the linear form of t, computed in Example 2.6, can be written down as follows:

$$lf(t) = \{\langle x,\ t\rangle,\ \langle x,\ x \cdot r\rangle,\ \langle y,\ r\rangle\}.$$

Hence we have (according to Definition 2.8):

$$\partial_x(t) = \{t,\ x \cdot r\} = \{x^*(xx + y)^*,\ x(xx + y)^*\},$$
$$\partial_y(t) = \{r\} = \{(xx + y)^*\}.$$

Thus, there are two partial derivatives of t w.r.t. x and one w.r.t. y. To compute further partial derivatives of t, we need linear forms of the obtained partial derivatives. One can check that these are as follows:

$$lf(x \cdot r) = \{\langle x,\ r\rangle\}, \qquad lf(r) = \{\langle x,\ x \cdot r\rangle,\ \langle y,\ r\rangle\},$$

hence

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

and so on.

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

Proposition 2.10. *For any term* $t \in \mathscr{T}_{Reg}$ *and word* $w \in \mathscr{A}^*$, *the following holds*:

$$\mathscr{L}(\partial_w(t)) = w^{-1}\mathscr{L}(t). \tag{61}$$

(*In particular, any partial derivative* $p \in \partial_w(t)$ *denotes a subset of the left quotient* $w^{-1}\mathscr{L}(t)$.)

Proof. We proceed by induction on w. The case $w = \lambda$ is obvious. Let w be a letter $x \in \mathscr{A}$. The equation

$$\mathscr{L}(t) = \mathscr{L}(o(t)) \cup \bigcup_{\langle y,\ p\rangle \in lf(t)} \{y\} \cdot \mathscr{L}(p), \tag{62}$$

follows directly from Proposition. 2.5; hence we have:

$$\begin{aligned} x^{-1}\mathscr{L}(t) &= x^{-1}\Big(\bigcup_{\langle y,\ p\rangle \in lf(t)} \{y\} \cdot \mathscr{L}(p)\Big) = \bigcup_{\langle x,\ p\rangle \in lf(t)} \mathscr{L}(p) \\ &= \bigcup_{p \in \partial_x(t)} \mathscr{L}(p) = \mathscr{L}(\partial_x(t)). \end{aligned} \tag{63}$$

The general case follows from (58) and (63) by the inductive argument: assuming (61) holds for a given w, we obtain:

$$\begin{aligned} (wx)^{-1}\mathscr{L}(t) &= x^{-1}(w^{-1}\mathscr{L}(t)) = x^{-1}\mathscr{L}(\partial_w(t)) = x^{-1}\Big(\bigcup_{p \in \partial_w(t)} \mathscr{L}(p)\Big) \\ &= \bigcup_{p \in \partial_w(t)} x^{-1}\mathscr{L}(p) \bigcup_{p \in \partial_w(t)} \mathscr{L}(\partial_x(p)) = \mathscr{L}\Big(\bigcup_{p \in \partial_w(t)} \partial_x(p)\Big) \\ &= \mathscr{L}(\partial_x(\partial_w(t))) = \mathscr{L}(\partial_{wx}(t)). \end{aligned} \tag{64}$$

(here we have used known properties of left quotients). □

Corollary 2.11. *For any $t \in \mathscr{T}_{Reg}$, $w \in \mathscr{A}^*$ the following equation holds in the algebra* **Reg**$[\mathscr{A}]$:

$$w^{-1}t = \Sigma\partial_w(t). \tag{65}$$

Proof. Both sides denote the left quotient $w^{-1}\mathscr{L}(t)$. □

Thus, the elements of the set $\partial_w(t)$ are, in a sense, "parts" of the derivative $w^{-1}t$, hence their name.

Example 2.12. Let us take again the term $t = x^*(xx + y)^*$ used in the previous examples. According to Definition 1.1, we obtain the following derivative of t w.r.t. x computed modulo ACI-axioms:

$$\begin{aligned} x^{-1}t &= x^{-1}x^*(xx + y)^* = (x^{-1}x^*) \cdot (xx + y)^* + x^{-1}(xx + y)^* \\ &= (x^{-1}x) \cdot x^* \cdot (xx + y)^* + x^{-1}(xx + y) \cdot (xx + y)^* \\ &= \lambda \cdot x^*(xx + y)^* + (x^{-1}(xx) + x^{-1}y) \cdot (xx + y)^* \\ &= \lambda \cdot x^*(xx + y)^* + ((x^{-1}x) \cdot x + 0) \cdot (xx + y)^* \\ &= \lambda \cdot x^*(xx + y)^* + (\lambda \cdot x + 0) \cdot (xx + y)^*. \end{aligned}$$

Using further SR-axioms (4)–(7), the result can be simplified to the expression

$$x^*(xx + y)^* + x(xx + y)^*$$

which is obviously equivalent to $\Sigma\partial_x(t)$. In the same manner, one can compute $y^{-1}t$ and check that it is equivalent to $\Sigma\partial_y(t)$. Note that Definition 1.1 suggests that the

derivatives $x^{-1}t$, $y^{-1}t$ are to be computed separately, while the terms $\Sigma\partial_x(t)$ and $\Sigma\partial_y(t)$ can be obtained *simultaneously* through the linear form $lf(t)$ (cf. Examples 2.6 and 2.9).

