ELSEVIER

Mathematics and Computers in Simulation 45 (1998) 59–71

MATHEMATICS AND COMPUTERS IN SIMULATION

Regular expression simplification

Alejandro A.R. Trejo Ortiz[a,*], Guillermo Fernández Anaya[b]

[a]*Dept. de Computación, Facultad de Ingeniería UNAM, México, D.F.*
[b]*Dept. de Matemáticas, Universidad Iberoamericana, México, D.F.*

Abstract

This paper presents a technique to simplify regular expressions. Several ideas from Computer Algebra and symbolic computing are applied to the regular expression symbolic domain. An implementation of the described technique has been developed. © 1998 IMACS/Elsevier Science B.V.

1. Introduction

Although Kleene's regular expressions were introduced in the middle 50's and they play a central role in many areas of computer science, such as in the design of sequential circuits, operating systems, www searches, text markup languages (SGML) and, very specially, in the theory of compiling, at the present time an automatic method for their simplification is not available. Such simplifications become desirable whenever you face involved language descriptions which require long and tedious sequences of manipulations. In fact, there are at least three important reasons for keeping regular expressions in a simplified form. First of all, simplified expressions usually require less memory. Second, the processing of simplified expressions is faster and simpler, and finally, equivalent expressions are easier to identify when they are in simplified form.

In this paper, we propose a first step to the solution of the regular expression simplification problem. Our system simplifies usual regular expressions, i.e., expressions which have only the $+$, $\cdot$ and $*$ operators. Simplify means, as far as we are concerned, that the length of the output language description will be less than (or, in the worst case, equal to) the length of the input language description.

In Section 1, the problem is stated. Some basic definitions are shown, and the structural similarities we have found among other symbolic domains and regular expressions are presented. In Section 2,

* Corresponding author. e-mail: guillermo.fernandez@uia.mx

PII S0378-4754(97)00086-4

some results and the technique for simplification are outlined. Here, the lack of canonical forms for regular expressions is the major feature of our problem (Hunt [8]). So we have to state a normal form for regular expressions. In Section 3, the method of representation and manipulation of expressions is explained. In Section 4, we present a sample from the outcome of our system. Finally, in Section 5, we expose our conclusions and future plans.

2. Definitions and preliminaries

Let Σ be a finite nonempty set designated as the input alphabet (or vocabulary). The elements of Σ are denoted as letters or symbols. Let Σ^* be the free semigroup with identity generated by Σ, where one may write the operation multiplicatively and denoted by juxtaposition (Kuich [9], Salomaa [14]). The elements of Σ^* are called words or sentences over the alphabet Σ. The symbol $\emptyset$ denotes the empty set. The identity of Σ^*, called the empty string or word, i.e. the word that does not contain any element, is denoted by λ (Salomaa [14]). By a language L in the alphabet Σ (or simply a language), we understand any subset of Σ^* (Mikolajczak [12]), namely,

$$L \subseteq \Sigma^*$$

Let us consider finite sentences consisting of the elements of Σ, the symbols $\emptyset$ and λ, the so-called regular operators, to wit, sum $(+)$, concatenation or product $(\cdot)$ and closure $(^*)$, and parentheses. The notion of a regular expression over the alphabet Σ and the sets that they denote are defined recursively as follows (Hopcroft [7], Mikolajczak [12]):

1. $\emptyset$ is a regular expression and denotes the empty set.
2. λ is a regular expression and denotes the set $\{\lambda\}$.
3. For each a in Σ, a is a regular expression and denotes the set $\{a\}$.
4. If r and s are regular expressions denoting the languages R and S, respectively, then $(r+s)$, $(r\cdot s)$ y (r^*) are regular expressions that denote the sets $R \cup S$, $R\cdot S$, y R^* , respectively.

We use the term simple symbol to refer to λ, $\emptyset$ or a symbol in Σ appearing in a regular expression. We may state that:

The *length of a regular expression* α, denoted $|\alpha|$, is the number of simple symbols composing the regular expression. For example, the regular expression $a\cdot b+\lambda$ has length 3.

