A Note on the Equivalence and Complexity of Linear Grammars

JOSÉ M. SEMPERE
Departamento de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, Camino de Vera s/n 46022 Valencia, Spain.
E-mail: jsempere@dsic.upv.es

Abstract. Linear languages can be characterized by regular-like expressions (*linear expressions*) according to a previous work. In this paper, we consider some equivalence properties of linear expressions in order to obtain a characterization of reversal and Kolmogorov complexity of linear languages. First, we introduce the relationship between regular expressions equivalence properties and linear expressions equivalence properties. Then, we define *permutation* and *compression* equivalence properties in order to handle linear expressions to obtain shorter equivalent ones. The study of reversal and Kolmogorov complexities associated to linear grammars is performed in the rest of the paper. We obtain a speed-up theorem for reversal complexity. Finally, we define a Kolmogorov-like complexity associated to linear grammars and we deduce upper bounds for such complexity measure.

Key words: formal languages, linear grammars, equivalence properties, reversal complexity, Kolmogorov complexity

1. Introduction

Regular-like expressions have been proposed by different authors in order to describe formal language classes. We can refer some works in such direction (Gruska, 1971; Yntema, 1971; Hashiguchi and Yoo, 1990; Yoo and Hashiguchi, 1991; Hashiguchi, 1994). In a previous work (Sempere, 2000), a class of regular-like expressions (*linear expressions*) has been proposed in order to describe the languages generated by linear grammars. So, *linear expressions* are regular-like expressions that take into account the structural information of linear grammars. Any linear expression denotes the set of strings that belong to a given linear grammar together with the way in which every string is derived from the axiom. Here, the derivation of a string is quite simple, given that, in any sentential form, there can only be a single auxiliary symbol, and the relevant information is defined in terms of linear changes from right to left (left to right) during the derivation steps.

We can study some aspects of linear languages by taking advantage of linear expressions. In this work we will focus on two different complexity measures of linear languages: *reversal* and *Kolmogorov* complexities. Reversal measure can be defined as the number of changes that occur in any derivation sequence in any grammar that generates a language, while Kolmogorov complexity is the minimum size of any program which describes the linear language generated by the grammar, together with the way in which every string is obtained. Both measures can be

Grammars **6:** 115–126, 2003.
© 2003 *Kluwer Academic Publishers. Printed in the Netherlands.*

easily related to linear expressions. So, given that in any linear expression we can observe how the linear changes occur then we can calculate its reversal complexity. By the other hand, any program that writes the linear expression of a given linear grammar denotes an upper bound of the Kolmogorov complexity of its language.

This work is structured as follows. First, some basic concepts concerning linear expressions and some equivalence properties are proposed. We use a normal form for linear expression in order to obtain a short description which will allow us to study the reversal and Kolmogorov complexities. Then, we focus on the study of some aspects of reversal complexity and Kolmogorov complexities. We study the reversal complexity of any linear grammar in normal form. Here, we will relate the reversal complexity of any linear grammar with the time complexity of regular languages. From the upper bound obtained before, we deduce a speed-up theorem based on linear expressions. Finally, we propose some results about the Kolmogorov complexity of linear languages from previously transformed linear grammars.

2. Basic Concepts and Linear Expressions

In this section, we review 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. Most of these concepts and results can be viewed in Sempere (2000).

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

Given any alphabet Γ, the set of all strings over Γ will be denoted by Γ^*. Given any string $x \in \Gamma^*$ the reverse of the string x will be denoted by x^r and its length by $|x|$. The empty string will be denoted by λ.

DEFINITION 2. 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 grammar, we can obtain an equivalent grammar in the following normal form:

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 work with linear grammars in the previously defined normal form. We will say that the language L is linear if there exists a linear grammar G such that $L = L(G)$. We will denote the class of linear languages by $\mathcal{LIN}$.

DEFINITION 3. Let Σ be an alphabet and $\Delta = \Sigma_{\{L,R\}}$. 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 a generalization of these rules, if $L \subseteq \Delta^*$ then $im_\Sigma(L) = \{im_\Sigma(x) : x \in L\}$.

