Journal of Automata, Languages and Combinatorics 5 (2000) 3, 343–354

ON A CLASS OF REGULAR-LIKE EXPRESSIONS FOR LINEAR LANGUAGES [1]

JOSÉ M. SEMPERE
Departamento de Sistemas Informáticos y Computación
Universidad Politécnica de Valencia, E-46071 Valencia, Spain
e-mail: jsempere@dsic.upv.es

ABSTRACT

Regular expressions define regular languages, so, there exist algorithms that can solve some important problems concerning regular languages such as finite automata synthesis or analysis by using regular expressions. In this work, we propose an extension of regular expressions to characterize a larger language class, linear languages. Linear languages form a class which is properly included in the context-free language class and which also properly includes the regular language class. From the definition proposed in this paper, an algorithm which obtains linear grammars from linear expressions (and vice versa) is formulated in a way similar to the one for regular expressions. We also review some problems concerning linear grammars such as the equivalence and the structural equivalence problem.

Keywords. Formal languages, linear languages, regular expressions, representation theorems

1. Introduction

Conventionally, regular languages have been defined by finite automata, right (left) linear grammars or regular expressions as presented in any basic book on formal language theory such as [6]. Some of these concepts have been extended and/or modified in order to define larger classes of languages. Therefore, pushdown automata and context-free grammars are able to define context-free languages. New (regular-like) expressions have been proposed for different (regular and non regular) language classes, and some works have taken such direction. So, YOKOMORI [15] proposed an extension of regular expressions to define context-free languages for inductive inference purposes. HASHIGUCHI and YOO [4, 5, 16] proposed regular-like expressions to characterize bounded star degree languages. GRUSKA [3] introduced the operation of *symbol iteration* and defined the context-free class in terms of union, product and symbol iteration operations, he proposed context-free expressions by using the previous operators. YNTEMA [14] proposed the *cap* operator and introduced *cap expressions* to

[1] Full version of a submission presented at the First International Workshop on *Descriptional Complexity of Automata, Grammars and Related Structures* held in Magdeburg, Germany, July 20–23, 1999.

characterize context-free languages through cap, concatenation and union operators. More on regular-like expressions for context-free languages can be found in SALOMAA's book [13]. Here, we focuse on the structural information of linear grammars in order to propose regular-like expressions to characterize them.

The definition of a descriptive formal language such as formal expressions, opens up the possibility of proposing effective algorithms in order to obtain grammars from formal expressions and vice versa. Furthermore, this definition can help to easily study some problems related to formal grammars such as descriptional complexity and reversal complexity, in the linear case.

This work is structured as follows. First, some basic concepts concerning regular expressions and some transformations related to obtaining finite automata are presented. Then linear grammars are defined and we propose an extension of regular expressions to define linear languages. From the definition of linear expressions, we propose some algorithms which obtain linear grammars from expressions and linear expressions from grammars. We relate this work to other problems such as the equivalence and the structural equivalence for linear grammars. Finally, we present the conclusions of this work and some guidelines for future works.

2. Some Basic Concepts About Regular Languages

In this section, we provide some basic concepts about the definition of regular languages and we present several transformations on these. The definitions have mainly been obtained from classical works on the formal language theory presented in [6, 13]. The concepts that we provide are basically focused on the relationship between regular grammars (finite automata) and regular expressions.

Definition 1 *Let Σ be an alphabet without the parenthesis symbols. A regular expression over Σ is defined in an inductive way as follows:*

1. *$\emptyset$ and λ are regular expressions,*
2. *for all $a \in \Sigma$, a is a regular expression,*
3. *if r is a regular then so is (r),*
4. *let r and s be regular expressions, then $r+s$, rs and r^* are regular expressions*

The only regular expressions are those defined according to these rules.

Any regular expression r denotes a language $L(r)$ which is defined as follows

1. $L(\emptyset)$ is the empty language,
2. $L(\lambda) = \{\lambda\}$,
3. $\forall a \in \Sigma \;\; L(a) = \{a\}$,
4. $L((r)) = L(r)$,
5. $L(r+s) = L(r) \cup L(s)$,
6. $L(rs) = L(r)L(s)$,
7. $L(r^*) = (L(r))^*$.

