An efficient algorithm for order evaluation of Strict Locally Testable languages

Antonio Magnaghi, Hidehiko Tanaka
The University of Tokyo, Japan
E-mail: {magnaghi, tanaka}@mtl.t.u-tokyo.ac.jp

1 *Abstract*

Strict k-local testability is an important concept in fields like pattern recognition, neural networks and formal languages theory. Words of a strict k-locally testable language L are parsed by decomposing the input in k-length sub strings without the need to consider context-dependent phenomena. First, we study the problem to decide if a language L is strict locally testable: an algorithm is presented to ascertain whether a value of k exists such that L is k-locally testable in a strict sense. Then we face the problem to determine the order of language L, e.g. the minimum value of parameter k so that string recognition can be optimally performed. Our approach relies on the development of the concept of a prefix path intersection graph. Through it, we can provide topological characterizations of strict local testability properties that can efficiently be tested in polynomial time. Moreover, the methods proposed in this paper distinguish from previously achieved results because we do not utilize algebraic concepts; in the past, strict local testability was studied in terms of the syntactic monoid structure.

2 Introduction and motivations

The concept of local testability (LT) has been broadly investigated in previous decades, yet it still represents an active area of research in the field of formal languages.

In such a context it is possible to identify two main research threads. One of them is concerned with linear sequences of symbols, e.g., string languages. The other analyzes more articulated structures, such as, for instance, images and tree languages [11, 15, 16]. In the case of strings, the wider class of Aperiodic Languages constitutes the formal framework for LT [4, 12]. Aperiodicity is revealed to be a linguistic universal, characterized in a variety of ways: grammatical inference [6], neural networks [12] and algebraic structures [4, 5, 12, 17]. The importance of LT springs from its close link to aperiodicity. A hierarchy of aperiodic languages was identified [4], imposing different constrains on the string recognition process. The hierarchy is composed of Definite, Reverse Definite, Locally Testable in a Strict Sense, Locally Testable and properly Aperiodic or Non-Counting languages at the top of the taxonomy.

The class of Locally Testable languages in a Strict Sense ($LTs.s.$) plays a crucial role in the whole Aperiodic hierarchy: Aperiodic languages are the closure of $LTs.s.$ w.r.t. boolean operators and concatenation [12]. Computational constraints imposed on $LTs.s.$ are actually met in several communication processes: this gives an intuitive valence to formal considerations. For a k-Locally Testable language in a Strict Sense ($L \in LT_k s.s.$) the recognition procedure is carried out on an input string x by a k-wide window to be moved along x. The sequences of symbols observed through the window are annotated in a record, regardless of the order or position they occupy in the string. After moving the window from one end to the other, x is accepted or rejected based on the set of sub strings that compose the produced record.

Also, locality property shows a link to parallel parsing of string languages. A word of a local language has such a syntactical structure that each sub string is analyzed independently from all the others. Hence, it is possible to decompose the input sentence among computational units of a parallel computer to simultaneously recognize the different parts, substantially improving parsing performance. Moreover, another feature emerges for $LTs.s.$ languages in relation to error identification. The presence of a syntax error is easy to detect and its position is precisely defined as well: it is located within the k-length sub string that does not match any element of the recognition sets of words used when parsing. On the contrary, in the general case when $LTs.s.$ property does not hold, error handling is more complex.

A systematic characterization of the different sub families of aperiodic languages was presented in the past [3, 5]. The adopted techniques, however, were quite elaborate and the algebraic approach left unsolved computational problems. In 1994 [10] it was proved that the optimization problem of the order of an LT language is NP-hard. In the following section, we describe the results we could achieve for $LTs.s.$ class.

3 Our results

In this paper, we provide an efficient algorithm for order evaluation of languages belonging to $LTs.s.$ To our knowledge, no such algorithm has been reported in the literature. Presented considerations describe how to prove that the order evaluation of a language L in $LTs.s.$ can be performed by a polynomial algorithm of time-complexity $o(|\Sigma|^2mn)$, where Σ, m and n are, respectively, the alphabet of L, the number of edges and the number of states of the finite state automaton accepting L. In order to obtain such a result, it was necessary to frame the concept of $LTs.s.$ in a different perspective. Our approach aims at capturing strict local testability in a direct manner, without employing any algebraic property of the syntactic monoid. Instead, a set-theoretically based analysis is carried out in order to link local testability to topological properties of the automaton of language L. Before being able to establish the polynomial algorithm of order evaluation, it was necessary to face strictly related problems, concerning the characterization of k-local testability in a strict sense ($LT_ks.s.$), and the decidibility of $LTs.s.$ property. The exposition follows the sequentiality of these conceptual units. In particular, the specific points we addressed can be summarized as follows:

1. *Characterization of $LT_ks.s.$ property*: for a specific integer value of k, the analyzed decision problem is: "$L \in LT_ks.s.$?"

   A sufficient and necessary condition is formulated (theorem 3). It involves topological properties of paths in the accepting automaton. Such a characterization has the advantage to impose determinism as a unique constraint, without requiring the automaton to be reduced. Nonetheless the minimal automaton case is studied and then conveniently employed in subsequent considerations.

2. *Development of an algorithm to decide $LTs.s.$ property (existential problem)*: given language L, does a value of k exist such that $L \in LT_ks.s.$?

   Our approach consists, first, in defining the Prefix-Path-Intersection Graph (PPIG). For its construction a fixed-point algorithm is formulated. Then its complexity is shown to be $o(|\Sigma|^2mn)$ (theorem 5).

3. *Development of an algorithm for the optimization problem of order evaluation*: for a language $L \in LTs.s.$, which is the minimum value of k_{min} such that $k_{min} = \min_k\{k : L \in LT_ks.s.\}$?