Now, we will define regular-like expressions for linear languages.

DEFINITION 4. 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 $\Sigma_{\{L,R\}}$ and it defines a regular language $L(r)$. By the other hand, 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. $im_\Sigma(rs) = \{im_\Sigma(xy) : x \in L(r), y \in L(s)\}$[1],
7. $im_\Sigma(r^*) = \{\lambda\} \cup im_\Sigma(rr^*)$.

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

THEOREM 1. *(*Sempere, 2000*) Let $\Sigma_{\{L,R\}}$ be an indexed alphabet. Then $L \subseteq \Sigma^*$ is a linear language iff there exists a linear expression r over $\Sigma_{\{L,R\}}$ such that $im_\Sigma(r) = L$.*

We give the following example of linear languages and linear expressions.

EXAMPLE 1. (a) The linear expression $(a_L b_R b_R)^*$ denotes the linear language defined by the set $\{a^i b^{2i} : i \geqslant 0\}$.

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

3. Equivalence Properties

We have shown the relationship between linear grammars in normal form and linear expressions. Now, we will provide some equivalence properties in order to obtain some results on complexity.

DEFINITION 5. Let r and s be linear expressions over $\Sigma_{\{L,R\}}$. We will say that r and s are regular-equivalent, and we will denote it by $r \equiv_{REG} s$, iff $L(r) = L(s)$. We will say that r and s are linear-equivalent, and we will denote it by $r \equiv_{LIN} s$, iff $im_\Sigma(r) = im_\Sigma(s)$.

PROPERTY 1. Let r and s be linear expressions over $\Sigma_{\{L,R\}}$. If $r \equiv_{REG} s$ then $r \equiv_{LIN} s$.

Proof. Let r and s be regular-equivalent linear expressions over $\Sigma_{\{L,R\}}$. Obviously, $x \in L(r)$ iff $x \in L(s)$. Now, let us suppose that $x \in im_\Sigma(r)$. Clearly, there exists $\overline{x} \in L(r)$ such that $im_\Sigma(\overline{x}) = x$. So, $\overline{x} \in L(s)$ given that $r \equiv_{REG} s$ and $x \in im_\Sigma(s)$. The result from $im_\Sigma(s)$ can be proved in a similar way, so $im_\Sigma(r) = im_\Sigma(s)$ and $r \equiv_{LIN} s$. □

EXAMPLE 2. The converse result of Property 1 is not true in general. Observe that $a_L b_R \equiv_{LIN} b_R a_L$ while $a_L b_R \not\equiv_{REG} b_R a_L$.

Now, we will provide some equivalence properties between linear expressions that do not preserve the regular-equivalence.

3.1. PERMUTATION

Permutation attempts to produce an order between the symbols of any linear expression. Here, the result will produce that, in any string, the symbols indexed by R appear later than the symbols indexed by L. We will formalize this operation with the function $\psi : \Sigma^*_{\{L,R\}} \to \Sigma^*_{\{L,R\}}$ defined as follows:

1. $\psi(\lambda) = \lambda$,
2. $(\forall a \in \Sigma)(\forall x \in \Sigma^*_{\{L,R\}})\ \psi(a_R x) = \psi(x) a_R$,
3. $(\forall a \in \Sigma)(\forall x \in \Sigma^*_{\{L,R\}})\ \psi(a_L x) = a_L \psi(x)$.

Obviously, if $x \in \Sigma^*_{\{L,R\}}$ then $\psi(x) \in \Sigma^*_{\{L\}}\Sigma^*_{\{R\}}$. Then, for every string x, we can set $\psi(x) = x_1 x_2$ where $x_1 \in \Sigma^*_{\{L\}}$ and $x_2 \in \Sigma^*_{\{R\}}$. Now we can define the transformation $\phi : \Sigma^*_{\{L,R\}} \to \Sigma^*_{\{L,R\}}$ as follows:

$$(\forall x \in \Sigma^*_{\{L,R\}}) \quad \phi(x) = x_1 x_2^r \quad \text{with} \quad \psi(x) = x_1 x_2.$$

We can formulate equivalence properties based on the function ϕ as follows.

