Rewriting Regular Inequalities

(extended abstract)

Valentin Antimirov*

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

1 Introduction

In this paper we apply algebraic specification and term-rewriting methods to the *containment problem* in the algebra **Reg**[$\mathcal{A}$] of regular events (languages) on a finite alphabet $\mathcal{A}$ formulated as follows: given two regular expressions r, t (on $\mathcal{A}$), to check if the inequality $r \leq t$ is valid in **Reg**[$\mathcal{A}$].

Standard approaches to the problem are based on translation of the expressions r, t into deterministic finite automata (DFAs); in fact, one can also do with a non-determistic one (NFA) for r. The main peculiarity of our approach is that we avoid such a translation and develop a term-rewriting system (t.r.s.) to reduce $r \leq t$ into a normal form; the latter is *false* whenever $r \leq t$ is not valid in **Reg**[$\mathcal{A}$]. This provides a purely algebraic (symbolic) decision procedure – both for the containment and word problems – that seems to be an interesting contribution *per se*.[2].

Moreover, being extended with some new rewrite rules, our t.r.s. in some cases provides derivations of polynomial size, while any algorithm based on translation of the expressions into DFAs gives rise to an exponential blow-up – we demonstrate this on examples. Of course, the worst-case complexity of our procedure is still exponential – this is not surprising, since the problem is PSPACE-complete [10, 8, 9]. At the end of the paper we provide a more general comparison of our solution with the standard ones by giving an automata-theoretic interpretation of the inference process implemented by our t.r.s.

Some ideas of the present work come from [2]. We also employ a recently introduced technique of *partial derivatives* from [1].

2 Basic Definitions and Notation

We use standard notions and notations of order-sorted algebra [7], term rewriting [5], and finite automata theory [11] without special comments.

* On leave from V.M.Glushkov Institute of Cybernetics, Kiev, Ukraine.

[2] It is worth recalling that the ground equational theory of **Reg**[$\mathcal{A}$] is not finitely based [12, 4, 6], so a (non-conditional) t.r.s. – on the signature of **Reg**[$\mathcal{A}$] – solving the word problem in the algebra does not exist. What makes our solution possible is that our t.r.s. works on regular inequalities and conjunctions of those and also involves some auxiliary operations.

Given a set X, we denote its cardinality by $|X|$, its power-set (the set of all subsets of X) by $\mathcal{P}(X)$, and the set of all *finite* subsets of X by $\mathbf{Set}[X]$. Recall that an *upper semilattice* is an algebra with a binary operation (called *join*) which satisfies the axioms of associativity, commutativity, and idempotency (*ACI-axioms*, for short). In particular, $\mathbf{Set}[X]$ forms an upper semilattice with the join $\_ \cup \_$ and the least element $\emptyset$. Sometimes we say that an operation has AC- or ACI-properties – this means the operation satisfies corresponding axioms.

Given a finite alphabet $\mathcal{A}$, let $\mathcal{A}^*$ denote the set of all finite words on $\mathcal{A}$ and also the free monoid on $\mathcal{A}$ with concatenation of words $w \cdot u$ as multiplication, and the empty word λ as the neutral element. Let $\mathcal{A}^+ = \mathcal{A} \setminus \{\lambda\}$.

The set $\mathbf{Reg}[\mathcal{A}]$ of regular events (languages) on $\mathcal{A}$ is the least subset of the power-set $\mathcal{P}(\mathcal{A}^*)$ which includes the empty set $\emptyset$, the singletons $\{\lambda\}$ and $\{\alpha\}$ for all $\alpha \in \mathcal{A}$ and is closed under the standard regular operations – concatenation $L_1 \cdot L_2$, union $L_1 \cup L_2$ and iteration (Kleene star) L^*. The set $\mathbf{Reg}[\mathcal{A}]$ together with the operations forms a *regular algebra*.

Given a regular language L, a *left quotient of L w.r.t. a word w*, written $w^{-1}L$, is the (regular) language $\{\, u \in \mathcal{A}^* \mid w \cdot u \in L \,\}$. It follows that $w \in L$ is equivalent to $\lambda \in w^{-1}L$ and that $L \subseteq L'$ implies $w^{-1}L \subseteq w^{-1}L'$ for any languages L, L' and word w.