Remark 2.13. To make the relation between partial derivatives and derivatives precise, it is worth noting that, in general, the terms $w^{-1}t$ and $\Sigma\partial_w(t)$ may be different even modulo the axioms (1)–(7). For example, consider the derivative

$$x^{-1}(xy+x)^* = (y+\lambda)(xy+x)^*,$$

computed modulo these axioms, and compare it with

$$\Sigma\partial_x((xy+x)^*) = \Sigma\{\ y(xy+x)^*,\ (xy+x)^*\ \} = y(xy+x)^* + (xy+x)^*.$$

To obtain the latter from the former, one has to use additionally the distributivity axiom (9).

Now we turn to the study of further properties of partial derivatives and of sets of partial derivatives.

3. Properties of partial derivatives

First we present a set of equations characterizing the function $\partial_x(\_)$.

Proposition 3.1. *For any regular terms* $a, b \in \mathcal{T}_{Reg}$, $a_0 \in \mathcal{T}_{Reg0}$, $a_1 \in \mathcal{T}_{Reg1}$ *and letters* $x,\ y \in \mathcal{A}$ *the following equations hold*:

$$\partial_x(0) = \emptyset \tag{66}$$

$$\partial_x(\lambda) = \emptyset \tag{67}$$

$$\partial_x(y) = \textbf{if}\ \ x = y\ \ \textbf{then}\ \ \{\lambda\}\ \textbf{else}\ \ \emptyset\ \textbf{fi} \tag{68}$$

$$\partial_x(a+b) = \partial_x(a) \cup \partial_x(b) \tag{69}$$

$$\partial_x(a^*) = \partial_x(a) \cdot a^* \tag{70}$$

$$\partial_x(a_0 \cdot b) = \partial_x(a_0) \cdot b \tag{71}$$

$$\partial_x(a_1 \cdot b) = \partial_x(a_1) \cdot b \cup \partial_x(b). \tag{72}$$

Proof. Just apply $\partial_x(\_)$ to both sides of the corresponding equations of Definition 2.4. □

Remark 3.2. Equations (66)–(72) form a complete recursive definition of the function $\partial_x(\_)$ which is an alternative to Definition 2.8. In practice, however, it is better to compute partial derivatives through the function $lf(\_)$ which allows one to get the whole set $\partial_{\mathcal{A}}(t)$ in one pass over t.

In the following lemma, we present several key facts about the function $\partial_w(\_)$. Here we use an auxiliary notation for the set of all non-empty suffixes of a word w:

$$Suf(w) = \{v \in \mathscr{A}^+ \mid \exists u \in \mathscr{A}^* : u \cdot v = w\} \tag{73}$$

Note that $Suf(wx) = Suf(w) \cdot x \cup \{x\}$.

Lemma 3.3. *For any regular terms a, b and word $w \in \mathscr{A}^+$ the following equation and inclusions hold:*

$$\partial_w(a + b) = \partial_w(a) \cup \partial_w(b) \tag{74}$$

$$\partial_w(a \cdot b) \subseteq \partial_w(a) \cdot b \cup \bigcup_{v \in Suf(w)} \partial_v(b) \tag{75}$$

$$\partial_w(a^*) \subseteq \bigcup_{v \in Suf(w)} \partial_v(a) \cdot a^*. \tag{76}$$

Proof. 1. Eq. (74) follows directly from (69) by induction on w using the following obvious property of $\partial_x(\_)$:

$$\partial_x(R_1 \cup R_2) = \partial_x(R_1) \cup \partial_x(R_2) \tag{77}$$

for all $R_1, R_2 \in \mathbf{SetReg}$.

2. To prove (75), first observe the following inclusion which follows from (71) and (72):

$$\partial_x(R \cdot b) \subseteq \partial_x(R) \cdot b \cup \partial_x(b) \tag{78}$$

for all $b \in \mathscr{T}_{Reg}$, $R \in \mathbf{SetReg}$.

Now we prove the inclusion (75) by induction on w. The base case, when $w \in \mathscr{A}$, follows from (78). Assuming that the inclusion holds for some $w \in \mathscr{A}^+$, we prove it for $w' = w \cdot x$ (where $x \in \mathscr{A}$):

$$\begin{aligned}
\partial_{wx}(a \cdot b) &= \partial_x(\partial_w(a \cdot b)) \subseteq \partial_x(\partial_w(a) \cdot b \cup \bigcup_{v \in Suf(w)} \partial_v(b)) \\
&= \partial_x(\partial_w(a) \cdot b) \cup \bigcup_{v \in Suf(w)} \partial_x(\partial_v(b)) \\
&\subseteq \partial_x(\partial_w(a)) \cdot b \cup \partial_x(b) \cup \bigcup_{v \in Suf(w)} \partial_{vx}(b) \\
&= \partial_{w \cdot x}(a) \cdot b \cup \bigcup_{v \in Suf(wx)} \partial_v(b).
\end{aligned} \tag{79}$$

This proves (75).