PROPERTY 2. Let $x \in \Sigma^*_{\{L,R\}}$, then $x \equiv_{LIN} \phi(x)$.

Proof. We will prove the property by induction over the length of every string x. First, we will take λ as our induction basis. Obviously, $\phi(\lambda) = \lambda$ and $x \equiv_{LIN} \phi(x)$.

Next, we enunciate an induction hypothesis such that for every string x of length up to n $x \equiv_{LIN} \phi(x)$.

Finally, we will take x with length $n+1$. We can consider two different cases. First, let $x = a_L\overline{x}$. Here, $\phi(x) = a_L\phi(\overline{x})$, and $im_\Sigma(\phi(x)) = a \cdot im_\Sigma(\phi(\overline{x})) = a \cdot im_\Sigma(\overline{x}) = im_\Sigma(x)$.

By the other hand, let $x = a_R\overline{x}$. Then $\phi(x) = \overline{x}_1(\overline{x}_2 a_R)^r = \overline{x}_1 a_R(\overline{x}_2)^r$ with $\overline{x}_1 \in \Sigma_L^*$ and $\overline{x}_2 \in \Sigma_R^*$. So, $im_\Sigma(\phi(x)) = im_\Sigma(\overline{x}_1) \cdot im_\Sigma(a_R(\overline{x}_2)^r) = im_\Sigma(\overline{x}_1) im_\Sigma((\overline{x}_2)^r)a = im_\Sigma(\phi(\overline{x}))a = im_\Sigma(\overline{x})a = im_\Sigma(x)$. □

Now, once we have proved the previous property we can extend the function ϕ to act over sets of strings. So, let L be a language defined over $\Sigma_{\{L,R\}}$ and $\phi(L) = \{\phi(x) \mid x \in L\}$. We can define the function ϕ over linear expressions as follows:

1. $\phi(\emptyset) = \emptyset$,
2. $\phi(\lambda) = \lambda$,
3. $\phi(a_R) = a_R$ and $\phi(a_L) = a_L$,
4. for every linear expression r, $\phi((r)) = (\phi(r))$,
5. for every pair of linear expressions r and s, $\phi(r+s) = \phi(r) + \phi(s)$,
6. for every pair of linear expressions r and s, $\phi(rs) = \{\phi(xy) : x \in L(r), y \in L(s)\}$,
7. for every linear expression r, $\phi(r^*) = (\phi(r))^*$.

From the previous rules and Property 2, we can deduce the following result.

COROLLARY 1. *Let r and s be linear expressions. Then, the following equivalence properties hold:*

1. $r + s \equiv_{LIN} \phi(r) + \phi(s)$,
2. $rs \equiv_{LIN} \phi(rs)$,
3. $r^* \equiv_{LIN} (\phi(r))^*$.

EXAMPLE 3. Let $r = a_L b_R a_L + (a_R a_L + b_R b_R)^*$. Then r is equivalent to $\phi(r) = a_L a_L b_R + (a_L a_R + b_R b_R)^*$.

Let $s = a_R a_R b_L^*$. Then, s is equivalent to $\phi(s) = b_L^* a_R a_R$.

3.2. COMPRESSION

Compression attempts to reduce the number of indexes in any linear expression. Here, we will obtain a more economic expression where the symbols with equal index are joined into a single word. It will be especially useful to fix an upper bound of the Kolmogorov complexity of any linear language. We will formalize this operation with the function φ as follows:

$$(\forall x \in \Sigma_{\{L,R\}}^*) \quad \varphi(x) = [im_\Sigma(x_1)]_L\,[im_\Sigma(x_2)]_R, \quad \text{with} \quad \phi(x) = x_1 x_2.$$

Observe that the function φ makes a representation change and its result is defined in terms of strings with at most one index L or R. We can define $\{\Sigma^*\}_{\{L\}}$ as the set of strings of Σ with the unique index L. In the same way, we define $\{\Sigma^*\}_{\{R\}}$ as the set of strings of Σ with the unique index R. We will define the function im_Σ to act over φ as follows:

$$(\forall x \in \Sigma^*_{\{L,R\}}) \quad im_\Sigma(\varphi(x)) = im_\Sigma(x_1)im_\Sigma(x_2) \quad \text{with} \quad \phi(x) = x_1x_2.$$

Now, we can obtain the following equivalence properties with respect to the function φ.