It has been formally proved that any regular language is defined by a regular expression and vice versa. Specifically, the *synthesis problem* is defined as the problem of finding a regular grammar (finite automata) which is equivalent to a given regular expression, while the *analysis problem* is defined as the opposite one, that is, finding a regular expression that denotes the language defined by a regular grammar (finite automata).

Different solutions have been proposed to solve the synthesis problem, such as the solutions proposed in [10] and [1] and, more recently, the proposal by HROMKOVIČ et al. [7]. The method proposed here to solve the synthesis problem for linear grammars is highly related to the method of derivatives of regular expressions proposed by BROZOZOWSKI [1]. Therefore, we are going to provide a basic definition which is related to this method.

Definition 2 *Let L be a language defined over the alphabet Σ and x be a string over the same alphabet. The right quotient of L with respect to x is denoted by $x^{-1}L$ and is defined to be the set $\{u \in \Sigma^* : xu \in L\}$. This set is known as the derivative of L with respect to x and can also be denoted by $der_x(L)$.*

It has been shown by NERODE's Theorem that, given a regular expression over the alphabet Σ, the different derivatives of the regular expression with respect to every string form a finite set. In the same way, from the set of different derivatives obtained from the regular expression, a deterministic finite automata (DFA) can be constructed which is equivalent to the regular expression as shown in [1]. The method for obtaining a DFA from a regular expression is known as the *derivative method*.

The analysis problem has received different solutions as well. Some of them can be found in [2]. Specifically, there exists a method which is based on systems of linear equations where the coefficients and variables of the system are denoted by regular expressions. Thus, the resolution of the system obtained from a regular grammar (finite automata) gives the desired regular expression as shown in the same book [2]. This method can be adapted in much the same way as the derivative method in order to work with the extension over regular expressions.

3. Extensions of Regular Expressions: Linear Expressions

In this section, we propose an extension of regular expressions in order to define the languages generated by linear grammars. This extension can be used to define regular languages as a particular case. Throughout this section, we will provide definitions and results which will make easier the subsequent methods for solving the analysis and synthesis problems related to linear languages.

Definition 3 *Let $\Delta = \{a_1, a_2, \ldots, a_n\}$ and $\Sigma = \{b_1, b_2, \ldots, b_m\}$ be two alphabets. We define the alphabet Δ indexed by Σ, denoted by Δ_Σ, as the set $\{a_{1_{b_1}}, a_{1_{b_2}}, \ldots, a_{1_{b_m}}, \ldots, a_{n_{b_1}}, \ldots, a_{n_{b_m}}\}$.*

Definition 4 *Let $G = (N, \Sigma, P, S)$ be a grammar. G is linear if every production in P is in one of the following forms.*

1. $A \to \alpha B \beta$, *where* $A, B \in N$ *and* $\alpha, \beta \in \Sigma^*$,
2. $A \to \alpha$, *where* $A \in N$ *and* $\alpha \in \Sigma^*$.

For every linear language, we can obtain the following normal form for a grammar that generates it.

1. $A \to aB \mid Ba$ where $A, B \in N$ and $a \in \Sigma$,
2. $A \to \lambda$ where $A \in N$.

From now on, we will deal with linear grammars in the previously defined normal form.

Definition 5 *Let* $\Delta = \Sigma_{\{L,R\}}$ *be an indexed alphabet. Given a string* x *over* Δ, *we define the image of* x *in* Σ, *denoted by* $im_\Sigma(x)$, *through the following rules:*

1. *if* $x = \lambda$, *then* $im_\Sigma(\lambda) = \lambda$,
2. *if* $x = a_L \cdot w$, *with* $w \in \Delta^*$, *then* $im_\Sigma(a_L \cdot w) = a \cdot im_\Sigma(w)$,
3. *if* $x = a_R \cdot w$, *with* $w \in \Delta^*$, *then* $im_\Sigma(a_R \cdot w) = im_\Sigma(w) \cdot a$.

As an extension of these rules, if $L \subseteq \Delta^*$ *then* $im_\Sigma(L) = \{im_\Sigma(x) : x \in L\}$.

Now, we can give a definition for the extended regular expressions that we call *linear expressions*.