Let $\mathbf{Reg1}[\mathcal{A}]$ be a subset of $\mathbf{Reg}[\mathcal{A}]$ consisting of all the regular languages containing the empty word λ. The complement of this subset w.r.t. $\mathbf{Reg}[\mathcal{A}]$ is denoted by $\mathbf{Reg0}[\mathcal{A}]$. These subsets, together with $\mathcal{A}$, determine an order-sorted structure on $\mathbf{Reg}[\mathcal{A}]$ with a sort $\mathcal{A}$ for the alphabet letters, sorts *Reg0* and *Reg1* for the expressions denoting elements of $\mathbf{Reg0}[\mathcal{A}]$ and $\mathbf{Reg1}[\mathcal{A}]$ respectively, and a sort *Reg* for all regular expressions. The corresponding order-sorted signature *REG* on a given alphabet $\mathcal{A} = \{\alpha_1, \alpha_2, \ldots, \alpha_k\}$ is presented below.

Signature *REG*		
sorts $\mathcal{A}$, *Reg*, *Reg0*, *Reg1*.		
subsorts $\mathcal{A} < Reg0 < Reg$; $Reg1 < Reg$.		
constants $\emptyset : Reg0$; $\lambda : Reg1$; $\alpha_1, \alpha_2, \ldots, \alpha_k : \mathcal{A}$.		
operations	$\_+\_ : Reg\ Reg \to Reg$	$\_\cdot\_ : Reg\ Reg \to Reg$
	$\_+\_ : Reg1\ Reg \to Reg1$	$\_\cdot\_ : Reg0\ Reg \to Reg0$
	$\_+\_ : Reg\ Reg1 \to Reg1$	$\_\cdot\_ : Reg\ Reg0 \to Reg0$
	$\_+\_ : Reg0\ Reg0 \to Reg0$	$\_\cdot\_ : Reg1\ Reg1 \to Reg1$
	$\_^* : Reg \to Reg1$	

Sets of ground terms on the signature *REG* of the sorts *Reg*, *Reg0*, and *Reg1* are defined in the usual way and denoted by $\mathcal{T}_{Reg}$, $\mathcal{T}_{Reg0}$, and $\mathcal{T}_{Reg1}$ correspondingly. Note that $\mathcal{T}_{Reg}$ is a disjoint union of $\mathcal{T}_{Reg0}$ and $\mathcal{T}_{Reg1}$. In what follows we call the elements of $\mathcal{T}_{Reg}$ *regular terms*. Given a regular term t, we denote by $||t||$ the number of all the occurrences of alphabet letters appearing in t and call this an *alphabetic width* of t.

Any regular term t denotes a regular language $\mathcal{L}(t)$ and this interpretation is determined by the homomorphism $\mathcal{L}(\_)$ from the term algebra $\mathcal{T}_{Reg}$ to $\mathbf{Reg}[\mathcal{A}]$

(defined in the usual way). This defines a standard interpretation of regular equations and inequalities in the regular algebra: $\mathbf{Reg}[\mathcal{A}] \models r = t$ means $\mathcal{L}(r) = \mathcal{L}(t)$ and $\mathbf{Reg}[\mathcal{A}] \models r \leq t$ means $\mathcal{L}(r) \subseteq \mathcal{L}(t)$. Recall that $\mathbf{Reg}[\mathcal{A}]$ is an upper semilattice w.r.t. the join $\_\cup\_$ and $\_\leq\_$ is the standard partial ordering associated with this semilattice.

The order-sorted presentation of $\mathbf{Reg}[\mathcal{A}]$ allows us to distinguish syntactically a class of *trivially inconsistent* inequalities of the form $a \leq b$ where $a \in \mathcal{T}_{Reg1}$, $b \in \mathcal{T}_{Reg0}$.

Next we reproduce several definitions and facts from [1].

Let **SReg** be the upper semilattice $\mathbf{Set}[\mathcal{T}_{Reg} \backslash \{\emptyset\}]$ of finite sets of non-zero regular terms. The interpretation $\mathcal{L}$ extends to **SReg** by $\mathcal{L}(R) = \bigcup_{t \in R} \mathcal{L}(t)$.