   The study of the PPIG properties relates the length of the longest path in the PPIG to the order of language L. Then, finally, the paper states the major result: the order of L can be evaluated in $o(|\Sigma|^2mn)$.

In addition to previous results, the introduced approach seems to have a worthwhile characteristic: the syntactic monoid and its algebraic structure are not involved. This leads us to think that such an approach might give insight on how to extend our considerations to different contexts, such as image and tree languages.

4 Preliminary definitions

Let Σ be a finite alphabet of symbols, and let Σ^* denote the universal language over Σ, including all the strings obtained by concatenation of alphabet elements. A subset L of Σ^* is a string language, or a string event, over Σ. If L defines a regular set (it

can be characterized through a regular expression), the language L is regular and it can be recognized by a finite state automaton M.
Our notation follows [12]. Being that k is a non-negative integer number, it is possible to define the following operators on a string x of length greater or equal to k:

$$L_k(x) = \{y : x = yw \wedge |y| = k\} \quad (1)$$

$$R_k(x) = \{w : x = yw \wedge |w| = k\} \quad (2)$$

$$I_k(x) = \{w : x = ywz \wedge y, w, z \neq \epsilon \wedge |w| = k\} \quad (3)$$

The operator $L_k(x)$ extracts the k-length prefix from the input string. Symmetrically, $R_k(x)$ produces the k-length suffix of word x. Equation (3) defines the set of properly internal k-length sub strings of x. If the length of x (denoted by $|x|$) equals k or (k+1), $I_k(x)$ is the empty set.
Let α_k,β_k,γ_k be sub sets of Σ^k; they are sets of strings over Σ whose length is k.
The language L is k-locally testable in a strict sense ($L \in LT_k s.s.$) if sets α_k,β_k,γ_k exist such that for every $x \in \Sigma^*$ ($|x| \geq k$):

$$\begin{aligned}&(x \in L) \iff \\ &(L_k(x) \in \alpha_k \wedge I_k(x) \subseteq \beta_k \wedge R_k(x) \in \gamma_k)\end{aligned} \quad (4)$$

Based on relation (4), a k-locally testable language in a strict sense has a property so that syntactic analysis can be performed locally. On a procedural level, parsing activity requires that the prefix ($L_k(x)$), suffix ($R_k(x)$) and the set on internal sub strings ($I_k(x)$) be extracted from string x. Recalling the initial window analogy, x can be parsed by a k-letters-wide loophole to be moved from left to right end one symbol at a time.
Correctness is evaluated using only the information collected through such a decomposition. No information about order or relative position of occurrence is kept. Definition (4) does not consider strings of L consisting of a number of symbols less than k. In this case, the number of possible words is limited, so parsing can be performed separately in a simple way.
In particular $\alpha_k, \beta_k, \gamma_k$ contain the recognition patterns necessary to ascertain whether a string belongs to L or not. α_k can be interpreted as the set containing all possible k-length prefixes of strings of L. Dually, γ_k is the set of all acceptable k-length suffixes. β_k is the set of all acceptable internal k-length sub strings of words of L.

$$\begin{aligned}\alpha_k &= \{x : w = xy \wedge |x| = k \wedge w \in L\} &(5)\\ \beta_k &= \{v : w = uvz \wedge \\ &\quad \wedge u, z \neq \epsilon \wedge |v| = k \wedge w \in L\} &(6)\\ \gamma_k &= \{y : w = xy \wedge |y| = k \wedge w \in L\} &(7)\end{aligned}$$

$L \in LT_k s.s.$ means that syntactical analysis can be correctly carried out through sets $\alpha_k, \beta_k, \gamma_k$, as defined above. On the contrary, if $L \notin LT_k s.s.$, the language recognized through such sets is a super set of L.

5 Basic concepts and formal tools

Let $M = (Q, \Sigma, \delta, q_0, F)$ [8] be a deterministic finite state automaton (DFA) accepting the regular language L. Q is the set of states. Σ is the input alphabet. δ is the transition function. $q_0 \in Q$ is the initial state and $F \subseteq Q$ is the non-empty set of final states. For any $q \in Q$ and $x \in \Sigma^*$, $\delta(q, x)$ denotes the state that results when input x is applied to M from state q. A string x is accepted by M if $\delta(q_0, x) \in F$, hence $L = \{x \in \Sigma^* : \delta(q_0, x) \in F\}$.
Let us define $\hat{M} = (\hat{Q}, \Sigma, \hat{\delta}, q_0, \hat{F})$ to be the automaton derived from M in such a way that: $\hat{Q} \subseteq Q$; $\hat{\delta}$ is a new partial transition function whose domain is a subset of $\hat{Q} \times \Sigma$, and image is $\hat{Q}$; the new set of final states $\hat{F}$ is a sub set of F. We require that $\forall q \in \hat{Q}, \exists x_1, x_2 \in \Sigma^* : \delta(q_0, x_1) = q \wedge \delta(q, x_2) \in \hat{F}$. A node q is in the set $\hat{Q}$ of $\hat{M}$ only if q can be reached in M from the initial state q_0 and if from q, it is possible to reach in M a final state belonging to F. The transition function is modified accordingly: $\hat{\delta}(q_1, a) = q_2$ is defined in $\hat{M}$ iff $q_1, q_2 \in \hat{Q}$ and $\delta(q_1, a) = q_2$ in M. $\hat{M}$ differs from M for the suppression of unreachable (from q_0) or unproductive states that do not allow an end to the computation in a final node. Consequently transitions from/to such suppressed states are eliminated.
$M_r = (Q_r, \Sigma, \delta_r, F_r)$ denotes the reduced DFA associated with M. M_r is unique if state-renaming

isomorphisms are neglected. q is a sink state in M if, from it, a final node can not be reached. When a sink state is reached: $\delta(q_0, x) \notin F$, this means that input string x does not belong to L. In M_r the sink state, if present, is unique and it is denoted by S. $\hat{M_r}$ results from M_r by eliminating S and all transitions: $\delta(q, a) = S, a \in \Sigma, q \in Q$.