Definition 6 *Let* $\Delta = \Sigma_{\{L,R\}}$ *be an indexed alphabet. A linear expression over* Δ *is defined in an inductive way by the following rules:*

1. $\emptyset$ *and* λ *are linear expressions,*
2. *for all* $a \in \Sigma$ a_L *and* a_R *are linear expressions,*
3. *if* r *is a linear expression then so is* (r),
4. *if* r *and* s *are linear expressions, then* $r + s$, rs *and* r^* *are linear expressions.*

Observe that any linear expression can be viewed as a regular one over $\Sigma_{\{L,R\}}$.

Any linear expression r over $\Sigma_{\{L,R\}}$ denotes a language $im_\Sigma(r)$ which is defined as follows

1. $im_\Sigma(\emptyset)$ is the empty language,
2. $im_\Sigma(\lambda) = \{\lambda\}$,
3. $im_\Sigma(a_L) = im_\Sigma(a_R) = \{a\}$,
4. $im_\Sigma((r)) = im_\Sigma(r)$,
5. $im_\Sigma(r + s) = im_\Sigma(r) \cup im_\Sigma(s)$,
6. $im_\Sigma(rs) = \{im_\Sigma(xy) : x \in L(r),\ y \in L(s)\}$ (see Definitions 1 and 5),
7. $im_\Sigma(r^*) = \{\lambda\} \cup im_\Sigma(rr^*)$.

Observe that if we consider any linear expression r over $\Sigma_{\{L,R\}}$ then $L(r)$ can be different from $im_\Sigma(r)$. So, the language that the expression r denotes as a regular expression is different from the one that it denotes as a linear expression.

Example 1 (a) The linear expression $(a_L b_R b_R)^*$ denotes the linear language defined by the set $\{a^i b^{2i} : i \in \mathbb{N}\}$.

(b) The linear expression $(a_L a_R + b_L b_R)^*$ denotes the linear language defined by $\{ww^r : w \in (a+b)^*\}$

Theorem 1 *If r is a linear expression over the indexed alphabet $\Sigma_{\{L,R\}}$, then $im_\Sigma(r)$ is a linear language.*

Proof. We will perform the proof as an induction process over the number of operations (unions, concatenations and closures) that appear in the linear expression in a way similar to Kleene's Theorem for regular expressions [10]. This means that we will provide an effective method for obtaining linear grammars in normal form for every linear expression.

Induction Basis If $r = \emptyset$, then the linear grammar $(\{S\}, \Sigma, \emptyset, S)$ generates $im_\Sigma(\emptyset) = \emptyset$

If $r = \lambda$, then the linear grammar $(\{S\}, \Sigma, \{S \to \lambda\}, S)$ generates the language $im_\Sigma(\lambda) = \lambda$.

$\forall a \in \Sigma$, if $r = a_L$, the corresponding linear grammar is as follows

$$(\{S, A\}, \{a\}, \{S \to aA;\ A \to \lambda\}, S).$$

$\forall a \in \Sigma$, if $r = a_R$, the linear grammar is given by

$$(\{S, A\}, \{a\}, \{S \to Aa;\ A \to \lambda\}, S).$$

Induction hypothesis Let r be a linear expression that contains a maximum number of n operations of unions, concatenations or closures with $n \geq 0$. Then there exists a linear grammar G_r that generates the language $im_\Sigma(r)$.

Induction step Let t be a linear expression that contains $n+1$ operations of unions, concatenations or closures. Let us analyse the different cases that can appear in t:

Case 1: $t = r + s$, with r and s being linear expressions that contain a maximum number of n operations. In this case, by induction hypothesis, there exist linear grammars $G_r = (N_r, \Sigma, P_r, S_r)$ and $G_s = (N_s, \Sigma, P_s, S_s)$ that generate the languages $im_\Sigma(r)$ and $im_\Sigma(s)$ respectively. We can assume, without loss of generality, that $N_r \cap N_s = \emptyset$. Thus, the linear grammar $G_t = (\{S_t\} \cup N_r \cup N_s, \Sigma, P, S_t)$ with $P = \{S_t \to \alpha : S_r \to \alpha \in P_r\} \cup \{S_t \to \beta : S_s \to \beta \in P_s\} \cup P_r \cup P_s$ generates the language $im_\Sigma(r) \cup im_\Sigma(s) = im_\Sigma(t)$.