PROPERTY 3. Let $x \in \Sigma^*_{\{L,R\}}$, then $x \equiv_{LIN} \varphi(x)$.

Proof. We will prove this statement as in Property 2, that is through an induction process over the length of every string x.

First, let us take $x = \lambda$. Here, $\varphi(\lambda) = \lambda$ and $im_\Sigma(\lambda) = im_\Sigma(\varphi(\lambda))$. So, $x \equiv_{LIN} \varphi(x)$.

Let us make enunciate an induction hypothesis such that for every string x of length up to n, $x \equiv_{LIN} \varphi(x)$.

Now, we will take strings of length $n + 1$. First, let us take $x = a_L\overline{x}$. Then $\varphi(a_L\overline{x}) = [a \cdot im_\Sigma(x_1)]_L[im_\Sigma(x_2)]_R$. Here, the following equalities hold:

$$im_\Sigma([a\cdot im_\Sigma(x_1)]_L[im_\Sigma(x_2)]_R) = a\cdot im_\Sigma(x_1)im_\Sigma(x_2) = a\cdot im_\Sigma(\overline{x}) = im_\Sigma(x).$$

By the other hand, let us take $x = a_R\overline{x}$. Here, the following equalities hold:

$$\varphi(a_R\overline{x}) = [im_\Sigma(x_1)]_L[im_\Sigma(a_Rx_2)]_R = [im_\Sigma(x_1)]_L[im_\Sigma(x_2)\cdot a]_R.$$

So, $im_\Sigma(\varphi(a_R\overline{x})) = im_\Sigma(x_1)im_\Sigma(x_2)\cdot a = im_\Sigma(a_R\overline{x}) = im_\Sigma(x)$. □

Now, once we have proved the previous property we can extend the function φ to act over sets of strings. So, let L be a language defined over $\Sigma_{\{L,R\}}$ and $\varphi(L) = \{\varphi(x) \mid x \in L\}$. We can define the function φ over linear expressions as follows:

1. $\varphi(\emptyset) = \emptyset$,
2. $\varphi(\lambda) = \lambda$,
3. $\varphi(a_R) = a_R$ and $\varphi(a_L) = a_L$,
4. for every linear expression r, $\varphi((r)) = (\varphi(r))$,
5. for every pair of linear expressions r and s, $\varphi(r + s) = \varphi(r) + \varphi(s)$,
6. for every pair of linear expressions r and s, $\varphi(rs) = \{\varphi(xy) : x \in L(r), y \in L(s)\}$,
7. for every linear expression r, $\varphi(r^*) = (\varphi(r))^*$.

From the previous rules and Property 3, we can deduce the following result.

COROLLARY 2. *Let r and s be linear expressions. Then, the following equivalence properties hold:*

1. $r + s \equiv_{LIN} \varphi(r) + \varphi(s)$,
2. $rs \equiv_{LIN} \varphi(rs)$,
3. $r^* \equiv_{LIN} (\varphi(r))^*$.

EXAMPLE 4. Let $r = a_L b_R a_L + (a_R a_L + b_R b_R)^*$. Then r is equivalent to $\varphi(r) = (aa)_L b_R + (a_L a_R + (bb)_R)^*$.

Let $s = a_L(b_R a_L b_L)^* b_L$. Then s is equivalent to $\varphi(s) = a_L((ab)_L b_R)^* b_L$.

4. Reversal Complexity of Linear Languages

The deterministic reversal complexity of a language (Wagner et al., 1986; Chen and Yap, 1991) is defined as the maximum number of tape head reversals during a deterministic Turing machine computation (i.e., the maximum number of direction changes that the tape heads of a deterministic Turing machine make during the processing of any input string). Let $DREVERSAL_k(f)$ be the class of languages accepted by k-tape deterministic Turing machines which make at most $\mathcal{O}(f(n))$ tape reversals on inputs of length n. We will denote by $DREVERSAL(f) = \bigcup_{k \geqslant 1} DREVERSAL_k(f)$. We can give a similar definition for nondeterministic Turing machines, so $NREVERSAL_k(f)$ denotes the class of languages accepted by k-tape nondeterministic Turing machines which make at most $\mathcal{O}(f(n))$ tape reversals on inputs of length n. It has been proved that $\mathcal{LIN} = NREVERSAL_1(1)$ (Wagner et al., 1986).