Let $\hat{G} = (V, \Sigma, P, < q_0 >)$ be the linear right-derivative context-free grammar associated univocally to $\hat{M}$ as follows:

1. the set V of non terminals contains a symbol for every state of $\hat{M}$:

$$V = \{< q >: q \in \hat{Q}\}$$

2. the terminal alphabet Σ of $\hat{G}$ equals the input alphabet of $\hat{M}$

3. the set P of linear productions is derived from $\hat{M}$ as follows:

$$\begin{aligned} P &= \{< q_1 > \to a < q_2 >: \\ & q_1, q_2 \in \hat{Q} \wedge a \in \Sigma \wedge \hat{\delta}(q_1, a) = q_2\} \cup \\ & \cup \{< q_f > \to \epsilon : q_f \in \hat{F}\} \end{aligned}$$

4. the grammar axiom $< q_0 >$ corresponds to the initial state q_0 of $\hat{M}$.

Let π be the omomorphism whose domain and image are respectively $(\Sigma \cup V)^*$, Σ^*:

$$\pi(a) = a, \forall a \in (\Sigma \cup \{\epsilon\})$$

$$\pi(< q_j >) = \epsilon, \forall q_j \in \hat{Q}$$

$$\pi(xy) = \pi(x)\pi(y), \forall x, y \in (\Sigma \cup V)^*$$

For any state q_i of $\hat{M}$ and for any integer k, the following set of strings is defined:

$$V_k(q_i) = \{\pi(\omega) :< q_i > \stackrel{k}{\Longrightarrow} \omega\} \tag{8}$$

$V_k(q_i)$ is the set of all words obtained through the application of $\pi(.)$ to derivations of length k starting from the non terminal $< q_i >$. If the last derivation to produce ω is not terminal, then $|\pi(\omega)| = k$; otherwise $|\pi(\omega)| = (k-1)$.

For every $q_i \in \hat{Q}$ and for every $x \in V_k(q_i)$, let us define the set $D_k(q_i, x)$:

$$\begin{aligned} D_k(q_i, x) &= \{y = xu \in V_{k+1}(q_i) : \\ & \quad u \in \Sigma \cup \{\epsilon\}\}, if|x| = k \qquad (9) \\ D_k(q_i, x) &= \emptyset, if|x| = (k-1) \qquad (10) \end{aligned}$$

$D_k(q_i, x)$ consists of the strings whose k-prefix equals x and that are produced by a chain of (k+1) derivations from $< q_i >$.

Lemma 1 *$x \in D_k(q_i, x)$ iff $\delta(q_i, x) \in F$*

6 Characterization of $LT_k s.s.$ property

In this section a set-theoretical characterization of $LT_k s.s.$ property is established (theorem 3). The adopted formulation of a necessary and sufficient condition directly appears to be of interest on a procedural level. Such a result constitutes a decidibility algorithm for ascertaining if L is in $LT_k s.s.$ We show in the following section that theorem 3 also represents a useful instrument for the solution to a different decision problem: "*Is L in LTs.s.?*," which contributes to an explication of the relation between $LT_k s.s.$ and $LTs.s.$

In the following considerations, q_i, q_h denote arbitrary states in $\hat{Q}$, if there is at least one edge in $\hat{M}$ entering the initial state q_0. Otherwise, q_i, q_h belong to $\hat{Q} - \{q_0\}$: in this case q_0 is used only once when string parsing begins, hence such a state is not considerated because it can not generate k-length recognition strings either in β_k or in γ_k.

Theorem 1 *Let L be in $LT_k s.s.$, then every DFA M accepting L is such that:*

$$D_{k-1}(q_i, x) = D_{k-1}(q_h, x)$$

for any $x \in V_{k-1}(q_i) \cap V_{k-1}(q_h)$ and for any q_i, q_h.

Proof

It will be proved that $L \notin LT_k s.s.$ if there exist two distinct states q_i, q_h and a string x such that:

$$x \in V_{k-1}(q_i) \cap V_{k-1}(q_h) \tag{11}$$

$$D_{k-1}(q_i, x) \neq D_{k-1}(q_h, x) \tag{12}$$

If $|x| = (k-2)$, then $D_{k-1}(q_i, x) = D_{k-1}(q_h, x) = \emptyset$ because of definition (10). Hence, necessarily $|x| = k-1$.

