INFORMATION AND COMPUTATION 108, 262–285 (1994)

On the Existence of Minimum Asynchronous Automata and on the Equivalence Problem for Unambiguous Regular Trace Languages*

DANILO BRUSCHI, GIOVANNI PIGHIZZINI, AND NICOLETTA SABADINI

Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, Via Comelico 39, 20135 Milano, Italy

Given a recognizable trace language T there always exists a minimum (up to isomorphism) monoid automaton which recognizes T. In this paper we consider finite state asynchronous automata, introduced by W. Zielonka ("Notes on Finite Asynchronous Automata," Technical Report, Institute of Mathematics, Warsaw Technical University, 1984) and we show that this result *cannot* be extended to this model. More precisely, we exhibit a recognizable trace language which does not admit a unique minimum asynchronous automaton. This result solves an open problem posed by Zielonka. We further prove that every recognizable trace language defined on a concurrent alphabet with a transitive dependency relation admits a unique minimum asynchronous automaton. In the second part of this paper we investigate the equivalence problem for regular trace languages. This problem has been investigated by many authors: in particular, Aalbersberg and Hoogeboom (1987) have shown that the equivalence problem for regular trace languages defined over a concurrent alphabet $\langle \Sigma, C \rangle$ is decidable *if and only if* the concurrency relation C is transitive. We show that this result does not hold when we restrict our attention to unambiguous regular trace languages: we exhibit a concurrent alphabet $\langle \Sigma', C' \rangle$ with a *non* transitive concurrency relation such that the equivalence problem for unambiguous regular trace languages on $\langle \Sigma', C' \rangle$ can be decided in polynomial time. Further we show that the problem of distinguishing whether or not a regular expression on a concurrent alphabet $\langle \Sigma, C \rangle$ defines an unambiguous regular trace language is *undecidable*. © 1994 Academic Press, Inc.

1. INTRODUCTION

In 1977 Mazurkiewicz introduced the idea of describing the behavior of concurrent systems through a generalization of the usual notion of formal language, considering languages on partially commutative monoids. He called such languages *trace languages*. Since then, many results have appeared which characterize and study properties of classes of trace languages (for a survey of these results the reader is referred to Aalbersberg

* A preliminary version of this paper appeared in the "Proceedings of the Fifth EATCS Symposium on Theoretical Aspects of Computer Science, Bordeaux, 1988," Lecture Notes in Computer Science, Vol. 294, pp. 334–346.

0890-5401/94 $6.00

Copyright © 1994 by Academic Press, Inc.
All rights of reproduction in any form reserved.

and Rozenberg, 1988). In particular, following the approach already used in the "classical" theory of formal languages, Mazurkiewicz (1977) introduced the notion of *regular trace language* and subsequently Bertoni *et al.* (1981) introduced the notion of *recognizable trace language* and Bertoni *et al.* (1985) that of *unambiguous trace language.*

This paper examines some problems related to such languages. More precisely, it consists of two independent parts. The first part solves an open problem concerning the existence of minimum asynchronous automata for given recognizable trace languages; the second part deals with the computational complexity of the equivalence problem for unambiguous regular trace languages.

In 1984, Zielonka (1984) introduced a new model of distributed recognizing devices called *finite state asynchronous automata*, and characterized the class of languages recognized by such machines as the class of recognizable trace languages. In the same work, Zielonka also posed the problem of finding an algorithm which, given an asynchronous automaton recognizing a trace language T, constructed the minimum asynchronous automaton recognizing T. In the first part of the paper, we exhibit a recognizable trace language which does not admit a unique minimum asynchronous automaton, thus *showing that, in general, this problem cannot be solved.* We further prove that every recognizable trace language defined on a concurrent alphabet with a transitive dependency relation admits a unique minimum asynchronous automaton.

With respect to the problems considered in the second part of the paper, we briefly recall that Bertoni *et al.* (1982) had shown that the *equivalence problem* for *regular* trace languages is undecidable. However, in the same paper, they also had shown that the equivalence problem is decidable if we only consider regular trace languages defined over concurrent alphabets with a transitive concurrency relation. Subsequently Aalbersberg and Welzl (1986) extended this result to universally regular trace languages, and finally, Aalbersberg and Hoogeboom (1987) shown that the equivalence problem for regular trace languages defined over a concurrent alphabet $\langle \Sigma, C \rangle$ is decidable *if and only if* the concurrency relation C is transitive.

This fact raises the question if the equivalence problem can be decided for some interesting subclasses of regular trace languages in a broader class of concurrent alphabets. We have been able to give an affirmative answer to such a question. In fact, we have found a concurrent alphabet $\langle \Sigma, C \rangle$, where C is *non* transitive, such that the equivalence problem for unambiguous regular trace languages over $\langle \Sigma, C \rangle$ can be decided in polynomial time by a deterministic Turing machine.

