RAIRO. INFORMATIQUE THÉORIQUE ET APPLICATIONS

A. AVELLONE
M. GOLDWURM

Analysis of algorithms for the recognition of rational and context-free trace languages

RAIRO. Informatique théorique et applications, tome 32, n° 4-6 (1998), p. 141-152

<http://www.numdam.org/item?id=ITA_1998__32_4-6_141_0>

Informatique théorique et Applications/Theoretical Informatics and Applications
(vol. 32, n° 4-5-6, 1998, pp. 141 à 152)

ANALYSIS OF ALGORITHMS FOR THE RECOGNITION OF RATIONAL AND CONTEXT-FREE TRACE LANGUAGES (*) (1)

by A. AVELLONE and M. GOLDWURM (2)

Communicated by W. BRAUER

Abstract. – *We present an algorithm for the recognition of rational trace languages that has a time complexity at most proportional to the number of prefixes of the input trace. In the worst case it requires $O(n^{\alpha})$ time and $O(n^{\alpha-1})$ space, where α is the size of the maximum clique in the independence alphabet; in the average case, it works in $O(n^{k})$ time, where k is the number of connected components of the dependence alphabet. This algorithm is based on a dynamic programming technique that can also be applied for the recognition of context-free trace languages. Here we present an extension of the classical CYK algorithm that requires $O(n^{3\alpha})$ time and $O(n^{2\alpha})$ space in the worst case, and $O(n^{3k})$ time and $O(n^{2k})$ space in the average case.* © Elsevier Paris

Résumé. – *Nous présentons un algorithme pour la reconnaissance des langages rationnels de traces qui a une complexité en temps au plus proportionnelle au nombre de préfixes de la trace d'entrée. Dans le pire cas, il a un comportement en $O(n^{\alpha})$ en temps et en $O(n^{\alpha-1})$ en espace, où α est la taille de la clique maximale de l'alphabet d'indépendance; en moyenne il a un comportement en $O(n^{k})$ en temps où k est le nombre de composantes connexes de l'alphabet de dépendance. Cet algorithme repose sur une technique de programmation dynamique qui peut aussi être appliquée à la reconnaissance de langages libres de contexte de traces. Nous présentons une extension de l'algorithme classique de CYK qui travaille en temps $O(n^{3\alpha})$ et en espace $O(n^{2\alpha})$ dans le pire cas et en temps $O(n^{3k})$ et en espace $O(n^{2k})$ en moyenne.* © Elsevier Paris

1. INTRODUCTION

In this work we present two algorithms for the recognition of trace languages and study its time and space complexity. Trace languages have been introduced in Computer Science to describe the behaviour of concurrent systems [13] and their properties have been studied in the classical framework of the theory of formal languages [9]. A trace language is defined as a subset of a trace monoid which is given by the quotient of a free monoid with respect

(*) Received January 1998; revised April 1998; accepted May 1998.

(1) This work has been supported by Ministero dell'Università e della Ricerca Scientifica e Tecnologica, Progetto 40% "Algoritmi, modelli di calcolo e strutture informative".

(2) Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano via Comelico 39, 20135 Milano-Italy {avellone,goldwurm}@dsi.unimi.it

Informatique théorique et Applications/Theoretical Informatics and Applications
0988-5004/98/04-05-06/© Elsevier, Paris

to the congruence obtained by allowing the commutation of certain pairs of generators. Hence a trace monoid is defined by a so-called independence alphabet, i.e. a pair $\langle \Sigma, \mathcal{I} \rangle$, where Σ is a finite set of symbols and $\mathcal{I}$ is the set of pairs of commutative elements in Σ. An element of such a monoid is called trace and is denoted by $[x]$, where x is a representative string.

Rational and context-free trace languages can be defined as in the case of free monoids and their main properties are well-known (see, for instance, [16, 6, 5, 14, 1]). In particular the corresponding membership problems have been studied in the literature [6, 15, 3, 7]. We recall that, given a trace monoid M defined by an independence alphabet $\langle \Sigma, \mathcal{I} \rangle$, the membership problem for a trace language $T \subseteq M$ is the problem of determining, for an input string $x \in \Sigma^*$, whether the trace $[x]$ belongs to T. Here, the independence alphabet $\langle \Sigma, \mathcal{I} \rangle$ and the trace language T are fixed and do not belong to the instance of the problem.