Let x have the form: $x = t_1 t_2 \ldots t_{k-1}, t_i \in \Sigma, 1 \leq i \leq k-1$. Condition (12) implies that $q_i \neq q_h$ and that at least one of the sets $(D_{k-1}(q_i, x) - D_{k-1}(q_h, x)), (D_{k-1}(q_h, x) - D_{k-1}(q_i, x))$ is not empty. For instance, let y belong to $D_{k-1}(q_i, x) - D_{k-1}(q_h, x) : y = x t_k (|y| = k)$. As $y \notin D_{k-1}(q_h, x)$, $\delta(q_h, y) = P$, where P is a sink state. Being that M is deterministic, and $q_i \neq q_h$, there must be two different strings w_1, w_2 that lead from q_0 to q_i and q_h respectively: $\hat{\delta}(q_0, w_1) = \hat{\delta}(q_0, a_1 a_2 \ldots a_m) = q_i; \hat{\delta}(q_0, w_2) = \hat{\delta}(q_0, b_1 b_2 \ldots b_n) = q_h$, where $m, n \geq 0$, but not both of them can equal zero, and $w_1 \neq w_2$. Let us consider the following states: $\tilde{q_i} = \hat{\delta}(q_i, x); \tilde{q_h} = \hat{\delta}(q_h, x); \tilde{q_i}' = \hat{\delta}(\tilde{q_i}, t_k), P = \delta(\tilde{q_h}, t_k)$. From $\tilde{q_i}'$ a final state q_f is reachable. Let $z = c_1 c_2 \ldots c_s, (s \geq 0)$ be the string leading to q_f: $\delta(\tilde{q_i}, z) = q_f \in F$. Now, let us consider the word $w = w_2 y z$, where the length of w is greater or equal to k. Previous considerations assure that $\delta(q_0, w_2 y) = P$. This implies $\delta(q_0, w_2 y z) = \delta(q_0, w) = P$, hence $w \notin L$.

In the reminder of the proof it will be verified that the existence of string w implies no proper sets $\alpha_k, \beta_k, \delta_k$ exist. If L were in $LT_k s.s.$, necessarily syntactical analysis should utilize $\alpha_k, \beta_k, \gamma_k$ as defined in (5), (6), (7). Nonetheless, in such a case a super set of L would be recognized.

Let us consider $L_k(w)$; two cases are possible according to the length of string w_2: (a)if $n \geq k$, $L_k(w) = b_1 b_2 \ldots b_k$, (b)if $0 \leq n < k$, $L_k(w) = b_1 b_2 \ldots b_n t_1 t_2 \ldots t_l$, where $n + l = k, 1 \leq l \leq k$.

(Case a) $\hat{\delta}(q_0, w_2) = q_h$, and from q_h a final state $q_{f'}$ is reachable, hence from $\hat{\delta}(q_0, b_1, b_2 \ldots b_k)$ the same node $q_{f'}$ is reachable. As a consequence, $b_1 b_2 \ldots b_k$ is the prefix of a string in L, hence because of (5) $L_k(w) \in \alpha_k$.

(Case b)The condition $x = t_1 t_2 \ldots t_k \in V_{k-1}(q_h)$ guarantees that for any state: $q_l = \hat{\delta}(q_0, w_2 t_1 t_2 \ldots t_l), 1 \leq l \leq k$, a path exists leading to a final state of $\hat{M}$ from q_l. Hence, it is possible to conclude again that $L_k(w) \in \alpha_k$.

Let us consider $R_k(w)$ and recall that $|z| = s$. (a)if $s \geq k$, $R_k(w) = c_{s+1-k} c_{s+2-k} \ldots c_s$; (b)if $0 \leq s < k$, $R_k(w) = t_l t_{l+1} \ldots t_k c_1 c_2 \ldots c_s, 1 \leq l \leq k$. In both case (a) and (b) we can prove that $R_k(w) \in \gamma_k$ in a fashion similar to the one used for $L_k(w)$.

In the end it is also possible to verify that $I_k(w) \subseteq \beta_k$.

Hence if conditions (11), (12) simultaneously hold, a string w exists such that $w \notin L$, but for which: $L_k(w) \in \alpha_k, I_k(w) \subseteq \beta_k, R_k(w) \in \gamma_k$. Thus we conclude $L \notin LT_k s.s.$

■

In order to prove theorem 2 in a more concise fashion, two preliminary lemmata are required (lemma 2 and lemma 3).

Lemma 2 *Let M be a DFA such that:*

$$D_{k-1}(q_i, x) = D_{k-1}(q_h, x)$$

for any $x \in V_{k-1}(q_i) \cap V_{k-1}(q_h)$, and for any q_i, q_h. Let $w = a_1 a_2 \ldots a_m, (m \geq k)$ be a string such that: $L_k(w) \in \alpha_k \wedge I_k(w) \subseteq \beta_k \wedge R_k(w) \in \gamma_k$. Then

$$\delta(q_0, a_1 a_2 \ldots a_r) \in \hat{Q}, 1 \leq r \leq m$$

As follows, the second lemma is stated, in order to prove subsequent theorem 2.

Lemma 3 *Let M be a DFA such that:*

$$D_{k-1}(q_i, x) = D_{k-1}(q_h, x)$$

for any $x \in V_{k-1}(q_i) \cap V_{k-1}(q_h)$ and for any q_i, q_h. Then the state $\delta(q_i, x)$ is equivalent to $\delta(q_h, x)$.

Proof

Let $x = a_1 a_2 \ldots a_k \in V_{k-1}(q_i) \cap V_{k-1}(q_h)$, and let us assume that $\tilde{q_i} = \delta(q_i, x)$ is not equivalent to $\tilde{q_h} = \delta(q_h, x)$. This implies the existence of a string $y = b_1 b_2 \ldots b_m (m \geq 0)$ such that $\delta(\tilde{q_i}, y) \in F$ and $\delta(\tilde{q_h}$

$, y) \notin F$ or $\delta(\widetilde{q_i}, y) \notin F$ and $\delta(\widetilde{q_h}, y) \in F$.
Consider for instance the first possible case, let $z = xy(|z| \geq k)$, and σ_i (resp. σ_h) be the path comprising the edges of M used by the involved transitions from q_i (resp.q_h) to $\delta(q_i, z)$ (resp. $\delta(q_h, z)$):

$$\sigma_i = (q_i, \delta(q_i, a_1))(\delta(q_i, a_1), \delta(\delta(q_i, a_1), a_2)) \ldots (\delta(q_i, a_1 a_2 \ldots a_k b_1 b_2 \ldots b_{m-1}), b_m)$$

$$(resp. \sigma_h = (q_h, \delta(q_h, a_1))(\delta(q_h, a_1), \delta(\delta(q_h, a_1), a_2)) \ldots (\delta(q_h, a_1 a_2 \ldots a_k b_1 b_2 \ldots b_{m-1}), b_m))$$