3. We prove the inclusion (76) by induction on $w \in \mathscr{A}^+$. The base case follows from (70). Assuming that the inclusion holds for some $w \in \mathscr{A}^+$, we consider it for $w' = w \cdot x$:

$$\partial_{wx}(a^*) = \partial_x(\partial_w(a^*)) \subseteq \partial_x(\bigcup_{v \in Suf(w)} \partial_v(a) \cdot a^*)$$

$$
\begin{aligned}
&= \bigcup_{v \in Suf(w)} \partial_x(\partial_v(a) \cdot a^*) \\
&\subseteq \bigcup_{v \in Suf(w)} (\partial_x(\partial_v(a)) \cdot a^* \cup \partial_x(a^*)) \\
&= (\bigcup_{v \in Suf(w)} \partial_{vx}(a) \cdot a^*) \cup \partial_x(a) \cdot a^* \\
&= \bigcup_{v \in Suf(wx)} \partial_v(a) \cdot a^*. \qquad (80)
\end{aligned}
$$

(we have used (78) on the way). This proves (76) and the lemma. □

This lemma leads to important consequences presented in the following two theorems. For a regular term t, $\|t\|$ denotes the number of all occurrences of letters from the alphabet $\mathscr{A}$ appearing in t (we call $\|t\|$ the *alphabetic width* of t). Let $\mathscr{PD}(t)$ stand for the set $\partial_{\mathscr{A}^*}(t)$ of all (syntactically different) partial derivatives of t. The first theorem says that this set is finite and provides a nice upper bound on its cardinality.

Theorem 3.4. *For any $t \in \mathscr{T}_{Reg}$ the following inequalities hold*:

$$|\partial_{\mathscr{A}^+}(t)| \leqslant \|t\|, \qquad (81)$$

$$|\mathscr{PD}(t)| \leqslant \|t\| + 1. \qquad (82)$$

Proof. Since $\mathscr{PD}(t) = \partial_\lambda(t) \cup \partial_{\mathscr{A}^+}(t)$, the first inequality implies the second one, so it suffices to prove (81). We proceed by induction on the structure of t.

Base cases, when t is 0, λ, or a letter from $\mathscr{A}$, are obvious. Now let us assume that (81) holds for some $a, b \in \mathscr{T}_{Reg}$ and consider three subcases.

Case 1: $t = a + b$. Then we have

$$\partial_{\mathscr{A}^+}(a+b) = \bigcup_{w \in \mathscr{A}^+} \partial_w(a+b) = \bigcup_{w \in \mathscr{A}^+} (\partial_w(a) \cup \partial_w(b)) = \partial_{\mathscr{A}^+}(a) \cup \partial_{\mathscr{A}^+}(b),$$

hence

$$
\begin{aligned}
|\partial_{\mathscr{A}^+}(a+b)| &= |\partial_{\mathscr{A}^+}(a) \cup \partial_{\mathscr{A}^+}(b)| \leqslant |\partial_{\mathscr{A}^+}(a)| + |\partial_{\mathscr{A}^+}(b)| \\
&\leqslant \|a\| + \|b\| = \|a+b\|.
\end{aligned}
$$

Case 2: $t = a \cdot b$. Then we have

$$
\begin{aligned}
\partial_{\mathscr{A}^+}(a \cdot b) &= \bigcup_{w \in \mathscr{A}^+} \partial_w(a \cdot b) \subseteq \bigcup_{w \in \mathscr{A}^+} (\partial_w(a) \cdot b \cup \bigcup_{v \in Suf(w)} \partial_v(b)) \\
&= (\bigcup_{w \in \mathscr{A}^+} \partial_w(a)) \cdot b \cup \bigcup_{w \in \mathscr{A}^+} \bigcup_{v \in Suf(w)} \partial_v(b) = \partial_{\mathscr{A}^+}(a) \cdot b \cup \partial_{\mathscr{A}^+}(b),
\end{aligned}
$$

hence

$$
\begin{aligned}
|\partial_{\mathscr{A}^+}(a \cdot b)| &\leqslant |\partial_{\mathscr{A}^+}(a) \cdot b \cup \partial_{\mathscr{A}^+}(b)| \leqslant |\partial_{\mathscr{A}^+}(a) \cdot b| + |\partial_{\mathscr{A}^+}(b)| \\
&\leqslant |\partial_{\mathscr{A}^+}(a)| + |\partial_{\mathscr{A}^+}(b)| \leqslant \|a\| + \|b\| = \|a \cdot b\|.
\end{aligned}
$$

Case 3: $t = a^*$. Then we have

$$\partial_{\mathscr{A}^+}(a^*) = \bigcup_{w\in\mathscr{A}^+} \partial_w(a^*) \subseteq \bigcup_{w\in\mathscr{A}^+} \bigcup_{v\in Suf(w)} \partial_v(a)\cdot a^* = \partial_{\mathscr{A}^+}(a)\cdot a^*,$$

hence

$$|\partial_{\mathscr{A}^+}(a^*)| \leqslant |\partial_{\mathscr{A}^+}(a)\cdot a^*| = |\partial_{\mathscr{A}^+}(a)| \leqslant \|a\| = \|a^*\|.$$

This ends the proof of (81) and of the theorem. □

Corollary 3.5. *For any set of words $W \subseteq \mathscr{A}^*$, the set $\partial_W(t)$ is finite and the following inequality holds:*

$$|\partial_W(t)| \leqslant \|t\| + 1. \tag{83}$$

Proof. Immediate, since $W \subseteq \mathscr{A}^*$ implies $\partial_W(t) \subseteq \partial_{\mathscr{A}^*}(t)$. □

Remark 3.6. Recall that a left quotient of a language $L \subseteq \mathscr{A}^*$ w.r.t. a language $W \subseteq \mathscr{A}^*$, written $W^{-1}L$, is defined as a union $\bigcup_{w\in W} w^{-1}L$. It is known that if L is a regular language, such a quotient is a regular language too (for any W).