Case 2: $t = rs$, with r and s being linear expressions that contain a maximum number of n operations. As in the previous case, by induction hypothesis, the linear grammars $G_r = (N_r, \Sigma, P_r, S_r)$ and $G_s = (N_s, \Sigma, P_s, S_s)$ generate the languages $im_\Sigma(r)$ and $im_\Sigma(s)$, respectively. Again, we can assume that $N_r \cap N_s = \emptyset$. We propose the following grammar that generates $im_\Sigma(t) = im_\Sigma(rs)$

$$G_t = (N_r \cup N_s, \Sigma, P'_r \cup P_s, S_r)$$

where P'_t is defined by the following rules:

- if $A \to aB \in P_r$, then $A \to aB \in P'_t$,
- if $A \to Ba \in P_r$, then $A \to Ba \in P'_t$,
- If $A \to \lambda \in P_r$, then $\forall \alpha : S_s \to \alpha \in P_s \ \ A \to \alpha \in P'_t$.

Case 3: $t = r^*$, with r being a linear expression that contains a maximum number of n operations. Again, by induction hypothesis, the linear grammar $G_r = (N_r, \Sigma, P_r, S_r)$ generates the language $im_\Sigma(r)$. We propose the following linear grammar to generate $im_\Sigma(t) = im_\Sigma(r^*)$:

$$G_t = (N_r, \Sigma, P_r \cup \{S_r \to \lambda\} \cup P'_r, S_r)$$

where P'_t is defined by the set

$$\{A \to \alpha : (A \to \lambda \in P_r) \wedge (S_r \to \alpha \in P_r)\}.$$ □

Example 2 From the proof of Theorem 1, we are going to construct a grammar that generates the language defined by the image of the linear expression $(a_L b_R b_R)^*$. The grammar will be defined step by step according to the application of every operation in the expression.

1. a_L:

$$S \to aA \qquad A \to \lambda$$

2. b_R:

$$S' \to A'b \qquad A' \to \lambda$$

3. $a_L b_R$:

$$S \to aA \qquad A \to A'b$$
$$S' \to A'b \qquad A' \to \lambda$$

The production $S' \to A'b$ can be deleted since it is useless.

4. $a_L b_R b_R$:

$$S \to aA \qquad A \to A'b \qquad A' \to A''b$$
$$S'' \to A''b \qquad A'' \to \lambda$$

As in the previous case, the production $S'' \to A''b$ is useless and can be deleted.

5. $(a_L b_R b_R)^*$:

$$S \to aA \mid \lambda \qquad A \to A'b$$
$$A' \to A''b \qquad A'' \to \lambda \mid aA$$

Obviously the last grammar generates the language $\{a^i b^{2i} : i \in \mathbb{N}\}$, which is the image of the linear expression $(a_L b_R b_R)^*$.

Now we are going to propose a solution for the analysis problem. Given a linear grammar, the problem is finding a linear expression whose image denotes the language generated by the grammar. The algorithm to apply in the resolution of this problem is based on a reduction of the linear grammar to a regular one and the subsequent resolution of the regular grammar using well known methods [2, 6]. Finally, the regular expression gives the desired linear expression. We provide the following definition which will help us to carry out this task

Definition 7 *Let $G = (N, \Sigma, P, S)$ be a linear grammar in normal form. We define the* extended regular grammar *of G and we denote it by G_{er} as the tuple $(N, \Sigma_{\{L,R\}}, P', S)$, where P' is defined by the following rules:*

1. *if $A \to \lambda \in P$, then $A \to \lambda \in P'$,*
2. *if $A \to aB \in P$, then $A \to a_L B \in P'$,*
3. *if $A \to Ba \in P$, then $A \to a_R B \in P'$.*

Lemma 2 *Let G be a linear grammar in normal form and G_{er} be its extended regular grammar. Then $x \in L(G_{er})$, if and only if $im_\Sigma(x) \in L(G)$.*