With $\widetilde{q_i\prime}$ (resp. $\widetilde{q_h\prime}$) we designate the node at a distance of (k-1) edges from $\delta(q_i, z)$ (resp. $\delta(q_h, z)$) along the path σ_i (resp. σ_h). As $|z| \geq k$, $\widetilde{q_i\prime}$ (resp. $\widetilde{q_h\prime}$) exists, and $\widetilde{x}$ is the (k-1)-suffix of z such that: $\delta(\widetilde{q_i\prime}, \widetilde{x}) = \delta(\widetilde{q_i}, y)$ (resp. $\delta(\widetilde{q_h\prime}, \widetilde{x}) = \delta(\widetilde{q_h}, y)$). We note that $\widetilde{x} \in V_{k-1}(\widetilde{q_i\prime}) \cap V_{k-1}(\widetilde{q_h\prime})$. However, the set equality $D_{k-1}(\widetilde{q_i\prime}, \widetilde{x}) = D_{k-1}(\widetilde{q_h\prime}, \widetilde{x})$ does not hold: $\delta(\widetilde{q_i\prime}, \widetilde{x}) = \delta(\widetilde{q_i}, y) \in F$, hence lemma 1 guarantees that $\widetilde{x} \in D_{k-1}(\widetilde{q_i\prime}, \widetilde{x})$; on the other hand, being $\delta(\widetilde{q_h\prime}, \widetilde{x}) = \delta(\widetilde{q_h}, y) \notin F, \widetilde{x} \notin D_{k-1}(\widetilde{q_h\prime}, \widetilde{x})$ (lemma 1). This represents a contradiction; necessarily $\delta(q_i, x)$ is equivalent to $\delta(q_h, x)$.
■

We can derive the following result as an immediate consequence of previous lemma:

Corollary 1 *Let M_r be a reduced DFA such that:*

$$D_{k-1}(q_i, x) = D_{k-1}(q_h, x)$$

for any $x \in V_{k-1}(q_i) \cap V_{k-1}(q_h)$ and for any q_i, q_h. Then:

$$\delta(q_i, x) = \delta(q_h, x)$$

It is now possible to proceed to theorem 2; it proves the validity of exchanging hypothesis and thesis in theorem 1.

Theorem 2 *Let M be a DFA such that:*

$$D_{k-1}(q_i, x) = D_{k-1}(q_h, x)$$

for any $x \in V_{k-1}(q_i) \cap V_{k-1}(q_h)$ and for any q_i, q_h. Then the language accepted by M is $LT_k s.s.$ w.r.t. α_k (5), β_k (6), γ_k (7).

Proof
Recalling the definition of $LT_k s.s.$ language (4), two implications must be verified.
Being $w \in \Sigma^*, |w| \geq k$:

$$w \in L \Rightarrow L_k(w) \in \alpha_k \wedge I_k(w) \subseteq \beta_k \wedge R_k(w) \in \gamma_k$$

and

$$L_k(w) \in \alpha_k \wedge I_k(w) \subseteq \beta_k \wedge R_k(w) \in \gamma_k \Rightarrow w \in L$$

Because of (5), (6), (7) the first of them holds in a straightforward manner, whereas the second one requires additional considerations.

Let w be a string of this form: $w = a_1 a_2 \ldots a_m (m \geq k)$, with the property that $L_k(w) \in \alpha_k, I_k(w) \subseteq \beta_k, R_k(w) \in \gamma_k$. Our aim is to show that w is syntactically correct: $\delta(q_0, w) = q_m \in F$.
$R_k(w) = a_{m-k+1} a_{m-k+2} \ldots a_m \in \gamma_k$. Let us consider the states: $q_{m-k+1} = \delta(q_0, a_1 a_2 \ldots a_{m-k+1})$ and $q_m = \delta(q_{m-k+1}, a_{m-k+2} a_{m-k+3} \ldots a_m)$. Both of them belong to $\hat{Q}$ because of lemma 2. The string $a_{m-k+1} a_{m-k+2} \ldots a_m$ is in γ_k, therefore a string y of L exists such that $a_{m-k+1} a_{m-k+2} \ldots a_m$ is its k-length suffix. Being y in L, a state $\widetilde{q}$ exists: $\delta(\widetilde{q}, a_{m-k+2} a_{m-k+3} \ldots a_m) = q_f \in F$. Lemma 2 assures that $\widetilde{q} \in \hat{Q}$. In particular: $a_{m-k+2} a_{m-k+3} \ldots a_m \in V_{k-1}(q_{m-k+1}) \cap V_{k-1}(\widetilde{q})$. Considering that the hypothesis of lemma 3 hold, we conclude the state $\delta(q_{m-k+1}, a_{m-k+2} a_{m-k+3} \ldots a_m)$ is equivalent to $\delta(\widetilde{q}, a_{m-k+2} a_{m-k+3} \ldots a_m)$. Therefore, q_m is equivalent to q_f: q_m belongs to F, that is $w \in L$.
■

Theorems 1 and 2 lead us to obtain directly the main result of this section, characterizing $LT_k s.s.$

Theorem 3 *A language L, accepted by a DFA M, is in $LT_k s.s.$ iff $D_{k-1}(q_i, x) = D_{k-1}(q_h, x)$ for any $x \in V_{k-1}(q_i) \cap V_{k-1}(q_h)$ and for any q_i, q_h.*

The following corollary is a direct consequence of theorem 3 and corollary 1:

Corollary 2 *A language L, accepted by the reduced DFA M_r, is in $LT_k s.s.$ iff $\delta(q_i, x) = \delta(q_h, x)$ for any $x \in V_{k-1}(q_i) \cap V_{k-1}(q_h)$ and for any q_i, q_h.*

7 $LTs.s.$ decidibility algorithm

In this section, we show that local testability in a strict sense can be checked through an acyclicity test on a convenient graph (PPIG), directly obtained from the automaton that accepts L. The overall complexity of the decision algorithm results in $o(|\Sigma|^2 mn)$, where m, n are the cardinality of the sets of edges and nodes of $\hat{M}_r$.