Here, we study the reversal complexity related to linear grammars. Obviously, there exists a strong connection between the linear changes that a Turing machine performs during its computation and the linear changes that a linear grammar carries out during a derivation process.

In order to define the number of linear changes from left to right (right to left) in a derivation process, we will focus our attention to the subscripts (L or R) of every symbol in a linear expression. Here, the subscripts give enough information to deduce the reversal complexity of a linear language generated by a given linear grammar.

DEFINITION 6. Let G be a linear grammar. We will define the reversal complexity of $L(G)$ as the maximum number of changes (left to right and right to left) that occur whenever $x \in L(G)$ is derived from the axiom of the grammar. Obviously, this complexity is defined as a function that goes from positive integers (i.e., the length of any string $x \in L(G)$) to positive integers (the number of changes).

We will denote the family of linear grammars in normal form by $\mathcal{GNF}$. So, $REVERSAL_{\mathcal{GNF}}(f)$ will denote the family of languages that can be generated by linear grammars in normal form with reversal complexity $f(n)$.

EXAMPLE 5. (a) Let G_1 be the linear grammar defined by the rules $S \to aA|\lambda$; $A \to Sb$. Then, $L(G_1) = \{a^n b^n : n \geqslant 0\}$. Then $L(G_1)$ can be described by the linear expression $(a_L b_R)^*$ and, obviously, $L(G_1) \in REVERSAL_{\mathcal{GNF}}(n)$.

(b) Let G_2 be the linear grammar defined by the rules $S \to aS|bA$; $A \to bA|\lambda$. Then, $L(G_2) = \{a^n b^m : n \geqslant 0, m > 1\}$. Then, $L(G_2)$ can be described by the linear expression $(a_L)^* b_L (b_L)^*$ and $L(G_2) \in REVERSAL_{\mathcal{G},\mathcal{NF}}(0)$.

Given that any linear grammar can be described by a linear expression, we can deduce the reversal complexity of the grammar as the number of L and R index changes in the expression. So, we can define the *reversal language* of a linear expression as $im_{\{L,R\}}(r)$ according to the following rules:

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

For any linear expression, every string of its reversal language will be called a *reversal word*. Any derivation process in a linear grammar has a reversal word associated, and it describes the number of linear changes that occur during the derivation process.

EXAMPLE 6. (a) Let $r = (a_L b_R b_R)^*$, then $im_{\{L,R\}}(r) = (LRR)^*$.
(b) Let $s = (a_L a_R + b_L b_R)^*$, then $im_{\{L,R\}}(s) = (LR)^*$.
(c) Let $t = a_L b_R a_L + (a_R a_L + b_R b_R)^*$, then $im_{\{L,R\}}(t) = LRL + (RL + RR)^*$.

THEOREM 2. *Let r be a linear expression over* $\Sigma_{\{L,R\}}$. *Then* $im_{\{L,R\}}(r)$ *is regular.*

Proof. We will construct a (right linear) regular grammar for every linear expression.

1. $r = \emptyset$
   Here the regular grammar $(\{S\}, \{L, R\}, \emptyset, S)$ generates the empty set.
2. $r = \lambda$
   The grammar $(\{S\}, \{L, R\}, \{S \to \lambda\}, S)$ generates the language $\{\lambda\}$.
3. $r = a_L$
   The grammar $(\{S\}, \{L, R\}, \{S \to L\}, S)$ generates $\{L\}$.
4. $r = a_R$
   The grammar $(\{S\}, \{L, R\}, \{S \to R\}, S)$ generates $\{R\}$.
5. $r = s + t$
   Let us suppose that regular grammars $G_s = (N_s, \{L, R\}, P_s, S_s)$ and $G_t = (N_t, \{L, R\}, P_t, S_t)$ exist such that $L(G_s) = im_{\{L,R\}}(s)$ and $L(G_t) = im_{\{L,R\}}(t)$ with $N_s \cap N_t = \emptyset$. The grammar $(N_s \cup N_t \cup \{S_r\}, \{L, R\}, P_s \cup P_t \cup \{S_r \to S_s \mid S_t\}, S_r)$ generates the reversal language $im_{\{L,R\}}(r)$.
6. $r = st$
   Again, the regular grammars $G_s = (N_s, \{L, R\}, P_s, S_s)$ and $G_t = (N_t, \{L, R\},$