A first algorithm for the recognition of rational trace languages is presented in [6] (see also [7]) and works in $O(n^\alpha)$ time in the worst case, where α is the size of the largest clique of the concurrent alphabet. That algorithm requires $\Theta(n^\alpha)$ time and $\Theta(n^\alpha)$ space for an input of size n even in the best case ([1]); hence its time and space complexity is of the same order of growth also in the average case. An attempt to reduce the average case complexity for this problem is achieved in [3] where a procedure is described which works in time at most proportional to $|x| \cdot \sharp \mathrm{Pref}[x]$ for any input $x \in \Sigma^*$, where $\sharp \mathrm{Pref}[x]$ is the number of prefixes of the trace $[x]$. For most independence alphabets this implies a significant reduction of the average computation time since, assuming equiprobable all strings of given length, the mean number of prefixes of a trace of length n is $\Theta(n^k)$, where k is the number of connected components of the dependence alphabet $\langle \Sigma, \mathcal{I}^c \rangle$ [4, 11]. As a consequence, while that algorithm works in $O(n^{\alpha+1})$ time in the worst case, it only requires $O(n^{k+1})$ time in the average case.

Here, we present another algorithm for the recognition of rational trace languages that works in time at most proportional to $\sharp \mathrm{Pref}[x]$ for any input $x \in \Sigma^*$. Then, it requires $O(n^\alpha)$ time in the worst case and $O(n^k)$ time in the average case. Note that we get a linear time algorithm in the average case whenever the dependence graph $\langle \Sigma, \mathcal{I}^c \rangle$ is connected. Moreover, the number of memory cells used during the computation is given by the

([1]) Given two sequences $\{f_n\}$ and $\{g_n\}$ of real numbers, we write $f_n = \Theta(g_n)$ if there exist two positive constants c, C and an integer $n_0 \in \mathbb{N}$ such that $cg_n \leq f_n \leq Cg_n$ for all $n \geq n_0$.

maximum number of prefixes of the input trace that have fixed length. This implies $O\left(n^{\alpha-1}\right)$ space in the worst case and, of course, $O\left(n^{Min\{k,\alpha-1\}}\right)$ space in the average case.

This algorithm is based on a particular representation of the prefixes of a trace introduced in [10] to compute the set of words belonging to a given trace. The same approach allows to simplify the algorithms for the recognition of context-free trace languages described in [7] and [3] which are based on Valiant's algorithm for context-free recognition [17]. Here we present a simpler procedure for the same problem which can be seen as a natural extension of the classical CYK algorithm [12]. It works in $O\left(n^{3\alpha}\right)$ time and $O\left(n^{2\alpha}\right)$ space in the worst case, and in $O\left(n^{3k}\right)$ time and $O\left(n^{2k}\right)$ space in the average case.

In this work, as a model of computation, we consider a Random Access Machine (RAM) under uniform cost criterion [2].

2. PRELIMINARIES

An *independence alphabet* is a pair $\langle \Sigma, \mathcal{I} \rangle$ such that Σ is a finite alphabet and $\mathcal{I} \subseteq \Sigma \times \Sigma$ is a symmetric and irreflexive relation on Σ; it is usually represented by an undirected graph, where Σ is the set of nodes and $\mathcal{I}$ the set of edges. The complementary graph $\langle \Sigma, \mathcal{I}^c \rangle$ will be called *dependence alphabet*. In the following we often use the parameters α and k that represent respectively the size of the maximum clique of $\langle \Sigma, \mathcal{I} \rangle$ and the number of connected components of $\langle \Sigma, \mathcal{I}^c \rangle$. Note that we always have $k \leq \alpha$.

The *trace monoid* defined by an independence alphabet $\langle \Sigma, \mathcal{I} \rangle$, is the quotient $\Sigma^* / \approx_{\mathcal{I}}$ of the free monoid Σ^* with respect to the smallest congruence $\approx_{\mathcal{I}}$ such that $ab \approx_{\mathcal{I}} ba$ for all pairs $(a, b) \in \mathcal{I}$. We denote such a monoid by $\mathbb{M}(\Sigma, \mathcal{I})$. A *trace language* T is a subset of $\mathbb{M}(\Sigma, \mathcal{I})$ and a *trace* is an element of $\mathbb{M}(\Sigma, \mathcal{I})$, usually denoted by $[x]$ for a representative string x. The length of a trace t is given by the length of any of its representatives. Analogously, for any set $L \subseteq \Sigma^*$, we represent by $[L]$ the trace language $\{t \in \mathbb{M}(\Sigma, \mathcal{I}) | \exists x \in L : t = [x]\}$. We also denote by $\cdot$ the product of the monoid $\mathbb{M}(\Sigma, \mathcal{I})$; clearly, for every $t_1, t_2 \in \mathbb{M}(\Sigma, \mathcal{I})$, $t_1 \cdot t_2 \in \mathbb{M}(\Sigma, \mathcal{I})$ is the trace $[xy]$ where $t_1 = [x]$ and $t_2 = [y]$. This operation can be extented to trace languages: for every $T_1, T_2 \subseteq \mathbb{M}(\Sigma, \mathcal{I})$, $T_1 \cdot T_2$ is the trace language $\{t \in \mathbb{M}(\Sigma, \mathcal{I}) | t = t_1 \cdot t_2, t_1 \in T_1, t_2 \in T_2\}$; analogously, the star operation is defined by $T^* = \bigcup_{n=0}^{+\infty} T^n$ for every trace language T, where $T^0 = \{[\varepsilon]\}$ (here ε denotes the empty word) and $T^n = T \cdot T^{n-1}$ for each $n \geq 1$.