Conway [11, p. 43] introduced a corresponding generalization of derivatives, called an *event derivative of t w.r.t. W*, and proved that the set of all event derivatives of t (w.r.t. all possible W) is finite [11, p. 43, Theorem 3] – this implies one direction of Kleene's main theorem (that any regular language is recognizable).

Now we note that the regular term $\Sigma\partial_W(t)$ is equivalent (in $\mathbf{Reg}[\mathscr{A}]$) to an event derivative of t w.r.t. W; thus Conway's theorem is a consequence of our Theorem 3.4. Moreover, we obtain the upper bound $2^{\|t\|+1}$ on the cardinality of the set of all event derivatives of t, that improves the upper bound given in [11, Theorem 4, p. 43]. □

Example 3.7. Using the notation of Example 2.9, we have $\|t\| = 4$ and

$$\mathscr{PD}(t) = \{t,\ x\cdot r,\ r\}.$$

At the same time, it can be shown that there are five SR-dissimilar word derivatives of t and that the set of all syntactically distinct word derivatives of t, computed by Definition 1.1, is infinite. □

Our second theorem clarifies the internal structure of partial derivatives.

Theorem 3.8. *Given a regular term $t \in \mathscr{T}_{Reg}$, any partial derivative of t is either λ, or a subterm of t, or a concatenation $t_0 \cdot t_1 \cdot \cdots \cdot t_n$ of several such subterms where n is not greater than the number of occurrences of concatenation and Kleene star appearing in t.*

Proof. The proof is by induction on t and has the same structure as in the previous theorem. The base cases, when t is 0, λ, or $y \in \mathscr{A}$, are obvious. Assuming that the

statement of the theorem is valid for regular terms a and b, the equation and inclusions proved in Lemma 3.3 imply that it is valid for $t = a + b$, $t = a \cdot b$, and $t = a^*$. Also, the inclusions (75) and (76) imply that each occurrence of concatenation or Kleene star in t gives rise to at most one extra concatenation in partial derivatives of t – this proves the upper bound formulated in the second statement of the theorem. □

It follows from the two theorems above that the set $\mathscr{PD}(t)$ 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 the number of lists is bounded by $\|t\|+1$. In the next section, it will be shown that this data structure is virtually a set of states of an NFA recognizing the language $\mathscr{L}(t)$.

4. Finite automaton constructions using partial derivatives

4.1. From regular expressions to small NFA's

Here we present a new algorithm turning a regular term t into an NFA having at most $\|t\| + 1$ states. The following theorem describes our construction.

Theorem 4.1. *Given a regular term t over an alphabet $\mathscr{A}$, let an automaton* **M** *over $\mathscr{A}$ have the set of states $M = \mathscr{PD}(t)$, the inital state $\mu_0 = t$, the transition function τ is defined by*

$$\tau(p,\ x) = \partial_x(p) \tag{84}$$

for all $p \in \mathscr{PD}(t)$, $x \in \mathscr{A}$, and the set of final states

$$F = \{p \in \mathscr{PD}(t) \mid o(p) = \lambda\}.$$

Then **M** *recognizes the language $\mathscr{L}(t)$.*

Proof. First we notice that (84) implies the equation

$$\tau(p,\ w) = \partial_w(p) \tag{85}$$

for any state $p \in M$ and word $w \in \mathscr{A}^*$ (proof by straightforward induction on w).

Now we prove that for any $p \in M$ the language $\mathscr{L}_M(p)$ accepted by the state p is equal to the language denoted by the partial derivative p, i.e.

$$w \in \mathscr{L}(p) \ \Leftrightarrow\ w \in \mathscr{L}_M(p) \tag{86}$$

for any word $w \in \mathscr{A}^*$. Indeed, $w \in \mathscr{L}(p)$ is equivalent to $o(w^{-1}p) = \lambda$, and hence (by Prop. 2.10) to $o(\Sigma\partial_w(p)) = \lambda$. The latter means there is at least one partial derivative in $\partial_w(p)$ with non-empty constant part, i.e.

$$\partial_w(p) \cap F \neq \emptyset. \tag{87}$$

On the other hand, $w \in \mathscr{L}_M(p)$ is by definition equivalent to $\tau(p,\ w) \cap F \neq \emptyset$ and hence to (87); this proves (86). In particular, the language $\mathscr{L}_M(\mu_0)$ recognized by $\mathbf{M}$ is equal to $\mathscr{L}(t)$. □