$P_t, S_t)$ generate $im_{\{L,R\}}(s)$ and $im_{\{L,R\}}(t)$ respectively with $N_s \cap N_t = \emptyset$. The grammar $(N_s \cup N_t, \{L, R\}, P'_s \cup P_t, S_s)$ where P'_s is defined by the rules:
(a) $(\forall A, B \in N_s)\ (\forall a \in \{L, R, \lambda\})\ A \to aB \in P_s \Rightarrow A \to aB \in P'_s$,
(b) $(\forall A \in N_s)\ (\forall a \in \{L, R, \lambda\})\ A \to a \in P_s \Rightarrow A \to aS_t \in P'_s$,
generates the reversal language $im_{\{L,R\}}(r)$.

7. $r = s^*$
Here, $G_s = (N_s, \{L, R\}, P_s, S_s)$ generates $im_{\{L,R\}}(s)$, and the grammar $G_r = (N_s \cup \{S_r\}, \{L, R\}, P_s \cup \{S_r \to S_s \mid \lambda\} \cup P'_s, S_r)$ where P'_s is defined by the rule:
$(\forall A \in N_s)\ (\forall a \in \{L, R, \lambda\})\ A \to a \in P_s \Rightarrow A \to aS_r \in P'_s$,
generates the reversal language $im_{\{L,R\}}(r)$. □

Now, we present a result related to the reversal complexity of linear languages.

THEOREM 3. $\mathcal{LIN} \subseteq REVERSAL_{\mathcal{GNF}}(n)$.

Proof. The proof is easily deduced from Theorem 2. Observe that the reversal language of a linear expression is regular. So, any reversal word can be tested in linear time and the number of changes is linear too. Hence, the function $f(n) = n$. □

Obviously, all linear languages share the upper bound stated in the previous theorem. Anyway, the regular grammars have reversal complexity that equals 0. That is, there is no direction changes during a derivation process in any regular grammar. It is confirmed by the linear expressions associated to regular grammars. The reversal language that can be obtained for the regular case is a subset of L^* or R^* (if we use right or left linear grammars, respectively).

4.1. A SPEED-UP THEOREM FOR REVERSAL COMPLEXITY

Our goal is to reduce the number of derivation changes in linear grammars. For every linear grammar in the normal form, we can obtain an equivalent one such that the number of reversals is reduced by a constant factor. In this case, we will use linear expressions to analyse the reversal complexity decrease. We will use the equivalence properties that we have described in Section 3.

First, we will define the reversal complexity of a linear expression as the number of index changes that, at most, occurs in any word of the expression. So, $REVERSAL_{\mathcal{LEXP}}(f)$ will denote the class of languages denoted by linear expressions with reversal complexity $f(n)$.

THEOREM 4. *Let L be a linear language. Then for every constant c such that $0 < c \leqslant 1$, $L \in REVERSAL_{\mathcal{LEXP}}(c \cdot n)$.*

Proof. We can easily construct a linear expression with no linear changes, in the case that L be finite. So, let L be infinite and let r be a linear expression for