The considerations below are restricted to the reduced DFA $M_r = (Q, \Sigma, \delta, q_0, F)$ that recognizes L. Corollary 2 to theorem 3 provides a necessary and sufficient condition for local testability that can significantly be expressed in terms of topological properties of paths on M_r. Let us consider two states q_i and q_h in $\hat{Q}$, from which it is possible to produce two paths, σ_i, σ_h respectively, not containing the sink state S, and labeled through the same k-length string x. M_r (the language L) is in $LT_k s.s.$ if, and only if, there exists a prefix u of x (not necessarily proper) leading to the same node in σ_i and σ_h: $\delta(q_i, u) = \delta(q_h, u) = q_c$. If the prefix u equals x, the condition expressed in corollary 2 is valid in a straightforward way. Otherwise, if u is a proper prefix, the determinism of M_r guarantees that from q_c the paths σ_i and σ_h necessarily coincide, hence $\delta(q_i, x) = \delta(q_h, x)$. Therefore, L is in $LT_k s.s.$ iff for any $q_i, q_h \in \hat{Q}$ and any arbitrary k-length string $x \in V_{k-1}(q_i) \cap V_{k-1}(q_h)$, paths σ_i, σ_h present an intersection node, reached through the same prefix string.

For every letter a of the input alphabet Σ, a set I_a is constructed composed of all nodes of M_r that are the target of an arc labeled with letter a:

$$I_a = \{q_i : \exists q_h \in \hat{Q} \wedge \hat{\delta}(q_h, a) = q_i\}, a \in \Sigma \tag{13}$$

Let N be the set containing all sets I_a, whose cardinality is greater than one:

$$N = \{I_a : a \in \Sigma \wedge |I_a| > 1\} \tag{14}$$

A function $\hat{\Delta}$ is defined on $N \times \Sigma$:

$$\hat{\Delta}(I_{a_1}, a_2) = \bigcup_{\substack{q_i \in I_{a_1} \\ \hat{\delta}(q_i, a_2) defined}} \{\hat{\delta}(q_i, a_2)\} \tag{15}$$

$\hat{\Delta}$ is such that: $\hat{\Delta}(I_{a_1}, a_2) \subseteq I_{a_2}, \forall a_1, a_2 \in \Sigma$.

Let us now introduce the Prefix-Path-Intersection Graph (PPIG). It is a graph produced by the fixed-point algorithm of Figure 1.

```
Input: set of nodes N ; alphabet Σ ; function Δ^
Output: PPIG=( N_PPIG , E_PPIG )
{SetType: N_PPIG , E_PPIG ,NewStates;
N_PPIG = N ; E_PPIG =∅; NewStates= N_PPIG ;
while(NewStates != ∅)
   {chose I_a1 ∈NewStates
   for every a_2 ∈ Σ
      {if(( Δ^( I_a1 , a_2 )∉ N_PPIG )&&(| Δ^( I_a1 , a_2 )|>1))
         {NewStates=NewStates ∪ { Δ^( I_a1 , a_2 )};
         N_PPIG = N_PPIG ∪{ Δ^( I_a1 , a_2 )};
         };
      if(( I_a1 , Δ^( I_a1 , a_2 ), a_2 )∉ E_PPIG )
         E_PPIG = E_PPIG ∪{( I_a1 , Δ^( I_a1 , a_2 ), a_2 )};
      };
   NewStates=NewStates - { I_a1 };
   };
return PPIG=( N_PPIG , E_PPIG );
}
```

Figure 1: PPIG Construction

The algorithm considers initially all macro-nodes defined in (13). From each node, all possible output arcs are taken into account. If function $\hat{\Delta}$ maps the node I_{a_1} to a new macro-node containing a number of M_r states greater than one, the set N_{PPIG} is consequently augmented.

The algorithm terminates when variable $NewStates$ is empty. It contains all nodes from which new possible transitions may originate. When $NewStates$ is empty, there is no possibility to further augment the graph: a fixed-point therefore is reached. The cardinality of $NewStates$ is limited by the power set of N and every iteration of the out-most for-loop reduces it by one element. Therefore $NewStates$ will be empty, hence the algorithm terminates.

Lemma 4 *All macro-nodes in the same loop of the*

PPIG contain the same number of nodes of M_r.

Lemma 4 assures that all the macro-nodes of the PPIG in any strongly-connected component contain the same number of states of M_r.
Finally we can prove the following theorem that provides an algorithm to test $LTs.s.$ property.