Let us consider how to compute the set $\mathscr{PD}(t)$ and the transition function τ (the set F can be obtained in an obvious way). A possible way to do this is to use of the following iterative procedure:

$$\langle \mathscr{PD}_0,\ \Delta_0,\ \tau_0 \rangle := \langle\ \{t\},\ \{t\},\ \emptyset \rangle, \tag{88}$$

$$\tau_{i+1} := \tau_i \cup \{\langle p,\ x,\ q\rangle \mid p \in \Delta_i \wedge \langle x,\ q\rangle \in lf(p)\}, \tag{89}$$

$$\Delta_{i+1} := \bigcup_{p \in \Delta_i} \{q \mid \langle x,\ q\rangle \in lf(p) \wedge q \notin \mathscr{PD}_i\}, \tag{90}$$

$$\mathscr{PD}_{i+1} := \mathscr{PD}_i \cup \Delta_{i+1} \tag{91}$$

for $i = 0, 1, \ldots$. Here τ is represented as a finite subset of $M \times \mathscr{A} \times M$ (i.e., a transition relation). The set Δ_{i+1} consists of new partial derivatives, i.e., of those appearing at the step $i+1$, but not at any of the previous steps. After at most $\|t\|$ iterations, the set Δ_{i+1} becomes empty – then $\mathscr{PD}_i$ and τ_{i+1} contain the needed results.

Remark 4.2. This gives a constructive solution, but its efficiency suffers from the necessity of testing the (negation of the) membership $q \in \mathscr{PD}_i$ which requires checking syntactic equality of partial derivatives. In the worst case, a partial derivative of a regular term t of the size n may have a size up to $O(n^2)$ (recall Theorem 3.8), there are up to $\|t\|$ elements in $\mathscr{PD}_i$, and the membership test has to be performed up to $\|t\|$ times. This leads to the $O(\|t\|^2 \cdot n^2)$ worst-case time complexity of the algorithm. It is a topic for further research to find a more efficient implementation of the construction presented in Theorem 4.1.

4.2. From regular expressions to DFA's: improvements to Brzozowski's algorithm

The NFA-construction presented above can be easily transformed into a procedure constructing DFA's. To see the idea of this transformation, note that Eq. (85) implies that the set $\partial_w(p)$ represents a "deterministic state" corresponding to the result of all possible transitions from the state p on the word w. Substituting p in (84) by such a deterministic state, we derive the equation

$$\tau(\partial_w(p),\ x) = \partial_x(\partial_w(p)) = \partial_{wx}(p)$$

which defines the corresponding "deterministic transitions".

Thus, the set

$$\mathscr{DD}(t) = \{\partial_w(t) \mid w \in \mathscr{A}^*\} \tag{92}$$

has to be taken as the set of states of the DFA, the initial state is the singleton $\{t\}$, the transition function is defined by

$$\tau(P,\ x) = \partial_x(P) \tag{93}$$

for all $P \in \mathscr{DD}(t), x \in \mathscr{A}$, and the set of final states is

$$F = \{P \in \mathscr{DD}(t) \mid o(\Sigma P) = \lambda\}. \tag{94}$$

Proposition 4.3. *The automaton* $\langle \mathscr{DD}(t),\ \tau,\ \{t\},\ F\rangle$ *presented above recognizes* $\mathscr{L}(t)$.

Proof. This follows easily from the fact that each deterministic state $P = \partial_w(t)$ represents a left quotient $w^{-1}\mathscr{L}(t)$ (recall Proposition. 2.10). □

The relation between partial derivatives and derivatives, given in Corollary 2.11, readily demonstrates that this construction is just a modification of the one of Brzozowski where each derivative $w^{-1}t$ is replaced by a corresponding set $\partial_w(t)$ of partial derivatives. Note, however, that this leads to several practical advantages:

1. One does not have to compute and to keep in memory the whole set $\mathscr{D}_E(t)$ of E-dissimilar derivatives. More precisely, one can use $\mathscr{PD}(t)$ as a *basis* for the set $\mathscr{DD}(t)$: each deterministic state $\partial_w(t)$ should be represented as a set of *references* to corresponding elements in $\mathscr{PD}(t)$.

2. Computing the set $\mathscr{DD}(t)$ represented as suggested above, one compares its elements just as sets of references, rather than checks equivalence of derivatives modulo $\mathscr{E}(ACI)$ or any other non-trivial congruence; clearly, the former test requires much less time than the latter.

3. The components $\tau(P,\ x)$ of the transition function should be computed through the function $lf(\_)$ which gives a *whole tuple* of transitions $\{\langle x,\ \partial_x(P)\rangle \mid x \in \mathscr{A}\}$ in one pass over P. This is more efficient than to compute separately each derivative w.r.t. $x \in \mathscr{A}$ that requires one pass over P for each x. The bigger the alphabet, the more one gains from this optimization.

Therefore, our modification can be implemented much more efficiently than the original Brzozowski construction.[9]

Our DFA-construction is also interesting from the theoretical point of view, because it demonstrates that the relation between partial derivatives and derivatives is quite similar to the well known relation between NFA's and DFA's provided by the classical subset construction [27]. To see this, suppose that a regular term t is turned into an NFA as described in Theorem 4.1. 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 $\mathscr{PD}(t)$. On the other hand, the same DFA – with the set of states $\mathscr{DD}(t)$ – can be obtained directly from t as described in Proposition. 4.3.

[9] Of course, one should bear in mind that either of these constructions may produce an output of an exponential size.

5. Implementation and examples

We have used the algebraic programming language OBJ3 [16] to develop a prototype implementation of our algorithms for computing partial derivatives and constructing NFA's.