Given $R \in \mathbf{SReg}$ such that $R = \{t_1, \ldots, t_k\}$ where all t_i are syntactically distinct, we write ΣR to denote a regular term $t_1 + \ldots + t_k$ up to an arbitrary permutation of the summands, i.e., the upper occurences of $\_+\_$ in this sum are considered modulo *ACI*-axioms. To complete this, $\Sigma\emptyset$ is defined to be $\emptyset$.[3]

Definition 1. (Linear forms of regular terms) 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 $\mathbf{Lin} = \mathbf{Set}[\mathcal{A} \times \mathcal{T}_{Reg}]$ be the upper semilattice of linear forms. The function $\mathit{lf}(\_) : \mathcal{T}_{Reg} \to \mathbf{Lin}$, returning a linear form of its argument, is defined recursively by the following equations:

$$\begin{aligned} \mathit{lf}(\emptyset) = \mathit{lf}(\lambda) &= \emptyset, & \mathit{lf}(t^*) &= \mathit{lf}(t) \odot t^*, \\ \mathit{lf}(x) &= \{\langle x, \lambda \rangle\}, & \mathit{lf}(r_0 \cdot t) &= \mathit{lf}(r_0) \odot t, \\ \mathit{lf}(r + t) &= \mathit{lf}(r) \cup \mathit{lf}(t), & \mathit{lf}(r_1 \cdot t) &= \mathit{lf}(r_1) \odot t \cup \mathit{lf}(t) \end{aligned}$$

for all $x \in \mathcal{A}$, $r, t \in \mathcal{T}_{Reg}$, $r_0 \in \mathcal{T}_{Reg0}$, $r_1 \in \mathcal{T}_{Reg1}$. These equations involve a binary operation $\_\odot\_ : \mathbf{Lin} \times \mathcal{T}_{Reg} \to \mathbf{Lin}$, which is an extension of concatenation to linear forms defined recursively by the following equations:

$$\begin{aligned} l \odot \emptyset = \emptyset \odot t &= \emptyset, & \{\langle x, \lambda \rangle\} \odot t &= \{\langle x, t \rangle\}, \\ l \odot \lambda &= l, & \{\langle x, p \rangle\} \odot t &= \{\langle x, p \cdot t \rangle\}, \\ \{\langle x, \emptyset \rangle\} \odot t &= \{\langle x, \emptyset \rangle\}, & (l \cup l') \odot t &= (l \odot t) \cup (l' \odot t,) \end{aligned}$$

for all $l, l' \in \mathbf{Lin}$, $t \in \mathcal{T}_{Reg} \setminus \{\emptyset, \lambda\}$, $p \in \mathcal{T}_{Reg} \setminus \{\emptyset, \lambda\}$. □

Definition 2. (Partial derivatives of regular terms). Given $t \in \mathcal{T}_{Reg}$ and $x \in \mathcal{A}$, a regular term p is called a *partial derivative of t w.r.t. x* if $\langle x, p \rangle \in \mathit{lf}(t)$. The equation

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

defines a function $\partial_{\_}(\_) : \mathcal{A} \times \mathcal{T}_{Reg} \to \mathbf{SReg}$ which returns a set of (non-zero) partial derivatives of its second argument w.r.t. the first one. The following

[3] The idea behind this construction is to take into account the ACI-properties only of *some* occurrences of the operation $\_+\_$ in regular terms and in this way to reduce the use of AC-matching in the rewrite systems presented below.

equations extend this function allowing words and sets of words as the first argument and sets of regular terms as the second one:

$$\begin{array}{ll} \partial_\lambda(t) = \{t\}, & \partial_w(R) = \bigcup_{r\in R} \partial_w(r), \\ \partial_{wx}(t) = \partial_x(\partial_w(t)), & \partial_W(t) = \bigcup_{w\in W} \partial_w(t). \end{array}$$

for all $w \in \mathcal{A}^*$, $W \subseteq \mathcal{A}^*$, $R \subset \mathcal{T}_{Reg}$. An 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 (i.e., from the set $\partial_{\mathcal{A}^+}(t)$). $\square$

Proposition 3. ([1]) *For any $t \in \mathcal{T}_{Reg}$ and $w \in \mathcal{A}^*$, $\mathcal{L}(\partial_w(t)) = w^{-1}\mathcal{L}(t)$. In particular, any partial derivative $p \in \partial_w(t)$ denotes a subset of $w^{-1}\mathcal{L}(t)$.* $\square$

Theorem 4. ([1]) *For any $t \in \mathcal{T}_{Reg}$, the number of all proper partial derivatives of t is less or equal to the alphabetic width of t, i.e. $|\partial_{\mathcal{A}^+}(t)| \leq ||t||$.* $\square$

The following is an alternative to the classical definition of *word derivatives* from [3] (cf. also [4, 11]).