Unnecessary parentheses can be avoided in regular expressions if we adopt the conventions that (Aho [1]):

1. The unary operator * has the highest precedence and is left associative.
2. The binary operator $\cdot$ has the second highest precedence and is left associative.
3. The binary operator $+$ has the lowest precedence and is left associative.

Under these conventions, $(a)+((b)^*\cdot (c))$ is equivalent to $a+b^*\cdot c$.

The notation $\alpha\equiv\beta$ used hereafter signifies that regular expression α and β are identical, in other words, contain the same letters in the same order. Besides, $\alpha=\beta$, where α and β denote the sets A and B, respectively, is a valid equation provided that the sets A and B are identical.

A regular expression r possesses the empty word property (e.w.p.), iff one the following conditions is fulfilled (Salomaa Sal [14]):

1. $r \equiv s^*$, for some regular expression s.
2. r is a sum of regular expressions, one of which possesses e.w.p.
3. r is a concatenation of regular expressions, each of which posseses e.w.p.

2.1. Normal functions

Let S be a set of expressions and let $\sim$ an equivalence relation on S. A normal function for $[S;\sim]$ is a computable function $f{:}S \rightarrow S$ which satisfies the following properties (Geddes [4]):

- $f(a) \sim a \forall a \in S$;
- $a \sim 0 \Rightarrow f(a) \equiv f(0)\ \forall\ a \in S$.

2.2. Normal forms

If f is a normal function for $[S;\sim]$ then an expression $\tilde{a} \in S$ is said to be a normal form if $f(\tilde{a}) \equiv \tilde{a}$ (Geddes [4]).

2.3. Algebra of regular expressions

It may be stated an Ω -algebra of regular expressions as follows:

$[E;+,\cdot,^*]$ the set of regular expressions over an alphabet Σ under the binary operations of union and concatenation and the unary operation of Kleene's closure. E is closed under $+,\cdot,^*$ (Hopcroft [7]).

Moreover, following is a set of eleven axioms and two rules that provide a formalization of the equivalence of regular expressions, and more generally describe what can be done with regular expressions (Salomaa [14]):

$$A_1\ (a+b)+c = a+(b+c)$$

$$A_2\ (a \cdot b) \cdot c = a \cdot (b \cdot c)$$

$$A_3\ a+b = b+a$$

$$A_4\ a \cdot (b+c) = a \cdot b + a \cdot c$$

$$A_5\ (a+b) \cdot c = a \cdot c + b \cdot c$$

$$A_6\ a+a = a$$

$A_7\ a \cdot \lambda = a$

$A_8\ a \cdot \emptyset = \emptyset$

$A_9\ a + \emptyset = a$

$A_{10}\ a^* = a \cdot a^* + \lambda$

$A_{11}\ a^* = (a + \lambda)^*$

There are two rules of inference:

R1 (**Substitution.**) Assume that γ' is the result of replacing an occurrence of α by β in γ. Then from the equations $\alpha=\beta$ and $\gamma=\delta$, one may infer the equation $\gamma'=\delta$ and the equation $\gamma'=\gamma$.

R2 (**Solution of equations.**) Assume that β does not posses e.w.p. Then from the equation $\alpha=\alpha\cdot\beta+\gamma$ one may infer the equation $\alpha=\gamma\cdot\beta^*$.

2.4. Algebraic structure of regular expressions

It may be proved that

$[E; +, \emptyset]$ is a commutative monoid
$[E; \cdot, \lambda]$ is a non-commutative monoid
$[E;+,\emptyset,\cdot,\lambda\ ]$ is a non-commutative semiring with identity because of
i) $P \cdot (Q + R) = P \cdot Q + P \cdot R$
ii) $(P + Q) \cdot R = P \cdot R + Q \cdot R$

2.5. Domain properties of regular expressions

It is obvious that:

$a+b = \emptyset \Leftrightarrow a=b = \emptyset$, and
$a\cdot b=\emptyset \Leftrightarrow a=\emptyset$ or $b=\emptyset$

2.6. Structural similarities among other symbolic domains and regular expressions

In order to develop our regular expression simplifier, we took several ideas from the simplification of other mathematical objects. In fact, our approach arose from the following observations:

- Symbolic data objects are recursives in nature (Cameron [3]).
- Because of [Z};+,0,·,1] is a commutative ring, many of the axioms A_1–A_{11} are familiar from arithmetic, if union and addition are thought of as analogous, and concatenation and multiplication

are likewise, although the analogy cannot be taken too far. Note that $\emptyset$ functions as the identity for union (analogous to zero in arithmetic) and that λ functions as the identity for concatenation (analogous to one).
- The basis for expression simplification is the existence of various simplification rules for each type of mathematical expression (Korsvold [11], Sammet [13], Knuth [10]).
- As in algebraic expression simplification, one may define identity removal when handling terms and factors involving zero and one elements (Sammet [13]). On the other hand, it has found that the problem of determining canonical forms for regular expressions is still not solved and appears to be rather difficult (Hunt [8]). We suspect that it is at least of NP-hard complexity.

3. Results

In this section, we present some ideas that we have found of importance in our way toward the solution of the problem which stirs our interest. Firstly, we review some of the existing material about substitutions and homomorphisms of regular languages, then we state a proposition regarding such homomorphisms. We conclude this section by showing our simplification scheme.

3.1. Substitutions and homomorphisms

A substitution f is a mapping f: $\Sigma \rightarrow \Delta^*$, for some alphabet Δ, with the following properties:

1. $f(\lambda) = \lambda$
2. $f(x \cdot a) = f(x) \cdot f(a)$
3. $f(a + b) = f(a) + f(b)$
4. $f(a^*) = f(a)^*$
5. $f(\emptyset) = \emptyset$

Note that $^* : \Sigma \rightarrow \Sigma^*$ is similar to a substitution even though it is not properly a substitution. In effect, we have:

1. $\lambda^* = \lambda$
2. $(a^* \cdot b^*)^* = (a + b)^*, \textit{however},$
3. $(a + b)^* \neq a^* + b^*$
4. $(a \cdot b)^* \neq a^* \cdot b^*$
5. $\emptyset^* = \lambda \neq \emptyset$

Substitutions do not preserve simpler regular expressions, v. gr.

$$f(0) = a,\ f(1) = b^*$$
$$f(0^* \cdot (0 + 1) \cdot 1^*) = a \cdot (a + b^*) \cdot (b^*)^*$$

which is equivalent to the simpler regular expression $a^* \cdot b^*$.

3.2. Homomorphisms

A type of substitution that is of special interest is the homomorphism. A homomorphism f is a substitution such that $f(a)$ contains a single string for each a (Hopcroft [7]). Usually, homomorphisms do not preserve simpler regular expressions, v. gr.

$$f(a) = \bar{a} \cdot c$$
$$f(b) = \bar{a} \cdot d$$
$$f(a + b) = \bar{a} \cdot c + \bar{a} \cdot d$$

but $\bar{a} \cdot (c + d)$ is equivalent and simpler than $\bar{a} \cdot c + \bar{a} \cdot d$, and recall that we started from $a+b$ as simple.

However, let w_1 and $w_2 \in \Sigma^*$ be as follows:

$$w_1 = x_1 \ldots x_n, \; w_2 = y_1 \ldots y_m$$

and $x_1 \ldots x_k = y_1 \ldots y_k$ nor $x_{n-k} \ldots x_n = y_{m-l} \ldots y_m$, in which $n - k = m - l$; in other words, strings that do not have either prefix or suffix taken in pairs, then the following statement is true.

3.3. Proposition

If E is a Simpler regular expression (Sre) and w_1, w_2 are specified as above, then there exists a homomorphism $f : \Sigma \to \Delta^*$ that preserves (Sre)'s, f given by $f(w_1) = \tilde{w}_1$ and $f(w_2) = \tilde{w}_2$, $\quad w_1, w_2 \in \Sigma^*$ and $\tilde{w}_1, \tilde{w}_2 \in \Delta^*$, that is to say,

$$f(E(w_1, w_2)) = \tilde{E}(\tilde{w}_1, \tilde{w}_2)$$

In other words, if E is (Sre) also $\tilde{E}$ is (Sre).