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

$$\mu := o(\mu) + x_1 \cdot \mu_1 + \cdots + x_k \cdot \mu_k \tag{95}$$

for each state $\mu \in M$ where $x_i \in \mathscr{A}$ and $\mu_i \in \tau(\mu, x_i)$, $i = 1 \ldots k$ (see, e.g., [9]). Here $o(\mu)$ is the *output* of the state μ equal to λ if $\mu \in F$, or to 0 otherwise – in the latter case it is omitted from the sum. The components $x_i \cdot 0$ are also omitted from the right-hand sides of state equations (thus, the resulting set of equations represents in general an incomplete NFA).

Our OBJ3-program consists of several order-sorted term-rewriting systems ("objects" in terms of OBJ3), some of them modulo associativity and commutativity of the set union $\cup$. Some of the objects represent the term algebra $\mathscr{T}_{Reg}$ and the algebras **SetReg** and **Lin** as abstract data types; the others implement Definition 2.4 and the Eqs. (88)–(91). The main function of the program takes a regular term as input and returns a set of state equations, representing a resulting NFA, and a set of partial derivatives corresponding to the states of the automaton.

Below we present and discuss several examples of NFA's constructed by this program. The main goal is to compare our NFA's with those obtained by some known algorithms from [5, 10, 14, 22, 31], Let us recall some characteristics of the latters.

Given a regular term r of size n, the algorithm by Thomson [31] (cf. also [17, 18]) produces an NFA with $O(n)$ states and edges; this NFA may have λ-transitions. The algorithms by McNaughton and Yamada [22], Glushkov [14], and Berry and Sethi [5] produce an NFA (without λ-transitions) whose non-initial states correspond to the occurences of letters in the input regular expression, i.e., the number of states is $\|r\|+1$ (that can be arbitrarily smaller than n). As for the number of edges, it may be quadratic in n.

Example 5.1. This is a running example from [5] – the regular expression

$$t = (ab + b)^* ba$$

on the alphabet $\mathscr{A} = \{a, b\}$. Our algorithm turns this expression into the following NFA with four states and five edges:

State equations	Partial derivatives
$1 := a \cdot 2 + b \cdot 1 + b \cdot 3$	$1 = (ab + b)^* ba$
$2 := b \cdot 1$	$2 = b(ab + b)^* ba$
$3 := a \cdot 4$	$3 = a$
$4 := \lambda$	$4 = \lambda$

Berry and Sethi's algorithm turns this expression into an NFA with six states (since $\|t\| = 5$) and 11 edges, cf. [5, p.121]. Three different states in the latter NFA are actually equivalent: the starting state, and two further states corresponding to the first and second occurrences of b in t. This equivalence is easily detected by our algorithm, for these three states correspond to the same partial derivative $1 = t$ (actually, to the same subterm of t) – this explains why our NFA is smaller.

Example 5.2. This is a running example from [10] – the regular expression

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

on the alphabet $\mathscr{A} = \{a, b\}$. Our program turns it into the following NFA with four states and five edges:

State equations	Partial derivatives
$1 := a \cdot 1 + b \cdot 1 + a \cdot 2$	$1 = (a + b)^* abb$
$2 := b \cdot 3$	$2 = bb$
$3 := b \cdot 4$	$3 = b$
$4 := \lambda$	$4 = \lambda$

In [10], the expression was first turned into McNaughton and Yamada's NFA with six states and eleven edges, then this NFA was transformed into a so-called *compressed normalized NFA* with five states and six edges through several non-trivial optimizations. Our algorithm produces a smaller NFA without any additional optimization.

Example 5.3. Let us consider a typical example of a regular expression appearing in the formal syntax description of programming languages – identifiers:

$$t = \mathsf{L} \cdot (\mathsf{L} + \mathsf{D})^*$$

where $\mathsf{L} = A + a + \cdots + Z + z$ stands for letters and $\mathsf{D} = 0 + 1 + \cdots + 9$ stands for digits (thus, we use the alphabet $\mathscr{A} = \{A, a, B, b, \ldots, Z, z, 0, 1, \ldots, 9\}$ in this example).

One can easily see that McNaughton and Yamada's NFA for this expression would have 115 states, since the alphabetic width of the expression is 114; Thompson's NFA would be even larger. In contrast to this, our algorithm turns t into the following NFA (actually, a DFA) with only two states, since there are only two syntactically different partial derivatives of t:

State equations	Partial derivatives
$1 := A \cdot 2 + a \cdot 2 + \ldots + z \cdot 2$	$1 = \mathsf{L} \cdot (\mathsf{L} + \mathsf{D})^*$
$2 := \lambda + A \cdot 2 + \ldots + z \cdot 2 + 0 \cdot 2 + \ldots + 9 \cdot 2$	$2 = (\mathsf{L} + \mathsf{D})^*$

This example demonstrates that in some cases our NFA's may be *arbitrarily* smaller than those obtained by the classical algorithms.

Example 5.4. Now we give an example of the opposite kind, when our algorithm does not improve the results of classical ones. Consider the following regular expression

$$t = a^*b^* \ldots z^*$$

Our algorithm turns it into the following NFA with 26 states and $26 \cdot 27/2 = 351$ edges:

State equations	Partial derivatives
$1 := \lambda + a \cdot 1 + b \cdot 2 + \ldots + z \cdot 26$	$1 = a^*b^*c^* \ldots z^*$
$2 := \lambda + b \cdot 2 + c \cdot 3 + \ldots + z \cdot 26$	$2 = b^*c^* \ldots z^*$
$\ldots$	$\ldots$
$26 := \lambda + z \cdot 26$	$26 = z^*$

The size of McNaughton and Yamada's NFA for t is of the same order of magnitude (since $\|t\| = 26$) and Thompson's NFA has much fewer edges. The latter is due to the presence of λ-transitions in Thompson's NFA's (cf. [30] for an explanation of this effect, and [10] for further comparisons of Thompson's and McNaughton and Yamada's NFA's).