Definition 5. (Derivatives) Given a term $t \in \mathcal{T}_{Reg}$ and a word $w \in \mathcal{A}^*$, the regular term $\Sigma\partial_w(t)$ (up to permutation of its summands) is called a *(word) derivative of t w.r.t. w*. Given a set of words $W \subseteq \mathcal{A}^*$, let $\mathcal{D}_W(t)$ be the set $\{ \Sigma\partial_w(t) \mid w \in W \}$ of all the derivatives of t w.r.t. all the words in W. $\square$

It follows from Prop. 3 that $\mathcal{L}(\Sigma\partial_w(t)) = w^{-1}\mathcal{L}(t)$ and from Theorem 4 that the set $\mathcal{D}_{\mathcal{A}^+}(t)$ has at most $2^{||t||}$ elements.

3 Containment Calculus for Regular Algebra

As the first step towards our t.r.s, we develop a simple calculus which provides a non-deterministic decision procedure for the containment problem in **Reg**[$\mathcal{A}$]. To formulate the calculus, we first introduce a new notion which will play a central rôle in all our constructions.

Definition 6. (Partial derivatives of regular inequalities) Given a word $w \in \mathcal{A}^*$ and two regular terms a, $b \in \mathcal{T}_{Reg}$, a regular inequality $p \leq q$ is called a *partial derivative of $a \leq b$ w.r.t. w* if $p \in \partial_w(a)$ and $q = \Sigma\partial_w(b)$. Let

$$\partial_w(a \leq b) = \{ p \leq \Sigma\partial_w(b) \mid p \in \partial_w(a) \}$$

be the set of all partial derivatives of $a \leq b$ w.r.t. w. Given a set of inequalities E and a set of words $W \subseteq \mathcal{A}^*$, let $\partial_W(E)$ denote the set $\bigcup_{a\leq b\in E} \bigcup_{w\in W} \partial_w(a \leq b)$ of all the partial derivatives of inequalities in E w.r.t. all the words in W. $\square$

Some basic properties of the partial derivatives are as follows.

Proposition 7. *Given a regular inequality $a \leq b$, the following holds:*
1) $\partial_{wx}(a \leq b) = \partial_x(\partial_w(a \leq b))$ for any $x \in \mathcal{A}$, $w \in \mathcal{A}^*$.
2) If **Reg**[$\mathcal{A}$] $\models a \leq b$, *then* **Reg**[$\mathcal{A}$] $\models \partial_{\mathcal{A}^*}(a \leq b)$. $\square$

Let an *atom* be either a regular inequality, or a boolean constant *true* or *false*. Now we are in a position to formulate the Containment Calculus CC which works on sets of atoms. It consists of the rule (*Disprove*), which infers *false* from a trivially inconsistent inequality, and two rules (*Unfold*) to infer a set of partial derivatives (w.r.t. all alphabet letters) from the other inequalities.

Containment Calculus CC		
(*Disprove*) :	$a_1 \leq b_0 \vdash_{CC}$ *false*	**for** $a_1 : Regl,\ b_0 : Reg0$
(*Unfold*) :	$a_0 \leq b \vdash_{CC} \partial_{\mathcal{A}}(a_0 \leq b)$	**for** $a_0 : Reg0,\ b : Reg$
(*Unfold*) :	$a_1 \leq b_1 \vdash_{CC} \partial_{\mathcal{A}}(a_1 \leq b_1)$	**for** $a_1 : Regl,\ b_1 : Regl$

Given an initial inequality $a \leq b$, an inference in the calculus is a sequence

$$S_0 \vdash_{CC} S_1 \vdash_{CC} \ldots, \qquad (1)$$

of sets of atoms, starting with $S_0 = \{a \leq b\}$, such that each set S_{i+1} is an extention of the previous one S_i by consequences produced by one of the inference rules applied to an inequality in S_i. The calculus is sound and complete in the following sense.

Theorem 8. *Given a regular inequality $a \leq b$, the following holds:*

1) $a \leq b$ is not valid in $\mathrm{Reg}[\mathcal{A}]$ *if and only if a set of atoms containing false is derivable in CC from $a \leq b$.*

2) There are at most $\|a\| \cdot 2^{\|b\|}$ different inequalities derivable in CC from $a \leq b$. □

It follows that after a finite number of steps, the sequence (1) ends up with a set of atoms S_i which is either inconsistent (contains *false*), or saturated (i.e., $\partial_{\mathcal{A}}(S_i) = S_i$). This implies a non-deterministic decision procedure for (dis)proving regular inequalities. Note that it may take up to $O(|\partial_{\mathcal{A}^+}(a \leq b)|)$ inference steps to (dis)prove an inequality $a \leq b$. Our next goal is to implement such a procedure as a t.r.s.