3.4. Proof

Let $f : \Sigma \mapsto \Delta^*$ be specified as $f(w_1) = \tilde{w}_1 = \bar{x}_1 \ldots \bar{x}_n, f(w_2) = \tilde{w}_2 = \bar{y}_1 \ldots \bar{y}_m$. Then, it is obvious that, for axiom A_1:

$$\begin{aligned} f(w_1 + (w_2 + w_3)) &= f(w_1) + f(w_2 + w_3) = f(w_1) + (f(w_2) + f(w_3)) = \tilde{w}_1 + (\tilde{w}_2 + \tilde{w}_3) \\ &= (\tilde{w}_1 + \tilde{w}_2) + \tilde{w}_3 \end{aligned}$$

We can prove the validity of our proposition for axioms A_2–A_{11} in a similar way. For example, for axiom A_{10} we have:

$$f(w_1^*) = f(w_1)^* = \tilde{w}_1^*$$

and also,

$$f(w_1 \cdot w_1^* + \lambda) = f(w_1) \cdot f(w_1^*) + f(\lambda) = \tilde{w}_1 \cdot \tilde{w}_1^* + \lambda = \tilde{w}_1^*$$

For axiom A_{11}:

$$f(w_1 + \lambda)^* = (f(w_1) + f(\lambda))^* = (\tilde{w}_1^* + \lambda)^*$$

For rule R_1:

$$\gamma = \alpha_1 \ldots \alpha_i \alpha \alpha_{i+1} \ldots \alpha_n$$
$$\gamma' = \alpha_1 \ldots \alpha_i \beta \alpha_{i+1} \ldots \alpha_n$$

Applying f we get:

$$\bar{\gamma} = \bar{\alpha}_1 \ldots \bar{\alpha}_i \bar{\alpha} \bar{\alpha}_{i+1} \ldots \bar{\alpha}_n$$
$$\gamma' = \bar{\alpha}_1 \ldots \bar{\alpha}_i \bar{\beta} \bar{\alpha}_{i+1} \ldots \bar{\alpha}_n$$

Then, if $\alpha = \beta$ and $\gamma = \delta$

$$f(\alpha) = f(\beta) = \bar{\alpha} = \bar{\beta}$$
$$f(\gamma) = f(\delta) = \bar{\gamma} = \bar{\delta}$$

From these equations, we can infer that

$$f(\gamma') = f(\delta) = \bar{\gamma}' = \bar{\delta}$$
$$f(\gamma') = f(\gamma) = \bar{\gamma}' = \bar{\gamma}$$

For rule R2 it is also easy to see that:

$$f(\alpha) = \bar{\alpha}$$
$$f(\alpha \cdot \beta + \gamma) = f(\alpha \cdot \beta) + f(\gamma) = f(\alpha) \cdot f(\beta) + f(\gamma) = \bar{\alpha} \cdot \bar{\beta} + \bar{\gamma}$$

Also,

$$f(\gamma \cdot \beta^*) = f(\gamma) \cdot f(\beta^*) = f(\gamma) \cdot f(\beta)^* = \bar{\gamma} \cdot \bar{\beta}^*$$

which implies

$$\bar{\alpha} = \bar{\gamma} \cdot \bar{\beta}^* \quad \square$$

Let us notice that the proposition above is very simple but, as far as we know, it is the first example of a homomorphism which preserves simpler regular expressions. Naturally, this homomorphism is not the only one, it is just the first step toward the study of homomorphisms preserving simpler regular expressions.

3.5. Simplification technique

As was stated in the section above, the lack of canonical forms for regular expressions (Hunt [8]) is the major feature of our problem, so it is not possible to develop a canonical simplifier for these expressions. However, various notions weaker than, but approximating canonical simplification have been introduced in the literature (Buchberger [2]). Normal simplification is one of them. This kind of simplification involves dealing with normal functions.