The class of *rational trace languages* over the monoid $\mathbb{M}(\Sigma, \mathcal{I})$ is defined as the smallest class of trace languages containing all finite sets and closed under the operations of union, product and star. It is well-known that a trace language T is rational if and only if $T = [L]$ for some regular language $L \subseteq \Sigma^*$ [16]. Analogously, we define the class of *context-free trace languages* as the class of subsets $T \subseteq \mathbb{M}(\Sigma, \mathcal{I})$ such that $T = [L]$, where $L \subseteq \Sigma^*$ is a context-free language [5]. The membership problem for a trace language $T \subseteq \mathbb{M}(\Sigma, \mathcal{I})$ such that $T = [L]$ for some $L \subseteq \Sigma^*$, is equivalent to the problem of verifying, for an input $x \in \Sigma^*$, whether there exists a word $y \in L$ such that $y \approx_{\mathcal{I}} x$.

The notion of prefix of a trace plays a key role in our analysis. Given two traces $p, t \in \mathbb{M}(\Sigma, \mathcal{I})$, we say that p is a *prefix* of t if $t = p \cdot q$ for some $q \in \mathbb{M}(\Sigma, \mathcal{I})$; similarly, q is called *suffix* of t. Note that if p is a prefix of t then there exist $x, y \in \Sigma^*$ such that $t = [x]$, $p = [y]$ and y is a (string) prefix of x. A suffix q of t is said to be *maximum independent* if $q = [a_1 a_2 \cdots a_h]$ where $(a_i, a_j) \in \mathcal{I}$ for every $i \neq j$, and q is the longest suffix of t that has this property.

In the following, for every $x \in \Sigma^*$, $\mathrm{Pref}[x]$ denotes the set of prefixes of $[x]$, $\mathrm{Pref}_i[x]$ the set of prefixes of $[x]$ that have length i, while $\sharp\mathrm{Pref}[x]$ and $\sharp\mathrm{Pref}_i[x]$ are the corresponding cardinalities. Observe that, while the number of prefixes of a word x is $|x| + 1$, in the case of a trace the number of prefixes does not depend only on its length. It is easy to prove that, for every $x \in \Sigma^*$, $\sharp\mathrm{Pref}[x] \leq c|x|^{\alpha}$, where α is the size of the maximum clique of $\langle \Sigma, \mathcal{I} \rangle$ and c is a constant only depending on the independence alphabet. Such a bound can be reduced in the average case: assuming equiprobable all strings $x \in \Sigma^*$ of given length, the average number of prefixes of a trace of length n is $\eta n^k + O(n^{k-1})$, where k is the number of connected components of the dependence alphabet $\langle \Sigma, \mathcal{I}^c \rangle$ and η is a constant only depending on $\langle \Sigma, \mathcal{I} \rangle$ [11].

A trace $t \in \mathbb{M}(\Sigma, \mathcal{I})$ can also be represented by a set of strings as follows. Let $\{\mathcal{A}_1, ..., \mathcal{A}_m\}$ be a clique cover of the dependence alphabet $\langle \Sigma, \mathcal{I}^c \rangle$, that is a family of subsets of Σ such that $\bigcup_{i=1}^{m} \mathcal{A}_i = \Sigma$, $\mathcal{A}_i \times \mathcal{A}_i \subseteq \mathcal{I}^c$ for each $i \in \{1, 2, ..., m\}$ and, for every $\{a, b\} \in \mathcal{I}^c$, both a and b belong to $\mathcal{A}_i$ for some $i \in \{1, 2, ..., m\}$. For instance $\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_m$ could be the set of all maximal cliques in $\langle \Sigma, \mathcal{I}^c \rangle$. For any $a \in \Sigma$, we denote by $I(a)$ the set of indices $\{i \in \{1, ..., m\} | a \in \mathcal{A}_i\}$. Note that, for any $a, b \in \Sigma$, (a, b) belongs to $\mathcal{I}$ if and only if $I(a) \cap I(b) = \emptyset$. For every $i \in \{1, 2, ..., m\}$ and any $x \in \Sigma^*$, we denote by $\Pi_{\mathcal{A}_i}(x)$ the projection of x over the subalphabet $\mathcal{A}_i$.