Proof. First, we will see that $x \in L(G_{er})$ implies that $im_\Sigma(x) \in L(G)$. The derivation sequence of x in G_{er} will have the following form:

$$S \overset{\alpha_1}{\underset{G_{er}}{\Rightarrow}} x_1 A_1 \overset{\alpha_2}{\underset{G_{er}}{\Rightarrow}} x_1 x_2 A_2 \overset{\alpha_3}{\underset{G_{er}}{\Rightarrow}} \cdots \overset{\alpha_{n-1}}{\underset{G_{er}}{\Rightarrow}} x_1 x_2 \cdots x_{n-1} A_{n-1} \overset{\alpha_n}{\underset{G_{er}}{\Rightarrow}} x_1 x_2 \cdots x_n = x$$

where every production α_i takes the form $A_{i-1} \to x_i A_i$. Therefore, by choosing the productions in G that define every α_i, which we denote by α'_i, we can obtain the following derivation sequence in G:

$$S \overset{\alpha'_1}{\underset{G}{\Rightarrow}} \gamma_1 A_1 \phi_1 \overset{\alpha'_2}{\underset{G}{\Rightarrow}} \gamma_2 A_2 \phi_2 \overset{\alpha'_3}{\underset{G}{\Rightarrow}} \cdots \overset{\alpha'_{n-1}}{\underset{G}{\Rightarrow}} \gamma_{n-1} A_{n-1} \phi_{n-1} \overset{\alpha'_n}{\underset{G}{\Rightarrow}} \gamma_n \phi_n$$

where $\gamma_i, \phi_i \in \Sigma^*$ for $i = 1, \ldots, n$, and it easy to prove that $\gamma_i \phi_i = im_\Sigma(x_1 \ldots x_i)$ for $i = 1, \ldots, n$. Therefore, we can conclude that $\gamma_n \phi_n = im_\Sigma(x)$ and $S \overset{*}{\underset{G}{\Rightarrow}} im_\Sigma(x)$. Thus, $im_\Sigma(x) \in L(G)$ as was previously stated.

The other implication to be proved, $im_\Sigma(x) \in L(G) \Rightarrow x \in L(G_{er})$, can be performed as before. □

Example 3 Given the linear grammar defined by the following rules

$$S \to aA \mid bB, \qquad A \to Aa \mid bB,$$
$$B \to aC \mid Bb, \qquad C \to \lambda,$$

its extended regular grammar is defined by the following productions

$$S \to a_L A \mid b_L B, \qquad A \to a_R A \mid b_L B,$$
$$B \to a_L C \mid b_R B, \qquad C \to \lambda.$$

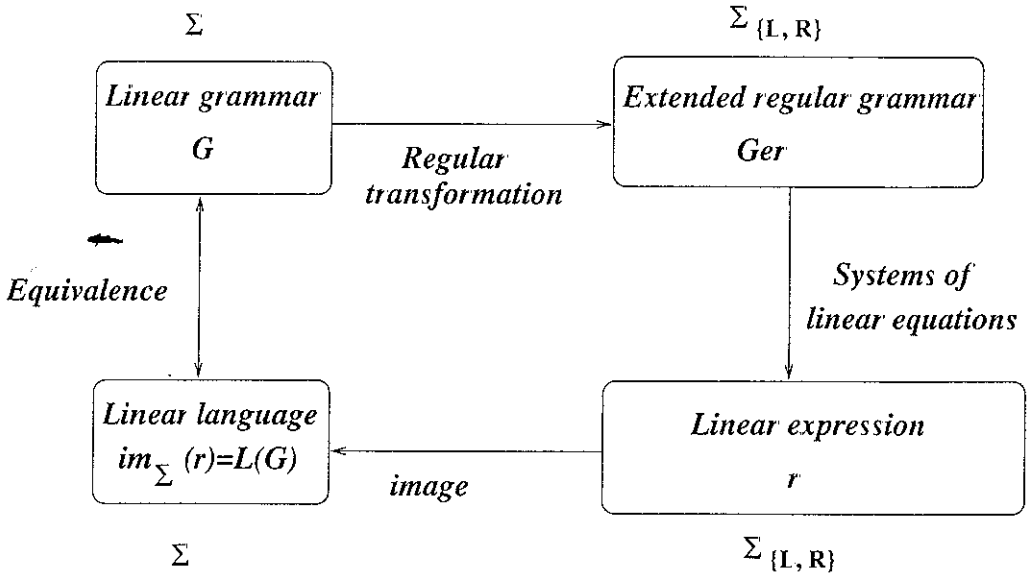

Figure 1: A scheme to obtain the language generated by a linear grammar

Theorem 3 *For every linear grammar G, there exists a linear expression r such that $L(G) = im_\Sigma(r)$.*