3.6. Normal function for regular expressions

A normal form for regular expressions over an alphabet Σ can be specified by the normal function:

E	Set of regular expressions over an alphabet Σ
$\sim$	Equivalence relation 'is equal to'
$\equiv$	Equivalence relation 'is identical to'
f	Normal function for $[E; \sim]$

f: $E \mapsto E$ satisfies

1. $f(a) \sim a \ \forall \ a \in E$
2. $a \sim \emptyset \Rightarrow f(a) \equiv f(\emptyset) \ \forall \ a \in E$

f specified as follows:

1. collect like terms
2. sort terms in lexicographic order
3. apply simplifying transformations

Our scheme for solution is based on the Rochester method for simplifying expressions (Goldberg [5]), which was a pioneer work in symbolic computing. We have modified such a method in order to get better results in our particular application area.

From this basis, we use an over-all scheme that first simplifies each argument of a regular expression and then uses the special simplification functions that exist for the regular operators, namely: sum, concatenation and closure. The simplification rules for regular expressions are implemented in these special simplification functions. We put the factors of a regular expression into a normal form. Furthermore, like terms and concatenation expressions are combined, and complete identity removal is performed.

4. Representation and manipulation of expressions

From the definition of regular expressions (Hopcroft Hop [7]), we conclude that there are, basically, two types of these expressions:

1. simple regular expressions, for example, a;
2. composite expressions, constructed in terms of simple expressions, and which may be of three kinds:
   (a) sum regular expressions, for example, $S+T$;
   (b) concatenation regular expressions, for example, $S \cdot T$;
   (c) closure regular expressions, for example, R^*.

4.1. Regular expression class

We used an infix representation for regular expressions. From an object-oriented point of view, we define the class Regular-Expression, which has the following elements:

1. the regular expression type
   - simple
   - sum
   - concatenation
   - closure;
2. If it is a simple expression, the character string;
3. If it is a sum or concatenation expression, operand 1 and operand 2;
4. If it is a closure expression, operand.

4.2. Methods for the regular expression class

The fundamental methods of the Regular-Expression class are:

Variable P	Recognizes if a regular expression is a simple expression.
Sum P	Recognizes if a regular expression is a sum expression.
Concatenation P	Recognizes if a regular expression is a concatenation expression.
Closure P	Recognizes if a regular expression is a closure expression.
Void P	Recognizes if a regular expression is the expression corresponding to $\emptyset$.
Empty P	Recognizes if a regular expression is the expression corresponding to λ.
Operand	Selects the operand of a closure expression.
Operand 1	Selects the operand 1 of a sum or concatenation expression.
Operand 2	Selects the operand2 of a sum or concatenation expression.
Make Closure	Constructs a closure expression.
Make Sum	Constructs a sum expression.
Make Concatenation	Constructs a concatenation expression.
Make Variable	Constructs a simple expression.
Equal	Determines if two regular expressions are the same at the form level of abstraction, that is, identical as strings of symbols.

4.3. Grammar for regular expressions

In the development of our system, we were concerned with the design of a grammar for regular expressions.

4.4. Ambiguous grammar

In order to develop a textual notation for the domain of regular expressions, we design a grammar for regular expressions as shown:

$$R ::= R + R \| R \cdot R \| R^* \| (R) \| a \| b \| \| \ldots \| z \| \emptyset \| \lambda$$

This grammar removes the immoderate use of parentheses as follows from the definition of regular expressions. In doing so, however, the grammar has introduced ambiguities – for example, $a+b \cdot c$ might correspond to either $((a+b)\cdot c)$ or $(a+(b\cdot c))$ in a complete parenthesized notation. In fact,

Table 1
Grammar

$\langle \text{expr} \rangle ::= \langle \text{sum} \rangle \parallel \langle \text{term} \rangle$
$\langle \text{term} \rangle ::= \langle \text{concat} \rangle \parallel \langle \text{factor} \rangle$
$\langle \text{factor} \rangle ::= \langle \text{simple} \rangle \parallel \langle \text{closure} \rangle$
$\langle \text{simple} \rangle ::= \langle \text{variable} \rangle \parallel \langle \text{parenthesis} \rangle$
$\langle \text{sum} \rangle ::= \langle \text{expr} \rangle + \langle \text{term} \rangle$
$\langle \text{concat} \rangle ::= \langle \text{term} \rangle \cdot \langle \text{factor} \rangle$
$\langle \text{closure} \rangle ::= \langle \text{simple} \rangle^*$
$\langle \text{variable} \rangle ::= \langle \text{alphanum} \rangle \{ \langle \text{alphanum} \rangle \} \parallel \emptyset \parallel \lambda$
$\langle \text{parenthesis} \rangle ::= (\langle \text{expr} \rangle)$
$a \parallel b \parallel c \parallel \ldots \parallel z \parallel$
$\langle \text{alphanum} \rangle ::= A \parallel B \parallel C \parallel \ldots \parallel Z \parallel$
$0 \parallel 1 \parallel 2 \parallel \ldots \parallel 9$