Now, consider a family of words $\{y_i\}_{i\in\{1,2,...,m\}}$ such that $y_i \in \mathcal{A}_i^*$ for every $i \in \{1, 2, ..., m\}$. For the sake of simplicity we denote it by $\{y_i\}$ when the clique cover is understood. We say that such a family is *reconstructible* if there exists $y \in \Sigma^*$ such that $y_i = \Pi_{\mathcal{A}_i}(y)$ for every $i \in \{1, 2, ..., m\}$. Let $\mathcal{R}(\Sigma, \mathcal{I})$ be the set of all reconstructible families. It is easy to show that, for every pair $\{x_i\}$, $\{y_i\} \in \mathcal{R}(\Sigma, \mathcal{I})$, the family of words $\{x_i y_i\}$ belongs to $\mathcal{R}(\Sigma, \mathcal{I})$. As a consequence $\mathcal{R}(\Sigma, \mathcal{I})$ forms a monoid with respect to the product $\cdot$ defined by $\{x_i\} \cdot \{y_i\} = \{x_i y_i\}$ for all $\{x_i\}$, $\{y_i\} \in \mathcal{R}(\Sigma, \mathcal{I})$. It can be proved that the monoids $\mathbb{M}(\Sigma, \mathcal{I})$ and $\mathcal{R}(\Sigma, \mathcal{I})$ are isomorphic [8, 10]. Hence, every trace $t \in \mathbb{M}(\Sigma, \mathcal{I})$ can be represented by the family $\{x_i\} \in \mathcal{R}(\Sigma, \mathcal{I})$ such that $x_i = \Pi_{\mathcal{A}_i}(x)$ for each $i \in \{1, 2, ..., m\}$ and x is any representative of t.

3. REPRESENTATION OF PREFIXES

Given an independence alphabet $\langle \Sigma, \mathcal{I} \rangle$, let us consider a clique cover $\{\mathcal{A}_1, \mathcal{A}_2, ..., \mathcal{A}_m\}$ of the complementary graph $\langle \Sigma, \mathcal{I}^c \rangle$ and a word $x \in \Sigma^*$. The *representation* of a prefix $p = [y]$ of $[x]$ is the array $(k_1, k_2, ..., k_m) \in \mathbb{N}^m$ such that k_i is the length of $\Pi_{\mathcal{A}_i}(y)$ for every $i \in \{1, 2, ..., m\}$ [10]. Note that $(0, 0, ..., 0)$ represents the empty prefix while the whole trace $[x]$ is represented by $(n_1, n_2, ..., n_m)$ where, for all $i \in \{1, ..., n\}$, n_i is the length of $\Pi_{\mathcal{A}_i}(x)$.

Our goal is to describe an algorithm that computes, for an input $x \in \Sigma^*$ of length n, the representations of all the prefixes of $[x]$. The method we apply is based on the properties of reconstructible families presented in the previous section. Note that for every $p \in \mathrm{Pref}_j[x]$, where $j \in \{1, ..., n\}$, there exists $q \in \mathrm{Pref}_{j-1}[x]$ such that $p = q \cdot [a]$ for a suitable $a \in \Sigma$. Hence, the representation of a prefix of length j can easily be computed from the representation of a suitable prefix of length $j - 1$. This suggests the following Procedure *Build Prefixes* that executes n cycles and, at the j-th cycle, it computes the representations of the prefixes of $[x]$ of length j. In Procedure *Build Prefixes,* for every $i \in \{1, 2, ..., m\}$, x_i denotes the projection $\prod_{\mathcal{A}_i}(x)$ and $x_i = x_i(1)x_2(2) \cdots x_i(n_i)$, where $x_i(j) \in \Sigma$ for all $j \in \{1, 2, ..., n_i\}$.

Procedure *Build Prefixes*

Input: $x \in \Sigma^*$ where $|x| = n$;

Output: $V = \{\underline{k} \in \mathbb{N}^m | \underline{k}$ is the representation of a prefix $p \in \mathrm{Pref}[x]\}$.

```
begin
  let 0 ∈ ℕ^m be the array (0, 0, ..., 0);
  R = V = {0};
  for j = 1, 2, ..., n do
    begin
      S = ∅;
      for r ∈ R do
        for a ∈ Σ (in a given order <) do
          if (∀i ∈ I(a)  x_i(r_i + 1) = a) then
            begin
              for i = 1, 2, ..., m do
(1)             if i ∈ I(a) then s_i = r_i + 1; else s_i = r_i;
(2)           if s ∉ S then S = S ∪ {s};
            end
      R = S; V = V ∪ S;
    end
end
```

The analysis of the procedure depends on the structure we use to represent the set R and S that contain the partial results of the computation. The set R can be represented by a simple list of arrays in $\mathbb{N}^m$ because the procedure only has to scan all its elements. On the contrary, for the set S, we define a data structure that allows to execute in constant time both the test for membership and the operation of insertion required at line (2). The same structure is used by the algorithm presented in section 4.

In order to describe such a structure let us consider the directed graph $\mathcal{G}_{[x]} = \langle V, E \rangle$, having labeled edges, where V is the set of all representations of prefixes of $[x]$, and E is the set of all ordered pairs $(\underline{r}, \underline{s})$ such that $s = r \cdot [a]$ for some $a \in \Sigma$, r and s being the prefixes represented by $\underline{r}$ and $\underline{s}$, respectively, and a is the label of the edge.