4 Proving Regular Inequalities by Rewriting

First we describe an order-sorted t.r.s. CC_{rew} which provides a unique normal form for any regular inequality and in this way makes the procedure presented by CC "more deterministic".

The signature of CC_{rew} is an extension of *REG* by several new sorts and operations. The sorts with their basic constructors and corresponding axioms and rewrite rules are presented in Table 1. The remaining rewrite rules are placed in Table 2. From here on we use variables declared as follows:

variables $x,\ y : \mathcal{A};\ a,\ b,\ c : Reg;\ a_0,\ b_0 : Reg0;\ a_1,\ b_1 : Regl$
$s,\ s',\ s'' : SReg;\ s_0 : SReg0;\ s_1 : SRegl;\ C,\ C',\ C'' : Conj$

Table 1. Sorts and basic constructors of CC_{rew}

Module *SREG* over *REG*	
sorts	$SReg,\ SReg0,\ SRegl.$
subsorts	$Reg0 < SReg0 < SReg,\quad Regl < SRegl < SReg,\quad Reg < SReg.$
operations	$\_\sqcup\_ : SReg\ SReg \to SReg \quad \_\sqcup\_ : SReg\ SRegl \to SRegl$
	$\_\sqcup\_ : SRegl\ SReg \to SRegl \quad \_\sqcup\_ : SReg0\ SReg0 \to SReg0$
axioms	$s \sqcup s' = s' \sqcup s, \quad (s \sqcup s') \sqcup s'' = s \sqcup (s' \sqcup s'')$
rules	$\emptyset \sqcup s \to s, \quad s \sqcup s \to s$

Module *CONJ* over *SREG*	
sorts	$Bool,\ Ineq,\ Conj.$
subsorts	$Bool < Ineq < Conj.$
const	$true,\ false : Bool$
operations	$\_\leq\_ : Reg\ SReg \to Ineq \quad \_\leq_o\_ : Reg\ SReg \to Ineq$
	$\_\wedge\_ : Conj\ Conj \to Conj$
axioms	$C \wedge C' = C' \wedge C, \quad (C \wedge C') \wedge C'' = C \wedge (C' \wedge C'')$
rules	$true \wedge C \to C,\ false \wedge C \to false,\ C \wedge C \to C$

The module *SREG* implements the upper semilattice **SReg** needed to represent right-hand sides of regular inequalities. The join $\_\sqcup\_$ is an "ACI-synonym" of $\_+\_$ (note that we keep treating $\_+\_$ as a free constructor).

The module *CONJ* defines sorts for regular inequalities and conjunctions of those. The constructor $\_\leq_o\_$ is a synonym of the constructor $\_\leq\_$; it will be used to mark some inequalities as "old" as explained below.

The module *DERIV* implements the function $der : \mathcal{A}, SReg \to SReg$ to compute derivatives of regular terms w.r.t. alphabet letters. It involves an auxiliary operation[4] $der_2 : \mathcal{A}, Reg, Reg \to SReg$ which satisfies the following invariant: $der_2(x,\ a,\ c) = \Sigma\{\ p{\cdot}c \mid p \in \partial_x(a)\ \}$.

The module *UNFOLD* implements the function $unf : Reg, SReg \to Conj$ to compute the result of unfolding a regular inequality into a conjunction of its partial derivatives. Again, we have to involve an auxiliary operation $unf_2()$.

The upper level of the inference process is implemented in the module *MAIN* through several rewrite rules working on inequalities and conjunctions. The trick here is that the rules (*Unfold*) keep the just unfolded inequality $a \leq s$ in the resulting conjunction, but make it "old". Then, if $a \leq s$ reappears as a result of unfolding of some other inequality, the rules (*Delete*) are used to remove it from a current conjuction. To make the trick work, we have to require that the rules (*Delete*) have a higher priority against the rules (*Unfold*) (i.e., the latter can be applied only when the former can not). This prevents from unfolding the same inequality several times and provides a normalising strategy for CC_{rew}.

[4] We also use conditionals and the equality on $\mathcal{A}$ in two rules, but since the alphabet is finite, the rules can easily be replaced by a finite number of another rewrite rules without these extra operations.