ambiguity is a common problem for symbolic notation systems which use infix operators (Aho [1]). But the grammar above has lots of ambiguities, and these do not depend on which particular infix operators are involved. Thus, expressions such as $a{+}b{+}c$ and $a \cdot b{\cdot}\ c$ are also ambiguous as stated by this grammar.

4.5. Unambiguous grammar

Proper definition of a textual notation requires that all grammatical ambiguities be resolute. In the case of regular expressions, we can resolve the ambiguities using the so-called mathematical conventions for precedence and associativity (Aho [1]).

The unambiguous grammar (Table 1) use these ideas to disambiguate the former grammar. Also introduces a rule for parenthesized expressions in order to allow any type of regular expressions to be represented in our system. Our grammar design approach used marker tokens systematically to distinguish among the different ways of expanding the grammatical rules (Holub [6]). Furthermore, in designing a parser for regular expressions, we have dealt with and provided solution for left-recursive grammar rules.

5. Examples

115 regular expressions were selected from various textbooks. For 110 of them, the simplified form produced by our scheme was less in length than the input expressions. For the other 5, the size of the output language description remained the same as the input language description. Some examples are shown immediately:

$$\emptyset \cdot R + \emptyset + R \mapsto R \tag{1}$$

$$\emptyset \cdot R \cdot R + \emptyset \mapsto \emptyset \tag{2}$$

$$\emptyset \cdot R + R + R \mapsto R \tag{3}$$

$$\emptyset \cdot R + \lambda + R \cdot R^* \mapsto R^* \tag{4}$$

$$\emptyset \cdot R + P \cdot Q + P \cdot R \mapsto P \cdot (Q + R) \tag{5}$$

$$\emptyset \cdot R + P \cdot Q + R \cdot Q \mapsto (P + R) \cdot Q \tag{6}$$

$$\emptyset \cdot R + \emptyset \cdot R \mapsto \emptyset \tag{7}$$

$$\emptyset \cdot R + R \cdot \emptyset \mapsto \emptyset \tag{8}$$

$$\emptyset \cdot R + \lambda \cdot R \mapsto R \tag{9}$$

$$\emptyset \cdot R + R \cdot \lambda \mapsto R \tag{10}$$

$$\emptyset \cdot R + R^* \cdot R^* \mapsto R^* \tag{11}$$

$$\emptyset \cdot R + P^* \cdot (Q \cdot P^*)^* \mapsto (P + Q)^* \tag{12}$$

$$\emptyset \cdot R + (P^* \cdot Q)^* \cdot P^* \mapsto (P + Q)^* \tag{13}$$

$$\emptyset \cdot R + \lambda^* \mapsto \lambda \tag{14}$$

$$\emptyset \cdot R + \emptyset^* \mapsto \lambda \tag{15}$$

$$\emptyset \cdot R + (R^*)^* \mapsto R^* \tag{16}$$

$$\emptyset \cdot R + (\lambda + R)^* \mapsto R^* \tag{17}$$

$$\emptyset \cdot R + (P^* + Q^*)^* \mapsto (P + Q)^* \tag{18}$$

$$\emptyset \cdot R + (P^* \cdot Q^*)^* \mapsto (P + Q)^* \tag{19}$$

$$(\lambda + \lambda \cdot (\lambda + R \cdot R^*))^* \mapsto R^* \tag{20}$$

$$(\lambda)^* \cdot 0 + \lambda \mapsto \lambda + 0 \cdot 0 \tag{21}$$

$$\lambda + 1^* \cdot 011^* \cdot ((1^* \cdot 011)^*)^* \mapsto \lambda + 1^* \cdot 011^* \cdot (1 + 011)^* \tag{22}$$