It is easy to prove that, for every pair of nodes $\underline{r}$, $\underline{s}$, representing the prefixes r, $s \in \mathrm{Pref}[x]$, and every word $y = a_1 a_2 \cdots a_h \in \Sigma^*$, $s = r \cdot [y]$ if and only if there is a directed path in $\mathcal{G}_{[x]}$ from $\underline{r}$ to $\underline{s}$ labeled by y (i.e. the path has length h and, for each $i \in \{1, 2, ..., h\}$, its i-th edge is labeled by a_i).

To represent this graph in the memory of our RAM model we maintain, for each node $\underline{r} \in V$, two lists $A(\underline{r})$ and $B(\underline{r})$ which yield, respectively,

the edges leaving from and coming into $\underline{r}$. More precisely, for every edge $(\underline{r}, \underline{s}) \in E$ labeled by $a \in \Sigma$, $A(\underline{r})$ contains the pair $(p_{\underline{s}}, a)$, where $p_{\underline{s}}$ is a pointer to the location of $\underline{s}$ in the machine memory. Similarly, for every edge $(\underline{v}, \underline{r}) \in E$ labeled by $a \in \Sigma$, the list $B(\underline{r})$ contains the pair $(p_{\underline{v}}, a)$.

It is clear that, for an input x, Procedure *Build Prefixes* computes all nodes of $\mathcal{G}_{[x]}$. We slightly modify the procedure in order to compute also the lists $A(\underline{r})$ and $B(\underline{r})$ for each node $\underline{r}$. At the j-th cycle, besides computing all nodes $\underline{s}$ representing prefixes of length j, the procedure also determines their lists $B(\underline{s})$ together with the lists $A(\underline{r})$ of all nodes $\underline{r}$ that represent prefixes of length $j-1$; it also maintains in a list S_j the (address of) nodes representing the prefixes of length j once they are computed. Moreover we assume that, during the execution of the j-th cycle, the procedure keeps in memory the lists S_{j-1}, $S_{j-2}, ..., S_{j-\alpha}$ together with lists $A(\underline{r})$ and $B(\underline{r})$ of their nodes $\underline{r}$ (if $j < \alpha$ it only keeps S_{j-1}, $S_{j-2}, ..., S_0$).

Such a data structure allows to test the membership to S_j in constant time. In order to show that, let $\underline{r}$, $\underline{s}$ and a be the elements considered at line (1), and let r and s be the prefixes represented by $\underline{r}$ and $\underline{s}$, respectively. Consider the maximum independent suffix q of s. Observe that a must occur in q. Therefore, let $y = a_1 a_2 \cdots a_h$ be a (possibly empty) word in Σ^* such that $q = [ya]$, where $a_i \in \Sigma$ for each i. We know that the set $C = \{a_1, a_2, ..., a_h, a\}$ forms a clique of $\langle \Sigma, \mathcal{I} \rangle$ which can be computed in constant time since $C = \{b \in \Sigma | \forall i \in I(b)\ x_i(s_i) = b\}$. Now, consider the prefix v of r such that $v \cdot [y] = r$ and let $\underline{v}$ be its representation. Then there exists a directed path in $\mathcal{G}_{[x]}$ from $\underline{v}$ to $\underline{r}$ labeled by y (if $h = 0$ then $v = r$ and $y = \varepsilon$). The edges occurring in this path have already been found by the algorithm because they connect nodes representating prefixes of length smaller than j. Moreover, $\underline{s}$ is already included in S_j if and only if $a < a_i$ for some $i \in \{1, 2, ..., h\}$. In this case there also exists a path of length $h+1$ from $\underline{v}$ to $\underline{s}$ passing through a node $\underline{r}'$ preceding $\underline{r}$ in S_{j-1}.

Hence, in order to test whether $\underline{s} \in S_j$, the procedure verifies whether $a_i < a$ for all $i \in \{1, 2, ..., h\}$: in the affirmative case it adds $\underline{s}$ to S_j, updates $A(\underline{r})$ and sets $A(\underline{s})$ and $B(\underline{s})$ respectively to the empty list and the list only containing $(p_{\underline{s}}, a)$. Otherwise, if $a < a_i$ for some $i \in \{1, 2, ..., h\}$, the procedure first determines the address of node $\underline{v}$ following backwards the path from $\underline{v}$ to $\underline{r}$ labelled by $a_1 a_2 \cdots a_h$ (any permutation of $\{a_1, a_2, ..., a_h\}$ actually leads to $\underline{v}$). Then, it computes the word $y' \in \Sigma^*$ obtained by sorting $\{a_1, a_2, ..., a_h, a\}$ according to $<$ and determines the address of $\underline{s}$ by following the path labelled by y' that starts from $\underline{v}$. Then the procedure

simply updates lists $A(\underline{r})$ and $B(\underline{s})$, adding $(p_{\underline{s}}, a)$ to $A(\underline{r})$ and $(p_{\underline{r}}, a)$ to $B(\underline{s})$.