Theorem 9. *CC_{rew} is normalising on ground terms by any rewrite strategy satisfying the priority condition formulated above. The normal form (unique modulo AC-axioms for $\_\sqcup\_$ and $\_\wedge\_$) of a regular inequality $r \leq t$ is either false, if $r \leq t$ is not valid in* **Reg**$[\mathcal{A}]$*, or a conjunction of all partial derivatives of $r \leq t$ otherwise.* □

At this point, we have obtained a term-rewriting solution to the containment problem in **Reg**$[\mathcal{A}]$. It turns out, however, that its efficiency can be substantially improved. Note that the complexity of derivations in CC_{rew} is determined mainly by the number of unfolding steps.[5] Now the idea is to add two group of new rewrite rules aimed at avoiding some redundant unfolding. The first group consists of rules (called *TAU*-rules) eliminating "trivial tautologies", e.g.

$$\emptyset \leq s \rightarrow true, \quad \lambda \leq s_1 \rightarrow true, \quad a \leq a \rightarrow true, \quad a \leq b_1 \cdot a \rightarrow true,$$

and so on. The second (actually, most important) group includes the following ***subsumption*** rules (*SUB*-rules, for short) eliminating some "trivial logical consequences" of already derived inequalities.

Subsumption rules	
$a \leq s \sqcup s' \wedge a \leq_o s \rightarrow a \leq_o s,$	$a \leq s \sqcup s' \wedge a \leq_o s \wedge C \rightarrow a \leq_o s \wedge C,$
$a \leq s \sqcup s' \wedge a \leq s \rightarrow a \leq s,$	$a \leq s \sqcup s' \wedge a \leq s \wedge C \rightarrow a \leq s \wedge C$

Note that the rules (*Delete*) in CC_{rew} can also be considered as the simplest kind of subsumption. Let us call the extended t.r.s. CC_{opt}.

Despite some of the new rules involve AC-operations in their left-hand sides, one-step application of the rules to a given conjunction can be computed in polynomial time (in the size of this conjunction). The more surprising is the fact that CC_{opt} can in some cases *exponentially* shorten the length of derivations comparing to CC_{rew}. We demonstrate this on the following examples. (We consider regular terms on the alphabet $\mathcal{A} = \{a, b\}$, omit concatenation symbol, and use r^k as a shorthand for k-times concatenation of r, regarding $r^{\emptyset}$ as λ.)

Example 1. Given a natural constant $n \geq 1$, consider a (valid) inequality $X \leq Y$ where $X = (a^*b)^*a^na^*$ and $Y = (a+b)^*a(a+b)^{n-1}$. It is known that the minimal DFA for Y has 2^n states (see [11, page 30])[6] and one can check that $|\partial_{\mathcal{A}^*}(X \leq Y)|$ is an exponent in n.

However, CC_{opt} rewrites $X \leq Y$ into the following normal form which contains only $n+2$ inequalities and this takes $O(n)$ unfolding and subsumption steps:

$$X \leq_o Y \;\wedge\; (a^*bX \leq_o Y \sqcup Y_1) \;\wedge$$
$$(a^{n-1}a^* \leq_o Y \sqcup Y_1) \;\wedge\; \ldots \;\wedge\; (a^* \leq_o Y \sqcup Y_1 \sqcup \ldots \sqcup Y_n).$$

where Y_k stands for $(a+b)^{n-k}$. □

[5] One unfolding step can be implemented very efficiently using the technique from [1].

[6] Thus, any decision procedure relying upon DFA for Y has to spend an exponential time for constructing it.

Example 2. Given a natural constant $n \geq 2$, consider a (valid) inequality $X \leq Y$ where $X = (b^*a)^*(ba)^{2n-3}b^*$ and $Y = (a+b)^*b(ab^*)^{n-2}((ab^*)^{n-1})^*$.

One can check that Y denotes a language from [13, Theorem 2.1] which is recognised by the minimal DFA with 2^{n-1} states and that $X \leq Y$ has an exponential number of partial derivatives. Nevertheless, $\mathcal{CC}_{opt}$ rewrites $X \leq Y$ by $O(n)$ unfolding and subsumption steps into a normal form containing only $4n-4$ inequalities. □

To provide a more general comparison of the decision procedure represented by our rewrite systems with the procedures based on finite automata, we present the following theorem.