Example 5.5. To present an example from another area, let us consider the following regular expression [10] which is a typical one appearing in the study of some properties of word-rewriting systems with variables [20, 21]:

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

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

State equations	Partial derivatives
$1 := a \cdot 1 + b \cdot 1 + b \cdot 2 + b \cdot 3$	$1 = t$
$2 := a \cdot 4$	$2 = abab(a + b)^*bab(a + b)^*$
$3 := b \cdot 5$	$3 = ba(a + b)^*bab(a + b)^*$
$4 := b \cdot 6$	$4 = bab(a + b)^*bab(a + b)^*$
$5 := a \cdot 7$	$5 = a(a + b)^*bab(a + b)^*$
$6 := a \cdot 8$	$6 = ab(a + b)^*bab(a + b)^*$
$7 := a \cdot 7 + b \cdot 7 + b \cdot 9$	$7 = (a + b)^*bab(a + b)^*$
$8 := b \cdot 7$	$8 = b(a + b)^*bab(a + b)^*$
$9 := a \cdot 10$	$9 = ab(a + b)^*$
$10 := b \cdot 11$	$10 = b(a + b)^*$
$11 := \lambda + a \cdot 11 + b \cdot 11$	$11 = (a + b)^*$

Note that $\|t\| = 22$, so McNaughton and Yamada's NFA for t would have 23 states; the size of Thompson's NFA for t would be even larger.

6. Concluding remarks

Since any non-empty word has a head and a tail, any language L over a finite alphabet $\mathscr{A}$ can be decomposed into a finite union of components of the form $\{x\}\cdot$

[10] Suggested by Gregory Kucherov.

$(x^{-1}L)$ for each $x \in \mathscr{A}$, possibly plus the empty word. In the case when L is regular, one can iterate this decomposition for the obtained left quotients $x^{-1}L$ and in this way come, after a finite number of steps, to an "abstract" DFA recognizing the language L.

As a matter of fact, derivatives of a regular expression r are constructive representations of corresponding left quotients of the language $\mathscr{L}(r)$. Therefore, it is quite natural that derivatives turned out to be a right tool for an effective representation of the above abstract construction – as was invented by Brzozowski. The main problem with this approach was the fact that infinitely many different derivatives may represent the same left quotient. Brzozowski solved this problem through the notion of similarity: the set of ACI-dissimilar word derivatives of any regular expression was proved to be finite.

In the present paper, we have provided an extension of this framework to the nondeterministic case through the new notion of partial derivative. The main contribution of the present paper is the notion of partial derivative of a regular expression. We have given a constructive definition of this notion and proved that it has several interesting features:

- Partial derivatives can be easily computed and efficiently represented. Any partial derivative of a regular expression r is either λ, or a list (of a bounded length) of subterms of r.
- There is at most $\|r\| + 1$ syntactically distinct partial derivatives of r. Therefore, the set $\mathscr{PD}(r)$ provides a relatively small basis for the set of all (dissimilar) word derivatives, as well as for the set of all event derivatives of r: any word (or event) derivative can be represented as a finite sum of partial derivatives. The relation between $\mathscr{PD}(r)$ and the set of word derivatives of r is quite similar to the relation between NFA's and DFA's determined by the subset construction.
- A regular expression r can be turned into a relatively small NFA whose states are (represented by) partial derivatives of r. This implies that our NFA's have no more states than those produced by several known algorithms (and we have demonstrated on examples that in some cases our NFA's are substantially smaller).
- Efficiency of Brzozowski algorithm for constructing DFAs can be improved through the representation of derivatives by sets of partial derivatives. Using partial derivatives in this way can be also fruitful in other procedures involving derivatives – e.g. in those for checking equivalence of regular expressions [3, 23].

The tight structural connection between regular expressions and NFAs, provided by partial derivatives, seems to be very fruitful. Recall that a similar connection to DFAs, provided by derivatives, has been successfully employed in several studies related to the algebra of regular events. We envisage a similar application for the partial derivatives. (As an example of such an application, let us mention our recent work [1] in which we employ partial derivatives to develop a new purely algebraic procedure for checking inequalities $r \leqslant t$ of regular expressions.)

Finally, let us mention two possible directions for further research.

1. It should be investigated whether there is a more efficient implementation of the NFA construction presented in Theorem 4.1. To improve its efficiency, one should find a way to compute the set $\mathscr{PD}(r)$ so as to avoid checking syntactic equality of partial derivatives.

2. It would be useful to find an appropriate definition of partial derivatives of *extended* regular expressions (with intersection, complementation, and other operations). Then, in particular, our NFA construction would directly extend to this class of regular expressions.

Acknowledgements

The author would like to thank Pierre Lescanne and Gregory Kucherov for helpful discussions and comments on a preliminary version of this paper.