Using this structure, instruction (2) of the procedure can be executed in constant time (i.e. its time cost only depends on the independent alphabet) because the length of the paths to be followed in $\mathcal{G}_{[x]}$ is smaller or equal to α and, for every node $\underline{r}$, both lists $A(\underline{r})$ and $B(\underline{r})$ contain at most α elements each. This means that, on our RAM model, Procedure *Build Prefixes* executes a constant number of steps for each prefix of the input trace. Therefore, we can state the following result.

PROPOSITION 1: *There are two positive constants a, b such that, for any input $x \in \Sigma^*$, Procedure "Build Prefixes" computes the representations of all prefixes of $[x]$ in time $T(x)$ satisfying the relations*

$$a\sharp\mathrm{Pref}[x] \leq T(x) \leq b\sharp\mathrm{Pref}[x].$$

4. RATIONAL RECOGNITION

Given an independence alphabet $\langle \Sigma, I \rangle$, consider a nondeterministic finite state automaton $\mathcal{A} = \langle Q, q_0, \delta, F \rangle$ over Σ, where Q is the set of states, q_0 the initial state, $\delta : Q \times \Sigma^* \to 2^Q$ the transition function and $F \subseteq Q$ the subset of final states. In this section we describe an algorithm for solving the membership problem for the trace language $[L]$, L being the language recognized by $\mathcal{A}$. Also in this case we assume that the independence alphabet $\langle \Sigma, I \rangle$ and the automaton $\mathcal{A}$ are fixed. Hence the input is simply given by a string $x \in \Sigma^*$.

Now, for every $t \in \mathbb{M}(\Sigma, I)$, let V_t denote the set of states reachable from q_0 by any representative of t, i.e.

$$V_t = \{q \in Q | \exists y \in t : q \in \delta(q_0, y)\}.$$

Then, for an input $x \in \Sigma^*$, the algorithm computes the set $V_{[x]}$. Clearly $[x] \in [L]$ if and only if $V_{[x]} \cap F \neq \emptyset$.

The set $V_{[x]}$ can be computed by applying the following property.

PROPOSITION 2: *Given an independence alphabet $\langle \Sigma, I \rangle$, let $t \in \mathbb{M}(\Sigma, I)$ be a trace of length $n \geq 1$ and let $t_1, t_2, ..., t_j$ be the prefixes of t of length $n-1$. For every $i \in \{1, 2, ..., j\}$ we denote by a_i the element of Σ such that*

$t = t_i \cdot [a_i]$. *Then* V_t *satisfies the following relation*

$$V_t = \bigcup_{i=1}^{j} \{p \in Q | \exists q \in V_{t_i} : p \in \delta(q, a_i)\}.$$

The proposition can be easily proved by induction on the length n of trace t. This property allows to compute the set V_t for a trace t of length n, once the sets V_{t_i} are known for all the prefixes t_i of t having length $n-1$. Therefore we can apply the same method used in the previous section to compute the representations of the prefixes of a given trace.

The algorithm follows exactly the scheme of Procedure *Build Prefixes*. It executes n cycles, where n is the length of the input string. At the j-th cycle it computes the sets V_s for all the prefixes s of the input trace that have length j together with the corresponding representations. The computation uses the sets V_r of the prefixes r of length $j-1$ that have been computed at the previous cycle. When an edge $(\underline{r}, \underline{s}) \in E$ with label $a \in \Sigma$ is found, then, using V_r, the algorithm computes the set

$$T = \{p \in Q | \exists q \in V_{\underline{r}} : p \in \delta(q, a)\}$$

and adds it to V_s.

Since the computation of set T can be done in constant time, the analysis of the algorithm can be achieved as in the previous section. This implies, for all inputs $x \in \Sigma^*$, an overall computation time $T(x)$ such that $a\sharp\mathrm{Pref}[x] \leq T(x) \leq b\sharp\mathrm{Pref}[x]$, where a and b are fixed positive constants.

As regards the space complexity note that, during the execution of the j-th cycle, the algorithm only has to keep the set V_r for all $\underline{r} \in S_{j-1}$ and the lists $S_{j-1}, S_{j-2}, \ldots, S_{j-\alpha}$. As a consequence, the number of memory cells used for an input $x \in \Sigma^*$ of length n is at most proportional to

$$\max\{\sharp\mathrm{Pref}_i[x] | i = 1, 2, \ldots, n\}.$$

PROPOSITION 3: *For every independence alphabet* $\langle \Sigma, I \rangle$ *there exists a constant* $c > 0$ *such that, for all* $x \in \Sigma^*$ *of length* n *and every* $i \in \{1, 2, \ldots, n\}$,

$$\sharp\mathrm{Pref}_i[x] \leq cn^{\alpha-1},$$

where α *is the size of the largest clique in* $\langle \Sigma, I \rangle$.