Theorem 4 *L is in $LTs.s.$ iff its PPIG is acyclic.*

Proof
(Necessary condition)
If a cycle $\eta = < I_{a_1}, I_{a_2}, \ldots, I_{a_l} >$ exists in the PPIG, all nodes in it contain the same number of states of M_r (lemma 4). Moreover, all of the PPIG macro-nodes contain at least two distinct states of M_r. Let us consider $q_1, q_2 \in I_{a_l}$ ($q_1 \neq q_2$) and the string: $y = a_1 a_2 \ldots a_l$. $\delta(q_1, y) = q_1$, $\delta(q_2, y) = q_2$; this guarantees the existence of an arbitrary length string x such that $\delta(q_1, x) \neq \delta(q_2, x)$, implying L is not in $LTs.s.$ (corollary 2). Hence necessarily the PPIG is cycle-free.
(Sufficient condition)
Being that the PPIG is cycle-free, all paths in it are simple. Hence, it is possible to consider the longest path σ in the graph. All paths labeled by strings in Σ^* whose length is greater than the length of σ are completely disjointed or met in one same node of M_r. Therefore, the condition expressed by corollary 2 is verified, assuming k is equal to the length of σ augmented by one.
■

We can proceed to evaluate the complexity of the construction algorithm for the PPIG and an upper bound to the cardinality of sets N_{PPIG}, E_{PPIG}. In the reminder of the paper all considerations will be related, through the PPIG, to the $\hat{M}_r$ graph. Hence, values m, n refer to the cardinality of the set of edges and nodes in $\hat{M}_r$.

Theorem 5 *The PPIG=(N_{PPIG}, E_{PPIG}) construction algorithm has a time complexity $o(|\Sigma|^2 mn)$ and $|N_{PPIG}| \leq C_1 |\Sigma| m$, $|E_{PPIG}| \leq C_2 |\Sigma|^2 m$, where C_1, C_2 are constants.*

Proof
We will outline the basic ideas as follows. Let $|NewStates|$ denote the overall number of different states that are inserted in variable $NewStates$ during the whole execution of the algorithm. Any time control flow reaches the while-cycle last instruction one element is eliminated from $NewStates$; hence, the cycle will be iterated $|NewStates|$ times. If we focus on the body of the for-statement, it is possible to note that operations can be carried out in a time proportional to n under the worst-case assumption. Therefore, the complexity is $o(|\Sigma| n |NewStates|)$. Hence, an upper bound for $|NewStates|$ value is required. Let us consider one of the macro-nodes given by (13) and the path $\sigma(I_a)$ in the PPIG composed of the following nodes: $\sigma(I_a) = < I_a, \hat{\Delta}(I_a, a), \hat{\Delta}(\hat{\Delta}(I_a, a), a), \hat{\Delta}(\hat{\Delta}(\hat{\Delta}(I_a, a), a), a) \ldots >$. We know that $\hat{\Delta}(I_a, a) \subseteq I_a; \hat{\Delta}(\hat{\Delta}(I_a, a), a) \subseteq \hat{\Delta}(I_a, a) \subseteq I_a; \ldots$ This assures that the length of $\sigma(I_a)$ is limited, and it contains the maximum possible number of nodes when it is cycle-free and every successive application of function $\hat{\Delta}$ decreases by one unit the number of M_r states in the input macro-node. Hence, $\max |\sigma(I_a)| \leq |I_a|$. Now let us analyze at the same time two distinct macro-nodes, I_{a_1}, I_{a_2} as defined in (13). The construction of the PPIG requires us to consider the a_2-labeled edges from nodes of $\sigma(I_{a_1})$. Let s_1 denote a node in $\sigma(I_{a_1})$. $\hat{\Delta}(s_1, a_2)$ can be a node in $\sigma(I_{a_2})$; in this case, no new node is added. $\hat{\Delta}(s_1, a_2)$, however, can be a new node. A node $s_2 \in \sigma(I_{a_2})$ exists such that $s_2 \supseteq \hat{\Delta}(s_1, a_2)$. In particular, let s_2 be the smallest set in $\sigma(I_{a_2})$ with the property that $s_2 \supseteq \hat{\Delta}(s_1, a_2)$. Only the new node $\hat{\Delta}(s_1, a_2)$ is introduced, because $\hat{\Delta}(\hat{\Delta}(s_1, a_2), a_2) = \hat{\Delta}(s_2, a_2)$. Hence, for every ordered couple $(\sigma(I_{a_1}), \sigma(I_{a_2}))$, the number of new nodes that it is possible to introduce does not exceed $\max |\sigma(I_{a_1})| \leq |I_{a_1}|$.
Now, therefore, we can consider the following chain, where c_1 is a constant:

$$
\begin{aligned}
|NewStates| &\leq c_1 \sum_{a \in \Sigma} (|I_a| + (|\Sigma| - 1)|I_a|) = \\
&= c_1 |\Sigma| \sum_{a \in \Sigma} |I_a| = c_1 |\Sigma| m
\end{aligned}
$$

The last equality is generated by the fact that the sets I_a can be directly mapped to a partition in the edges set of M_r.

In order to conclude: $|NewStates| \leq c_1|\Sigma|m$, $|N_{PPIG}| \leq c_1|\Sigma|m$. It is possible also to evaluate an upper bound for the cardinality of the set of edges E_{PPIG}, considering the arcs belonging to every path $\sigma(I_{a_1})$ and the arcs connecting every node of N_{PPIG}. As the language automaton is deterministic, the maximum number of edges from every macro-node in the PPIG is $|\Sigma|$, hence we have:

$$\begin{aligned} |E_{PPIG}| &\leq c_2 \sum_{a\in\Sigma}(|I_a| + |\Sigma|(|\Sigma|-1)|I_a|) \leq \\ &\leq c_3|\Sigma|^2 \sum_{a\in\Sigma} |I_a| = c_3|\Sigma|^2 m \end{aligned}$$

where c_2, c_3 are constants.
■

Once the PPIG is constructed, the acyclicity test (theorem 4) can be performed through the algorithm of [14], whose complexity is $o(\max(|N_{PPIG}|, |E_{PPIG}|) = o(|\Sigma|^2 m)$.