References

[1] V.M. Antimirov, Rewriting regular inequalities, December 1994, Proc. FCT'95, to be presented at the conf. *Fundamentals of Computation Theory*, August, 1995. in:

[2] V.M. Antimirov, Partial derivatives of regular expressions and finite automata constructions, in: E.W. Mayr and C. Puech, eds., *12th Annual Symp. on Theoretical Aspects of Computer Science. Proceedings.*, Lecture Notes in Computer Science, Vol. 900 (Springer, Berlin, 1995) 455–466.

[3] V.M. Antimirov and P.D. Mosses, Rewriting extended regular expressions, *Theoret. Comput. Sci.* **143** (1995) 51–72.

[4] D. Benanav, D. Kapur and P. Narendran, Complexity of matching problems, *J. Symbolic Comput.* **3**(1 & 2) (1987) 203–216.

[5] G. Berry and R. Sethi, From regular expressions to deterministic automata, *Theoretical Comput. Sci.*, 48:117–126, 1986.

[6] J. Berstel, Finite automata and rational languages. an introduction, in: J. Pin, ed., *Finite Automata and Applications*, Lecture Notes in Computer Science, Vol. 386 (Springer, Berlin, 1987) 2–14.

[7] A. Brüggemann-Klein, Regular expressions into finite automata, *Theoret. Comput. Sci.* **120** (1993) 197–213.

[8] J.A. Brzozowski, Derivatives of regular expressions, *J. ACM* **11** (1964) 481–494.

[9] J.A. Brzozowski and E.L. Leiss, On equations for regular languages, finite automata, and sequential networks, *Theoret. Comput. Sci.* **10** (1980) 19–35.

[10] 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, eds., *Combinatorial Pattern Matching. Proceedings.* Lecture Notes in Computer Science, Vol. 644 (Springer, Berlin, 1992) 88–108.

[11] J.H. Conway, *Regular Algebra and Finite Machines* (Chapman and Hall, London, 1971).

[12] N. Dershowitz and J.-P. Jouannaud, Rewrite systems, in: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science*, Vol. B, Chap. 6. (Elsevier, Amsterdam; and MIT Press, Cambridge, MA, 1990).

[13] A. Ginzburg, A procedure for checking equality of regular expressions, *J. ACM* **14**(2) (1967) 355–362.

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

[15] J.A. Goguen and J. Meseguer, Order-sorted algebra I: Equational deduction for multiple inheritance, overloading, exceptions and partial operations, *Theoret. Comput. Sci.* **105** (1992) 217–273.

[16] J.A. Goguen and T. Winkler, Introducing OBJ3. Technical Report SRI-CSL-88-9, Computer Science Lab., SRI International, 1988.

[17] R.M. Goldberg, Finite state automata from regular expression trees, *Computer J.* **36**(7) (1993) 623–630.

[18] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages, and Computation* (Addison-Wesley, Reading, MA, 1979).

[19] D. Krob, Differentiation of K-rational expressions, *Internat. J. Algebra Comput.* **2**(1) (1992) 57–87.

[20] G. Kucherov and M. Rusinowitch, Complexity of testing ground reducibility for linear word rewriting systems with variables, in: *Proc. 4th Internat. Workshop on Conditional and Typed Term Rewriting Systems*, Jerusalem (Israel), July 1994. (To appear in the LNCS series).

[21] G. Kucherov and M. Rusinowitch, On Ground-Reducibility Problem for word rewriting systems with variables, in: E. Deaton and R. Wilkerson, eds., *Proc. 1994 ACM/SIGAPP Symp. on Applied Computing*, Phoenix (USA), 1994. (ACM-Press, New York, 1995).

[22] R. McNaughton and H. Yamada, Regular expressions and state graphs for automata, *IEEE Trans. on Electronic Computers*, **9**(1) (1960) 39–47.

[23] Y. Mizoguchi, H. Ohtsuka and Y. Kawahara, A symbolic calculus of regular expressions, *Bull. Inform. Cybernet.* **22**(3–4) (1987) 165–170.

[24] J. Myhill, Finite automata and the representation of events, Technical Report WADD TR-57-624, Wright Patterson AFB, Ohio, 1957.

[25] A. Nerode, Linear automaton transformations, in: *Proc. AMS* **9** (1958) 541–544.

[26] D. Perrin, Finite automata, in: J. van Leeuwen, ed., *Handbook of Theoretical Computer Science*, Vol B, Chap. 1. (Elsevier, Amsterdam; and MIT Press, Cambridge, MA, 1990).

[27] M.O. Rabin and D. Scott, Finite automata and their decision problems, *IBM J. Res. Development* **3**(2) (1959) 114–125.

[28] V.N. Redko, On defining relations for the algebra of regular events, *Ukrain. Mat. Zh.* **16** (1964) 120–126.

[29] A. Salomaa, *Theory of Automata* (Pergamon, Oxford, 1969).

[30] S. Sippu and E. Soisalon-Soininen, *Parsing Theory. Vol. 1: Languages and Parsing*, EATCS Monographs on Theoretical Computer Science (Springer, Berlin, 1988).

[31] K. Thompson, Regular expression search algorithms, *Comm. ACM* **11**(6) (1968) 419–422.

[32] B.W. Watson, A taxonomy of finite automata construction algorithms, Computing Science Note 93/43, Eindhoven University of Technology, The Netherlands, 1993.