L. We can obtain an equivalent expression $\varphi(r)$ for L. Let us observe that in any linear expression t, we can apply the equivalence property $t^* \equiv_{REG} (\lambda + t + tt + \cdots + t^{k-1}(t^k)^*$. So let us take any subexpression in $\varphi(r)$ that is affected by the closure operator (i.e., s) and let us apply the previous equivalence property with $k = \lceil 1/c \rceil$. Given that $s \equiv_{LIN} (\lambda + s + ss + \cdots + s^{k-1})(s^k)^*$, then it is equivalent to $(\varphi(\lambda) + \varphi(s) + \cdots + \varphi(s^{k-1}))\varphi((s^k)^*)$. Now let us fix our attention to $\varphi((s^k)^*)$. We can apply the following equivalence property $\varphi((s^k)^*) \equiv_{LIN} (\varphi(s^k))^*$. It is easy to prove that in $(\varphi(s^k))^*$ only a linear change is needed every k symbols. Consequently, the reversal complexity is reduced k times (i.e., it is augmented c times). So, if we apply the latter equivalence transformation to all the subexpressions in $\varphi(r)$ affected by the closure operator, then we can reduce the number of changes needed for their strings. The rest of subexpressions that are not affected by closure operators can be transformed trivially, in order to reduce the number of reversals. □

EXAMPLE 7. Let $r = (a_L b_R)^*$. We can speed-up the previous expression within a factor of 2 by introducing the following equivalent expression:

$$(\lambda + a_L b_R)((aa)_L (bb)_R)^*.$$

We can speed-up r within a factor of 3 by introducing the equivalent expression:

$$(\lambda + a_L b_R + (aa)_L (bb)_R)((aaa)_L (bbb)_R)^*.$$

Obviously, the previous result can be enunciated for linear grammars (not in normal form) by using the equivalence properties between linear grammars and linear expressions. So, the reversal complexity of any linear language can be independently analysed by using linear grammars or linear expressions.

5. Kolmogorov Complexity of Linear Languages

Kolmogorov complexity (Li and Vitànyi, 1993) measures the amount of information needed to describe objects. Here, we are interested in Kolmogorov complexity of linear languages, that is the number of bits needed to describe them. We can define the Kolmogorov complexity of a linear language as the Kolmogorov complexity of a linear grammar that generates the language. Obviously, given that linear grammars of different size can generate the same language, the Kolmogorov complexity of a linear grammar gives an upper bound of the Kolmogorov complexity of the language that it generates.

Another interest in measuring the Kolmogorov complexity of a linear language is that it can be considered as an infinite string. So, the language can be viewed as an infinite string with some repeated symbols (i.e., the separators). Then, if $L = \{x_1, x_2, \cdots\}$, it can be viewed as the string $< L >= x_1 \# x_2 \# \cdots$. Under this point of view, linear languages are non random given that they can be compressed by describing one of its linear expressions.

First, we give some formal definitions in order to fix the notation that we will use. Then, an upper bound for the Kolmogorov complexity of any linear language will be obtained.

Given any program p, we will denote by $|p|$ its length and we will denote by $out(p)$ its output.

DEFINITION 7. Let x be a sequence over Σ, then its Kolmogorov complexity is defined as $\mathcal{K}(x) = min\{|p| : out(p) = x\}$. That is the size of the minimal program that writes x.

Now, we can give a definition in order to work with linear languages.

DEFINITION 8. Let L be a linear language, then its Kolmogorov complexity, by using linear expressions, is defined as $\mathcal{K}_{lexp}(L) = min\{|r| : im_\Sigma(r) = L\}$. That is the size of the minimal linear expression r such that r denotes L.[2]

Then, the Kolmogorov complexity of any linear language will be upper-bounded by the Kolmogorov complexity of a linear expression that generates it. So, we will denote by $\mathcal{K}(r)$ the complexity of describing a linear expression that can be considered as a string over a predefined alphabet (that contains single symbols, operators, indexes and parentheses).