Proof. By applying the scheme of Figure 1, we can obtain a linear expression for the given linear grammar.

We construct the extended regular grammar G_{er} from G. Then, by the methods shown in [2], we can calculate a regular expression r to denote $L(G_{er})$. This expression is a linear expression that denotes $im_\Sigma(L(G))$ as established in Lemma 1. □

Example 4 Given the extended regular grammar of Example 3, its linear expression can be obtained by solving the following equation system:

$$\left.\begin{array}{l} S = a_L A + b_L B \\ A = a_R A + b_L B \\ B = a_L C + b_R B \\ C = \lambda \end{array}\right\}$$

The linear expression associated to this equation system is obtained from the solutions of the system which are the following:

$$C = \lambda,$$
$$B = b_R^* a_L,$$
$$A = a_R^* b_L b_R^* a_L,$$
$$S = a_L a_R^* b_L b_R^* a_L + b_L b_R^* a_L$$

4. Another Synthesis Algorithm

Once we have established the equivalence between linear expressions and linear languages, we introduce a different method to solve the synthesis problem which is based on the derivative method proposed by BROZOZOWSKI [1]. So, we propose a method for obtaining linear grammars from the derivatives of linear expressions with respect to any string in the indexed alphabet

From the derivative rules, we can deduce a method for obtaining a linear grammar which is equivalent to a given linear expression. Therefore, if t is a linear expression over $\Sigma_{\{L,R\}}$, then by $D(t)$ we denote the set of all the different derivatives of the linear expression with respect to the strings of the alphabet. This set is finite, given that t is a regular expression over the indexed alphabet. The construction of a linear grammar which is equivalent to the linear expression is carried out as follows:

$G = (N, \Sigma, P, S)$ where $N = D(t)$, $S = der_\lambda(t)$, P is defined by the following rules with $x \in \Sigma^*_{\{L,R\}}$:

$$der_x(t) \to a \cdot der_{xa_L}(t),$$
$$der_x(t) \to der_{xa_R}(t) \cdot a,$$
$$\text{If } \lambda \in im_\Sigma(der_x(t)) \text{ then } der_x(t) \to \lambda.$$

Example 5 Given the linear expression $(a_L b_R b_R)^*$, the set of all the different derivatives with respect to $\{a, b\}_{\{L\,R\}}$ is calculated as follows

$$der_\lambda((a_L b_R b_R)^*) = (a_L b_R b_R)^*,$$
$$der_{a_L}((a_L b_R b_R)^*) = b_R b_R (a_L b_R b_R)^*,$$
$$der_{b_L}((a_L b_R b_R)^*) = \emptyset,$$
$$der_{a_R}((a_L b_R b_R)^*) = \emptyset,$$
$$der_{b_R}((a_L b_R b_R)^*) = \emptyset,$$
$$der_{a_L a_L}((a_L b_R b_R)^*) = \emptyset,$$
$$der_{a_L b_L}((a_L b_R b_R)^*) = \emptyset,$$
$$der_{a_L a_R}((a_L b_R b_R)^*) = \emptyset,$$
$$der_{a_L b_R}((a_L b_R b_R)^*) = b_R (a_L b_R b_R)^*,$$
$$der_{a_L b_R a_L}((a_L b_R b_R)^*) = \emptyset,$$
$$der_{a_L b_R b_L}((a_L b_R b_R)^*) = \emptyset,$$
$$der_{a_L b_R a_R}((a_L b_R b_R)^*) = \emptyset,$$
$$der_{a_L b_R b_R}((a_L b_R b_R)^*) = (a_L b_R b_R)^*.$$