Proof: Let x be a string of length $n \geq 1$ in Σ^* and let C be a clique of size α in (Σ, I). We can consider C as a trace in $\mathbb{M}(\Sigma, I)$ whose symbols are distinct, mutually independent elements. It is not difficult to see that, for every $i \in \{1, 2, ..., n\}$,

$$\sharp \mathrm{Pref}_i[x] \leq \sharp \mathrm{Pref}_i(C^m)$$

where $m = \lceil \frac{n}{\alpha} \rceil$ and C^m is the m-th power of C in the monoid $\mathbb{M}(\Sigma, I)$.

Now observe that $\sharp \mathrm{Pref}_i(C^m)$ is at most the number of i-combinations with repetitions of a set of α elements. Hence, for every $i \in \{1, 2, ..., n\}$, we have $\sharp \mathrm{Pref}_i(C^m) \leq \binom{i+\alpha-1}{\alpha-1}$. Since $\binom{i+\alpha-1}{\alpha-1} \leq ci^{\alpha-1}$ for a suitable constant $c > 0$ only depending on α, we obtain the result. □

The analysis presented in this section can be summarized by the following.

PROPOSITION 4: *Given an independence alphabet $\langle \Sigma, \mathcal{I} \rangle$ and a finite state automaton $\mathcal{A}$ accepting a language $L \subseteq \Sigma^*$, the membership problem for the trace language $[L]$ can be solved by an algorithm that, in the worst case, requires $O(n^\alpha)$ time and $O(n^{\alpha-1})$ space, where α is the size of the largest clique of $\langle \Sigma, \mathcal{I} \rangle$. In the average case, assuming equiprobable all input strings of length n, it works in $O(n^k)$ time and $O(n^{Min\{k,\alpha-1\}})$ space, where k is the number of connected components of $\langle \Sigma, \mathcal{I}^c \rangle$.*

5. CONTEXT-FREE RECOGNITION

A further application of the procedure presented in Section 3 concerns the membership problem for context-free trace languages. A procedure for solving the problem can be obtained as a natural extension of the classical CYK algorithm for context-free (string) recognition. An analogous extension has been obtained for Valiant's algorithm [7].

Given an independence alphabet $\langle \Sigma, \mathcal{I} \rangle$, let $G = \langle V, \Sigma, S, P \rangle$ be a context-free grammar in Chomsky normal form which generates a language $L \subseteq \Sigma^*$, where V is the set of variables, $S \in V$ is the initial variable and P the set of productions. For an input $x \in \Sigma^*$ of length n the algorithm verifies whether there exists a string $y \in L$ that belongs to $[x]$. In an initial phase Procedure *Build Prefixes* is applied to compute the sets $P_0, P_1, ..., P_n$ where, for each i, $P_i = \{\underline{r} \in \mathbb{N}^m | \underline{r}$ represents a prefix $r \in \mathrm{Pref}_i[x]\}$. Then, for each pair of integers i, j, $0 \leq i < j \leq n$, and all pairs of representations

$(\underline{r}, \underline{s}) \in P_i \times P_j$, the algorithm computes the set

$$T_{\underline{r},\underline{s}} = \{A \in V | \exists y \in \Sigma^* : A \Rightarrow_G^* y,\ r \cdot [y] = s\}.$$

Clearly, a positive answer is returned if and only if $S \in T_{\underline{0},\underline{x}}$, where $\underline{0}$ and $\underline{x}$ represent $[\varepsilon]$ and $[x]$, respectively. Observe that, for each $i \in \{0, 1, ..., n-1\}$ and every $(\underline{r}, \underline{s}) \in P_i \times P_{i+1}$, we have $T_{\underline{r},\underline{s}} = \{A \in V | \exists (A \to a) \in P : r \cdot [a] = s\}$. Moreover, for every $\ell \in \{2, 3, ..., n\}$, $i \in \{0, 1, ..., n - \ell\}$, and each $(\underline{r}, \underline{s}) \in P_i \times P_{i+\ell}$, a variable A belongs to $T_{\underline{r},\underline{s}}$ if and only if, for some $h \in \{1, ..., \ell - 1\}$, there are $\underline{u} \in P_{i+h}$ and $(A \to BC) \in P$ such that $B \in T_{\underline{r},\underline{u}}$ and $C \in T_{\underline{u},\underline{s}}$. Therefore the algorithm first computes $T_{\underline{r},\underline{s}}$ for each $(\underline{r}, \underline{s}) \in P_i \times P_{i+1}$ and all $i \in \{0, 1, ..., n - 1\}$. Then, it executes the following procedure:

begin
  for $\ell \in \{2, 3, ..., n\}$ **do**
    for $i \in \{0, 1, ..., n - \ell\}$ **do**
      for $(\underline{r}, \underline{s}) \in P_i \times P_{i+\ell}$ **do**
        begin
          $T_{\underline{r},\underline{s}} = \emptyset$;
          for $h = \{1, 2, ..., \ell - 1\}$ **do**
            for $\underline{u} \in P_{i+h}$ **do**
          $T_{\underline{r},\underline{s}} = T_{\underline{r},\underline{s}} \cup \{A \in V | \exists (A \to BC) \in P : B \in T_{\underline{r},\underline{u}}, C \in T_{\underline{u},\underline{s}}\}$;
        end
end

Note that, if $\mathcal{I} = \emptyset$, the previous procedure reduces to the classical CYK algorithm.