In any linear expression we must specify: (1) a set of strings according to an alphabet Σ (the *generators*), (2) a set of indexes to define the linear changes (L and R), and (3) a sequence of closures, products and unions that appear in the expression. So, any linear expression r can be defined by $\eta_m(x_{1y_1}x_{2y_2}\cdots x_{my_m})$, where $x_1, \cdots, x_m$ are the generators, $y = y_1y_2\cdots y_m \in \{L, R\}^*$ are the indexes for linear changes and η gives the order in which every operator appears in the expression. We can give the following upper bound for $\mathcal{K}_{lexp}(L)$:

$$\begin{aligned}\mathcal{K}_{lexp}(L) \leqslant\ & \mathcal{K}(r) \leqslant \mathcal{K}(x_1) + \cdots + \mathcal{K}(x_m) + \mathcal{K}(y) \\ & + \mathcal{K}(\eta_m(x_{1y_1}\cdots x_{my_m})) + \mathcal{O}(1),\end{aligned}$$

with $im_\Sigma(r) = L$ and $r = \eta_m(x_{1y_1}\cdots x_{my_m})$. Now, we can deduce a new expression for describing η_m. Here, the important data is the place where every operator appears and the operators themselves. So, the upper bound for describing η_m can be the following one:

$$\mathcal{K}(\eta_m(x_{1y_1}\cdots x_{my_m})) \leqslant log(|x_1|) + \cdots + log(|x_m|) + \mathcal{O}(1).$$

We can apply some equivalence properties in order to obtain a new upper bound as follows.

THEOREM 5. *Let L be a linear language over Σ and r be a linear expression such that $L = im_\Sigma(r)$. Then $\mathcal{K}_{lexp}(L) \leqslant \mathcal{K}(\varphi(r)) \leqslant \mathcal{K}(r)$.*

Proof. Obviously, the first inequality holds given that we can describe L by describing $\varphi(r)$. The second inequality can be deduced by the following facts:

1. The number of subscripts L and R in $\varphi(r)$ is less than or equal to the number of subscripts in r.
2. The number of product operators in r is greater than (or equal to) the number of product operators in $\varphi(r)$. So, the term η can be reduced. □

Finally, we can relate reversal and Kolmogorov complexity. Here, our result states that whenever the reversal complexity is decreased then the Kolmogorov complexity is augmented.

THEOREM 6. *Let L be a linear language over Σ and r be a linear expression such that $im_\Sigma(r) = L$. Let c be a constant value such that $0 < c \leqslant 1$ and r_c be a linear expression for L such that the reversal complexity of L according to r is decreased $\lceil 1/c \rceil$ times. Then $\mathcal{K}(r) \leqslant \mathcal{K}(r_c)$.*

Proof. Let r_c be a linear expression for L obtained from r by applying Theorem 4. Obviously, we can observe that r_c has more generators than r. In addition, r_c has more operators (i.e., unions and concatenations) than r. So, it can be deduced that we spend more bits in describing r_c than in describing r. □

Notes

[1] Given that r and s are linear expressions, they are regular expressions over $\Sigma_{\{L,R\}}$. Here $L(r)$ and $L(s)$ are regular languages and x and y are strings over $\Sigma_{\{L,R\}}$.

[2] Observe that the size of any linear expression can be defined as its length by taking into account the number of indexed symbols, the number of operators and the number of parenthesis symbols.

References

Chen, J. and C. Yap. Reversal complexity, *SIAM Journal on Computing* 20(4): 622–638, 1991.

Gruska, J. A characterization of context-free languages, *Journal of Computer and System Sciences* 5: 353–364, 1971.

Hashiguchi, K. The infinite 2-star height hierarchy of extended regular languages of star degree at most two, *Information and Computation* 114: 237–246, 1994.

Hashiguchi, K. and H. Yoo. Extended regular expressions of star degree at most two, *Theoretical Computer Science* 76: 272–284, 1990.

Li, M. and P. Vitànyi. *An Introduction to Kolmogorov Complexity and Its Applications*. Springer, Berlin, 1993.

Salomaa A. *Formal Languages*. Academic Press, New York, 1973.

Sempere, J.M. On a class of regular-like expressions for linear languages, *Journal of Automata, Languages and Combinatorics* 5(3): 343–354, 2000.

Wagner, K. and G.Wechsung. *Computational Complexity*. Reidel, Dordrecht, 1986.

Yntema, M.K. Cap expressions for context-free languages, *Information and Control* 18: 311–318, 1971.

Yoo, H. and K. Hashiguchi. Extended automata-like regular expressions of star degree at most (2,1), *Theoretical Computer Science* 88: 351–363, 1991.