8 Order evaluation

The order of a language $L \in LTs.s.$ is defined as the minimum value k_{min} of parameter k such that L is in $LT_{k_{min}}s.s.$ If L is in $LT_k s.s.$, clearly L belongs also to $LT_{k'}$, where $k' > k$. In general, however, it is not true that L is in $LT_{k''}s.s.$ with $k'' < k$. k_{min} determination allows us to optimally parse the input string. This section faces the problem so we can evaluate the order of L and, as a result, we propose an algorithm whose complexity is $o(|\Sigma|^2 mn)$.

Theorem 6 *Let L be a language whose PPIG is acyclic ($L \in LTs.s.$), and let k_{PPIG} denote the length of the longest path in the PPIG. Then the order of L is:*

$$k_{min} = k_{PPIG} + 2$$

Proof
For a macro-node I_a of the PPIG, let us consider a path $\sigma =< I_a, I_{a1}, I_{a_2} \ldots, I_{a_l} >$ originating in I_a and ending in a macro-node I_l with no output edges. The length of σ augmented by one unit equals the length of the shortest common prefix for all M_r states in I_a, so that for any string $x = a_1 a_2 \ldots a_l y$ ($y \in \Sigma^+$): $\delta(q_i, x) = \delta(q_j, x)$; $q_i, q_j \in I_a$. Let us define the value: $k_{I_a} = \max_{\sigma(I_a)}\{|\sigma(I_a)| : \sigma(I_a)$ *is a path originating in* $I_a\}$. k_{I_a} exists because PPIG is cycle-free and the value $k_{I_a} + 1$ represents the shortest common prefix to guarantee that all possible paths originating from nodes in I_a with a common prefix have the same intersection node. Therefore, considering all nodes of PPIG, let us define the value: $k_{PPIG} = \max_{I_a \in thePPIG}\{k_{I_a}\}$. Based on what is stated above, the minimum value of k satisfying corollary 2 needs to respect the following constraint:

$$k_{PPIG} + 1 = k_{min} - 1$$

We can finally determine the order of L: $k_{min} = k_{PPIG} + 2$
■

Hence, the complete procedure for order evaluation of language L consists of the following steps:

1. to build the PPIG: complexity $o(|\Sigma|^2 mn)$ (theorem 5)

2. to test whether the PPIG is cycle-free: complexity $o(|\Sigma|^2 m)$

3. to find the longest path in the PPIG: being the PPIG a directed cycle-free graph, the longest path can be determined by algorithms [2] of complexity linear in the number of edges of the graph, thus because of theorem 5: $o(|E_{PPIG}|) = o(|\Sigma|^2 m)$.

Theorem 7 *The optimization problem of order evaluation for a language $L \in LTs.s.$ is polynomial with complexity $o(|\Sigma|^2 mn)$.*

9 Acknowledgments

We are grateful to Eileen Mary Purcell for the precious editing suggestions she gave us.

References

[1] A. Aho, J. Hopcroft, J. Ullman. *The design and analysis of computer algorithms.* Addison-Wesley, 1974

[2] R. K. Ahuja, T. L. Magnanti, J. B. Orlin. *Network flows.* Prentice-Hall, 1993

[3] J. A. Brzozowski, I. Simon. *Characterization of locally testable events.* Discrete Mathematics, Vol. 4, 1973, pp. 243-271

[4] J. A. Brzozowski. *Hierarchies of aperiodic languages.* R.A.I.R.O. Information Théorique (vol. 10, No. 8, août 1976, pp. 33-49)

[5] R. S. Cohen, J. A. Brzozowski. *Dot-depth of star-free events.* J. Computer and System Sc., Vol. 5, 1971, pp. 1-16

[6] S. Crespi-Reghizzi, M. A. Melkanoff, L. Lichten. *The use of grammatical inference for designing programming languages.* Comm. ACM 16, 2 (Feb. 1973), pp. 83-90

[7] M. Harrison. *Introduction to formal language theory.* Addison-Wesley, 1978

[8] J. Hopcroft, J. Ullman. *Introduction to automata theory, languages and computation.* Addison-Wesley, 1979

[9] S. M. Kim, R. McNaughton, R. McCloskey. *A polynomial time algorithm for local testability problem of deterministic finite automata.* I.E.E.E. Trans. Comput., 40, 1991, pp. 1087-1093

[10] S. M. Kim, R. McNaughton. *Computing the order of a locally testable automaton.* SIAM J. Comput. Vol. 23, No. 6, pp. 1193-1215, December 1994

[11] A. Magnaghi. *Local Testability in Parse Trees of Artificial Languages* (in Italian) Master Thesis, Adviser Prof. S. Crespi-Reghizzi, Polytechnic of Milan, 1996

[12] R. McNaughton, S. Papert. *Counter-free automata.* M.I.T. Press, Cambridge, MA, 1971

[13] M. Perles, O. Rabin, E. Shamir. *The theory of definite automata.* I.E.E.E. Trans. Electronic Computers EC-12, 1963, pp.233-243

[14] R. E. Tarjan. *Depth first search and linear graph algorithms.* SIAM J. Computing, 1, 146-60, 1972

[15] A. Potthoff, W. Thomas. *Regular tree languages without unary symbols are star free.* in 9th International Conference on Fundamentals of Computation Theory, Zoltán Ésik, ed., Lecture Notes in Compute Science 710 (1993), pp. 396-405

[16] M. Steinby. *A theory of language varieties.* in Tree Automata and Languages, M. Nivat and A. Podelski (editors), Elsevier Publ., 1992, pp. 57-81

[17] Y. Zalcstein. *Locally testable languages.* J. Computer and System Sc., Vol. 6, 1972, pp. 151-167