Theorem 10. *Given a regular inequality $a \leq b$, let* **M** *be an NFA with the input alphabet $\mathcal{A}$, the set of states $M = \partial_{\mathcal{A}^*}(a \leq b)$, the initial state $\mu_0 = a \leq b$, the transition function $\tau : M \times \mathcal{A} \to$ Set$[M]$ defined for all $(p \leq q) \in M$, $x \in \mathcal{A}$ by $\tau(p \leq q,\, x) = \partial_x(p \leq q)$, and the set of final states $F = \{\ (p \leq q) \in M \mid p \in \mathcal{T}_{Reg1} \ \wedge\ q \in \mathcal{T}_{Reg0}\ \}$. Then this automaton recognises the language $\mathcal{L}(a) \setminus \mathcal{L}(b)$.*

Now it should be clear that, given an initial inequality $a \leq b$, the rewriting process in $\mathcal{CC}_{rew}$ (and so the inference process of $\mathcal{CC}$) can be interpreted as a gradual "top-down" construction of the non-deterministic automaton **M**: the rules (*Unfold*) generate new states reachable form a given one and the rule (*Disprove*) checks whether one of the obtained states is final. It can be shown that the automaton **M** represents a reachable part of a direct product $M_a \times \neg M_b$ of an NFA M_a for a and a complement of a DFA M_b for b (constructions of M_a and M_b can be found in [1]). Therefore, on the class of valid initial inequalities the complexity of the decision procedure represented by $\mathcal{CC}_{rew}$ is of the same order as the complexity of the algorithm checking emptiness of $M_a \cap \neg M_b$. But the "lazy" nature of the inference procedure gives an advantage on the class of non-valid inequalities: such an inequality can be refuted as soon as we have inferred a "final state" – a partial derivative $p \leq q$ with $p \in \mathcal{T}_{Reg1}$ and $q \in \mathcal{T}_{Reg0}$, and this may happen before we have completed constructing the whole sets of states for M_a and M_b. Thus, on this class of inputs our procedure is certainly better[7] than one that first constructs M_a and $\neg M_b$ and only then checks emptiness of their intersection.

The optimised rewrite system $\mathcal{CC}_{opt}$ provides a decision procedure which has even more advantages: now some valid inequalities can also be proved without constructing the whole sets M_a, M_b. The examples presented above demonstrates that this may be crucial when M_b has a relatively big number of states. But, of course, the subsumption test takes an extra time and may happen to be useless on some inputs. It seems that only experiments with a practical programming implementation of $\mathcal{CC}_{opt}$ may show whether this procedure can compete with the standard ones in *most* cases.

[7] Of course, in the *worst* case one has to generate all partial derivatives in order to find a final state in $M_a \cap \neg M_b$

To conclude, we would like to repeat the point made already in the introduction: the main contribution of the present paper is a new *purely symbolic* method for checking inequalities in $Reg[\mathcal{A}]$ which also provides the concept of *algebraic normal forms* for regular inequalities. The theoretical results and ideas of the present paper may also lead to new practical applications, but this is a topic for another research.

References

1. V. M. Antimirov. Partial derivatives of regular expressions and finite automata constructions. In E. W. Mayr and C. Puech, editors, *12th Annual Symposium on Theoretical Aspects of Computer Science. Proceedings.*, volume 900 of *Lecture Notes in Computer Science*, pages 455–466. Springer-Verlag, 1995.
2. V. M. Antimirov and P. D. Mosses. Rewriting extended regular expressions. *Theoretical Comput. Sci.*, 143:51–72, 1995.
3. J. A. Brzozowski. Derivatives of regular expressions. *J. ACM*, 11:481–494, 1964.
4. J. H. Conway. *Regular Algebra and Finite Machines.* Chapman and Hall, 1971.
5. N. Dershowitz and J.-P. Jouannaud. Rewrite systems. In J. van Leeuwen, A. Meyer, M. Nivat, M. Paterson, and D. Perrin, editors, *Handbook of Theoretical Computer Science*, volume B, chapter 6. Elsevier Science Publishers, Amsterdam; and MIT Press, 1990.
6. Z. Ésik and L. Bernátski. Equational properties of Kleene algebras of relations with conversion. *Theoretical Comput. Sci.*, 137:237–251, 1995.
7. 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.
8. H. B. Hunt III, D. J. Rosenkrantz, and T. G. Szymanski. On the equivalence, containment, and covering problems for the regular and context-free languages. *J. Comput. Syst. Sci.*, 12:222–268, 1976.
9. T. Jiang and B. Ravikumar. Minimal NFA problems are hard. *SIAM J. Comput.*, 22(6):1117–1141, 1993.
10. A. R. Meyer and L. J. Stockmeyer. The equivalence problem for regular expressions with squaring requires exponential space. In *Proceedings of the 13th Ann. IEEE Symp. on Switching and Automata Theory*, pages 125–179. IEEE, 1972.
11. 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.
12. V. N. Redko. On defining relations for the algebra of regular events. *Ukrainian Mat. Z.*, 16:120–126, 1964.
13. S. Yu, Q. Zhuang, and K. Salomaa. The state complexity of some basic operations on regular languages. *Theoretical Comput. Sci.*, 125:315–328, 1994.