From these derivatives, a linear grammar which is equivalent to the linear expression is obtained using the auxiliary symbols $S = der_\lambda((a_L b_R b_R)^*)$, $A = der_{a_L}((a_L b_R b_R)^*)$ and $B = der_{a_L b_R}((a_L b_R b_R)^*)$. The productions of the grammar are the following:

$$S \to aA \mid \lambda, \qquad A \to Bb, \qquad B \to Sb.$$

5. Equivalence and Structural Equivalence for Linear Grammars

From linear expressions, we can offer a different point of view to the resolution of some problems which are related to linear languages such as the equivalence problem between linear grammars [12] or the structural equivalence problem for linear grammars [8, 9, 11]. The first problem was proved to be unsolvable by ROZENBERG [12], so we cannot make any progress related to this. However, we can provide a different method for solving the second problem which, unfortunately, maintains its level of complexity.

First, let us consider a result which relates the equivalence problem for linear grammars to the results presented in this work. This relation is established in the following theorem.

Theorem 4 *The equivalence problem for linear expressions is unsolvable. That is, given two different linear expressions r and s, there does not exist an effective procedure to establish whether $im_\Sigma(r) = im_\Sigma(s)$*

Proof. The equivalence problem for linear grammars was proven to be unsolvable by ROZENBERG [12]. It is easy to reduce the equivalence problem for linear grammars to the equivalence problem for linear expressions. Consequently, the problem stated in the theorem is also unsolvable. □

The structural equivalence problem for linear grammars is stated as follows: Given two linear grammars G_1 and G_2, the solution consists of determining whether the set of derivation skeletons of G_1 (the set of derivations of G_1 where the auxiliary symbols are not distinguished) is equal to the set of derivation skeletons of G_2. This problem was proved to be PSPACE-complete [8, 9, 11]. We can reduce the structural equivalence problem for linear grammars to the equivalence problem for regular ones. The equivalence problem for regular grammars was also proved to be PSPACE-complete [8].

Given two linear grammars G and G', the structural equivalence problem can be established from the equivalence problem between G_{er} and G'_{er}.

Theorem 5 *Let G and G' be linear grammars in normal form and G_{er} and G'_{er} be their corresponding extended regular grammars. Then G is structurally equivalent to G' if and only if G_{er} is equivalent to G'_{er}*

Proof. Trivial from the proof of Lemma 1. □

We can extend the last result to linear grammars as follows

Theorem 6 *Let G_1 and G_2 be linear grammars. Then there exist linear grammars in normal form G'_1 and G'_2 which are equivalent to G_1 and G_2 respectively, such that*

(a) *if G_1 is structurally equivalent to G_2 then G'_1 is structurally equivalent to G'_2,*

(b) *if G'_1 is structurally equivalent to G'_2 then G_1 is equivalent to G_2.*

Proof. Let us obtain G'_1 and G'_2 from G_1 and G_2 as follows: For every production in G_1 (or G_2) in the form $A \to a_1 \ldots a_n B b_1 \ldots b_m$ substitute it by the set of productions $A \to a_1 A_1, \ldots, A_{n-1} \to a_n B_1$ and $B_1 \to B_2 b_m, \ldots, B_m \to B b_1$. For every production in the form $A \to a_1 \ldots a_n$ substitute it by the set of productions $A \to a_1 A_1, \ldots, A_{n-1} \to a_n A_n$ and $A_n \to \lambda$. The productions $A \to a_1 \ldots a_n B$ and $A \to B b_1 \ldots b_m$ are transformed in a similar way.

(a) Suppose that G_1 is structurally equivalent to G_2, then trivially so are G'_1 and G'_2.

(b) On the other hand, if G'_1 is structurally equivalent to G'_2 then G_1 is (not necessarily structurally) equivalent to G_2. The factorization of the production rules to obtain G'_1 and G'_2, obviously, is not an injective mapping. □

6. Conclusions and Future Work

Throughout the present paper, we have presented linear expressions as a new formalism for defining linear languages. This proposal has allowed us to establish new methods for solving the analysis and synthesis problems which are related to linear grammars. Finally, we have proposed a new method for solving the structural equivalence problem for linear grammars.