$$\lambda + R \cdot R^* \mapsto R^* \tag{23}$$

$$0^* \cdot 1 \cdot (\lambda + (0+1)0^* \cdot 1)^* \cdot (0+1) \cdot (00)^* + 0 \cdot (00)^* \mapsto 0^* \cdot 1 \cdot ((0+1) \cdot 0^* \cdot 1)^* \cdot (0+1) + 0) \cdot 00^* \tag{24}$$

$$\emptyset + R \cdot (R + \emptyset) + R + R \cdot (\lambda + R \cdot R^*) \cdot (P \cdot Q + P \cdot R) \mapsto R + R \cdot R^* \cdot (P \cdot (Q + R)) \tag{25}$$

6. Conclusions, limitations and future plans

We have made a start in the solution of the general problem of regular expression simplification. However, the performance of our system is satisfactory for the time being.

We have modified the Rochester algorithm introducing several functions that put the expressions in a normal form in order to optimize the processing. In order to do that we have stated a normal form for regular expressions. Furthermore, we propose an unambiguous grammar for the regular expression symbolic domain. We have applied notions of computer algebra and symbolic computing to the regular expression context, like normal simplifier, identity removal, and so forth. We are convinced that it is necessary to know more about the regular expression algebra in order to obtain better results, for example, homomorphisms that preserve (Sre)'s. The lack of canonical forms for regular expressions may lead to different simplified expressions even if the expressions are equivalent. Nonetheless, this can be tempered if we use normal forms, as we do. Our future plans are, basically: Explore the relation between regular expressions and automata, specially automata and regular expression homomorphisms and substitutions, and with the usage of commutative algebra and category theory. Apply optimization techniques, i.e., minimize the number of regular operators of an expression and state a metric for regular sets which is related to the number of operators of a regular expression. Generalize our system to include extended regular expressions, i.e., those which contain cap and operators. We think that this use of fundamental concepts of Computer Algebra to regular expressions is a suitable approach and may be of use to the computer scientist engaged in compiler theory or in other areas where these expressions are used.

References

[1] A.V. Aho, R. Sethi, J.D. Ullman, Compilers: Principles, Techniques and Tools, Addisson–Wesley, Reading, Massachusetts, 1986.
[2] B. Buchberger, G.E. Collins, R. Loos (Eds.), Computer Algebra, Symbolic and Algebraic Computation, 2nd ed., Springer, Wien, Österreich, 1983.
[3] R. Cameron, A. Dixon, Symbolic Computing with LISP, Prentice Hall, Englewood Cliffs, New Jersey, 1992.
[4] K.O. Geddes, S.R. Czapor, G. Labahn, Algorithms for Computer Algebra, Kluwer Academic Publisher, Boston, Massachusetts, 1992.
[5] S.H. Goldberg, Solution of an electrical network using a digital computer, M.S. thesis, Electrical Engineering Dep't., MIT, Cambridge, Mass., 1959.

[6] A.I. Holub, Compiler Design in C, Prentice Hall, Englewood Cliffs, New Jersey, 1990.
[7] J.E. Hopcroft, J.D. Ullman, Introduction to Automata Theory, Languages and Computation, Addison–Wesley, Reading, Massachusetts, 1979.
[8] H.B. Hunt III, D.J. Rosenkrantz, T.G. Szymanski, On the equivalence, containment, and covering problems for the regular and context-free languages, JCSS 12(1976).
[9] W. Kuich, A. Salomaa, Semirings, Automata, Languages, Springer, 1986.
[10] D.E. Knuth, The Art of Computer Programming, Fundamental Algorithms, 2nd ed., vol. 1, Addison–Wesley Publishing Company, Reading, Massachusetts, 1973.
[11] K. Korsvold, An on-line program for non-numerical algebra, CACM 9 (1966).
[12] B. Mikolajczak (Ed.), Algebraic and structural automata theory, Annals of Discrete Mathematics 44, North-Holland, 1991.
[13] J. Sammet, Survey of formula manipulation, CACM 9 (1966).
[14] A. Salomaa, Two complete axiom systems for the algebra of regular vents, JACM 13 (1966).