Table 2. Further rewrite rules of CC_{rew}

Module *DERIV* over *SREG*	
$der(x,\ \emptyset)$	$\to \emptyset$
$der(x,\ \lambda)$	$\to \emptyset$
$der(x,\ y)$	$\to$ **if** $x = y$ **then** λ **else** $\emptyset$ **fi**
$der(x,\ a + b)$	$\to der(x,\ a) \sqcup der(x,\ b)$
$der(x,\ a \sqcup s)$	$\to der(x,\ a) \sqcup der(x,\ s)$
$der(x,\ a^*)$	$\to der_2(x,\ a,\ a^*)$
$der(x,\ a_0 \cdot b)$	$\to der_2(x,\ a_0,\ b)$
$der(x,\ a_1 \cdot b)$	$\to der_2(x,\ a_1,\ b) \sqcup der(x,\ b)$
$der_2(x,\ \emptyset,\ c)$	$\to \emptyset$
$der_2(x,\ \lambda,\ c)$	$\to \emptyset$
$der_2(x,\ y,\ c)$	$\to$ **if** $x = y$ **then** c **else** $\emptyset$ **fi**
$der_2(x,\ a + b,\ c)$	$\to der_2(x,\ a,\ c) \sqcup der_2(x,\ b,\ c)$
$der_2(x,\ a^*,\ c)$	$\to der_2(x,\ a,\ a^* \cdot c)$
$der_2(x,\ a_0 \cdot b,\ c)$	$\to der_2(x,\ a_0,\ b \cdot c)$
$der_2(x,\ a_1 \cdot b,\ c)$	$\to der_2(x,\ a_1,\ b \cdot c) \sqcup der_2(x,\ b,\ c)$

Module *UNFOLD* over *CONJ+DERIV*	
$unf(\emptyset,\ s)$	$\to true$
$unf(\lambda,\ s)$	$\to true$
$unf(x,\ s)$	$\to \lambda \leq der(x,\ s)$
$unf(a^*,\ s)$	$\to unf_2(a,\ a^*,\ s)$
$unf(a + b,\ s)$	$\to unf(a,\ s) \wedge unf(b,\ s)$
$unf(a_0 \cdot b,\ s)$	$\to unf_2(a_0,\ b,\ s)$
$unf(a_1 \cdot b,\ s)$	$\to unf_2(a_1,\ b,\ s) \wedge unf(b,\ s)$
$unf_2(\emptyset,\ c,\ s)$	$\to true$
$unf_2(\lambda,\ c,\ s)$	$\to true$
$unf_2(x,\ c,\ s)$	$\to c \leq der(x,\ s)$
$unf_2(a^*,\ c,\ s)$	$\to unf_2(a,\ a^* \cdot c,\ s)$
$unf_2(a + b,\ c,\ s)$	$\to unf_2(a,\ c,\ s) \wedge unf_2(b,\ c,\ s)$
$unf_2(a_0 \cdot b,\ c,\ s)$	$\to unf_2(a_0,\ b \cdot c,\ s)$
$unf_2(a_1 \cdot b,\ c,\ s)$	$\to unf_2(a_1,\ b \cdot c,\ s) \wedge unf_2(b,\ c,\ s)$

Module *MAIN* over *UNFOLD*	
(*Delete*) :	$a \leq s \wedge a \leq_o s \to a \leq_o s$
(*Delete*) :	$a \leq s \wedge a \leq_o s \wedge C \to a \leq_o s \wedge C$
(*Disprove*) :	$a_1 \leq s_0 \to false$
(*Unfold*) :	$a_0 \leq s \to unf(a_0,\ s) \wedge a_0 \leq_o s$
(*Unfold*) :	$a_1 \leq s_1 \to unf(a_1,\ s_1) \wedge a_1 \leq_o s_1$