Future work related to this paper can be summarized as follows:

1. We can apply indexed alphabets to define other language classes such as context-free ones. In this case, it would not be enough to use an indexed alphabet such as $\Sigma_{\{L,R\}}$, since, in general, the structural relationships between the auxiliary symbols in the grammar are more complex than in the linear case. Therefore, we should use a different indexed alphabet to take into account some relations such as precedence between symbols and the number of symbols in every production.
2. If we turn our attention to studying only the structural aspects of the grammar, then we can use the same indexed alphabet $\Sigma_{\{L,R\}}$, but we need to define an image over $\{L, R\}$. We could also study some aspects of linear grammars such as the number of linear changes from left to right (right to left) that are carried out during the derivation of any string in the grammar (i. e. its reversal complexity). From this study, we might be able to impose new normal forms on the structure of the grammar.

Acknowledgements

The author is grateful to ERKKI MÄKINEN and PEDRO GARCÍA for helpful comments and discussion on this work. Also, he is grateful to SHENG YU for nice comments and suggestions made during the Workshop on *Descriptional Complexity of Automata, Grammars and Related Structures* in Magdeburg, July 1999. Sharp remarks and suggestions made by the anonymous referees are also acknowledged.

References

[1] J. BROZOZOWSKI, Derivatives of Regular Expressions *Journal of the Association for Computing Machinery* **11** (1964) 4, 481–494

[2] J. CARROLL, D. LONG, *Theory of Finite Automata*. Prentice-Hall, 1989.

[3] J. GRUSKA, A Characterization of Context-free languages. *Journal of Computer and System Sciences* **5** (1971), 353–364.

[4] K. HASHIGUCHI, H. YOO, Extended regular expressions of star degree at most two. *Theoretical Computer Science* **76** (1990), 272–284.

[5] K. HASHIGUCHI, The Infinite 2-Star Height Hierarchy of Extended Regular Languages of Star Degree at Most Two. *Information and Computation* **114** (1994), 237–246.

[6] J. HOPCROFT, J. ULLMAN, *Introduction to Automata Theory, Languages and Computation* Addison-Wesley Publishing Company, 1979.

[7] J. HROMKOVIČ, S. SEIBERT, T. WILKE, *Translating Regular Expressions into Small ϵ-Free Nondeterministic Finite Automata*. In: R. REISCHUK, M. MORVAN (eds.), *Proc. 14th Annual Symposium on Theoretical Aspects of Computer Science (STACS'97)*. LNCS **1200**, Springer-Verlag, 1997, 55–66.

[8] H. HUNT III, D. ROSENKRANTZ, T. SZYMANSKI, On the Equivalence, Containment, and Covering Problems for Regular and Context-Free Languages. *Journal of Computer and System Sciences* **12** (1976), 222–268.

[9] H. HUNT III, D. ROSENKRANTZ, T. SZYMANSKI, The covering problem for linear context-free grammars. *Theoretical Computer Science* **2** (1976), 361–382.

[10] S. C. KLEENE, Representation of Events in Nerve Nets and Finite Automata. In: C. E. SHANNON, J. MCCARTHY (eds.), *Automata Studies*. Princeton University Press, 1956, 3–41.

[11] M. PAULL, S. UNGER, Structural Equivalence of Context-Free Grammars. *Journal of Computer and System Sciences* **2** (1968), 427–463.

[12] G. ROZENBERG, Direct Proofs of the Undecidability of the Equivalence Problem for Sentential Forms of Linear Context-Free Grammars and the Equivalence Problem for 0L Systems. *Information Processing Letters* **1** (1972), 233–235.

[13] A. SALOMAA, *Formal Languages*. Academic Press, 1973.

[14] M. K. YNTEMA, Cap Expressions for Context-Free Languages. *Information and Control* **18** (1971), 311–318.

[15] T. YOKOMORI, Inductive Inference of Context-free Languages Based on Context-free Expressions. *International J. Computer Math.* **24** (1988), 115–140.

[16] H. YOO, K. HASHIGUCHI, Extended automata-like regular expressions of star degree at most (2,1). *Theoretical Computer Science* **88** (1991), 351–363.

(Received: September 2, 1999; revised: February 11, 2000)