It is easy to verify that the algorithm works in time $T(x)$ and space $S(x)$ such that

$$a(\sharp \mathrm{Pref}[x])^3 \leq T(x) \leq b(\sharp \mathrm{Pref}[x])^3, c(\sharp \mathrm{Pref}[x])^2 \leq S(x) \leq d(\sharp \mathrm{Pref}[x])^2,$$

for suitable positive constants a, b, c, d and for every input $x \in \Sigma^*$. This proves the following result:

PROPOSITION 5: *An extension of the CYK algorithm solves the membership problem for context-free trace language in $O(n^{3\alpha})$ time and $O(n^{2\alpha})$ space in the worst case, and in $O(n^{3k})$ time and $O(n^{2k})$ space in the average case, where α and k are the coefficients introduced in the previous propositions.*

REFERENCES

1. I. J. AALBERSBERG and E. WELZL, Trace languages defined by regular string languages, *R.A.I.R.O. – Informatique Théorique et Applications,* 1986, *20*, pp. 103-119.
2. A. V. AHO, J. E. HOPCROFT and J. D. ULLMAN, *The design and analysis of computer algorithms,* Addison-Wesley, 1974.
3. A. BERTONI and M. GOLDWURM, On the prefixes of a random trace and the membership problem for context-free trace languages. In L. HUGUET and A. POLI, eds, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes (Proceedings AAECC 5), Menorca (Spain) June 15-19, 1987,* number 356 in Lecture Notes in Computer Science, pp. 35-59, Berlin-Heidelberg-New York, 1989, Springer.
4. A. BERTONI, M. GOLDWURM and N. SABADINI, Analysis of a class of algorithms for problems on trace languages. In Th. BETH and M. CLAUSEN, eds., *Applicable Algebra, Error-Correcting Codes, Combinatorics and Computer Algebra (Proceedings AAECC 4), Karlsruhe (FRG) September 23-26, 1986,* number 307 in Lecture Notes in Computer Science, pp. 202-214, Berlin-Heidelberg-New York, 1988, Springer.
5. A. BERTONI, G. MAURI and N. SABADINI, Context-free trace languages, In *Proceedings 7th Coll. on Trees in Algebra and Programming (CAAP), Lille (France),* 1982, pp. 32-42.
6. A. BERTONI, G. MAURI and N. SABADINI, Equivalence and membership problems for regular trace languages, In *Proc. 9th ICALP,* number 140 in Lecture Notes in Computer Science, pp. 61-71, Berlin-Heidelberg-New York, 1982, Springer.
7. A. BERTONI, G. MAURI and N. SABADINI, Membership problems for regular and context free trace languages, *Information and Computation,* 1989, *82*, pp. 135-150.
8. R. CORI and D. PERRIN, Automates et commutations partielles, *R.A.I.R.O. – Informatique Théorique et Applications,* 1985, *19* (1), pp. 21-32.
9. V. DIEKERT and G. ROZENBERG (Eds.), *The book of traces,* World Scientific, 1995.
10. C. DUBOC, Commutations dans des monoides libres : un cadre théorique pour l'étude du parallélisme. Thèse, Faculté des Sciences de l'Université de Rouen, 1986.
11. M. GOLDWURM, Probabilistic estimation of the number of prefixes of a trace, *Theoretical Computer Science,* 1992, *92*, pp. 249-268.
12. M. H. HARRISON, *Introduction to formal language theory,* Addison-Wesley, 1978.
13. A. MAZURKIEWICZ, Concurrent program schemes and their interpretations, DAIMI Rep. PB 78, Aarhus University, Aarhus, 1977.
14. E. OCHMANSKI, Regular behaviour of concurrent systems, *Bulletin of the European Association for Theoretical Computer Science (EATCS),* Oct 1985, *27*, pp. 56-67.
15. W. RYTTER, Some properties of trace languages, *Fundamenta Informaticae,* VII, 1984, pp. 117-127.
16. M. SZIJÁRTÓ, A classification and closure properties of languages for describing concurrent system behaviours, *Fundamenta Informaticae,* 1981, *4* (3), pp. 531-549.
17. L. G. VALIANT, General context-free recognition in less than cubic time, *Journal of Computer and System Sciences,* 1975, *10*, pp. 308-315.