Finally, we show that the problem of distinguishing whether or not a given regular expression over $\langle \Sigma, C \rangle$ defines an unambiguous regular trace language is *undecidable.*

We point out that the terminology on trace languages is not yet well defined; for example, regular trace languages are called existentially regular in Aalbersberg and Rozenberg (1988) and rational in Sakarovitch (1987). Recognizable trace languages are called consistently regular in Aalbersberg and Rozenberg (1988) and sequentially regular in Zielonka (1987).

2. Preliminary Definitions

In this section, basic definitions and facts about trace languages and algebraic structures supporting them, i.e., free partially commutative monoids, are recalled.

Definition 2.1. A *concurrent alphabet* is a pair $\langle \Sigma, C \rangle$, where

(a) $\Sigma = \{\sigma_1, \sigma_2, ..., \sigma_m\}$ is a finite alphabet;

(b) $C \subseteq \Sigma \times \Sigma$ is a symmetric and irreflexive relation, the *concurrency relation.*

The complementary relation of the concurrency relation C is called *dependency relation* and in the following it will be denoted by C^c. As usual C and C^c will be represented as graphs.

Definition 2.2. The *free partially commutative monoid* (fpcm, for short) $F(\Sigma, C)$ generated by a concurrent alphabet $\langle \Sigma, C \rangle$ is the quotient structure $F(\Sigma, C) = \Sigma^* / \equiv_C$, where $\equiv_C$ is the least congruence on Σ^* which extends the set of "commutativity laws" $\{\sigma_1\sigma_2 = \sigma_2\sigma_1$ s.t. $\sigma_1, \sigma_2 \in \Sigma$ and $(\sigma_1, \sigma_2) \in C\}$.

Definition 2.3. A *trace* t on a concurrent alphabet $\langle \Sigma, C \rangle$ is an element $t \in F(\Sigma, C)$; i.e., a trace is a congruence class of words. $[w]_C$ denotes the class containing the string w. A *trace language* T is a subset of $F(\Sigma, C)$.

Given a language $L \subseteq \Sigma^*$ it is possible to associate with it a trace language as follows:

Definition 2.4. Given a language L on Σ and a concurrency relation $C \subseteq \Sigma \times \Sigma$, the set $[L]_C = \{[w]_C \mid w \in L\}$ is the *trace language generated by L under C.*

Conversely, it is possible to associate with every trace language $T \subseteq F(\Sigma, C)$ a language $\text{lin}(T)$ on Σ as follows:

DEFINITION 2.5. Given the canonical morphism ψ from Σ^* to $F(\Sigma, C)$, we define the *linearization* of a trace t as $\text{lin}(t) = \psi^{-1}(t)$. The linearization of a trace language T is the language $\text{lin}(T) = \bigcup_{t \in T} \text{lin}(t)$.

We define on the set $2^{F(\Sigma, C)}$ of trace languages the usual operations of union $\cup$, product $\cdot$ and star closure $(-)^*$ with the usual meanings. As in the sequential case, the class $\mathbf{Reg}(\Sigma, C)$ of *regular trace languages* on $\langle \Sigma, C \rangle$ is defined as follows [Mazurkiewicz (1977)]:

DEFINITION 2.6. The class $\mathbf{Reg}(\Sigma, C)$ of *regular trace languages* over the concurrent alphabet $\langle \Sigma, C \rangle$ is the least class of subsets of $F(\Sigma, C)$ containing finite trace languages and closed with respect to the operations $\cup, \cdot, (-)^*$,

The class of unambiguous regular trace languages [Bertoni *et al.* (1985)] can be defined analogously to the class of regular trace languages, introducing the operations $\sqcup$, $\circ$, and $(-)^{@}$, obtained respectively as unambiguous restrictions of the operations, $\cup$, $\cdot$, $(-)^*$. More formally $\sqcup$, $\circ$, and $(-)^{@}$ are defined as follows:

$$T_1 \sqcup T_2 = \begin{cases} T_1 \cup T_2 & \text{if } T_1 \cap T_2 = \varnothing \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$T_1 \circ T_2 = \begin{cases} T_1 \cdot T_2 & \text{if } (\forall x, z \in T_1, \forall y, w \in T_2 \\ & \qquad (xy = zw \rightarrow x = z \wedge y = w)) \\ \text{undefined} & \text{otherwise.} \end{cases}$$

$$T^{@} = \begin{cases} T^* & \text{if } T \text{ is the basis of a free} \\ & \qquad \text{submonoid } T^* \text{ of } F(\Sigma, C) \\ \text{undefined} & \text{otherwise.} \end{cases}$$

DEFINITION 2.7. The class $\mathbf{UR}(\Sigma, C)$ of *unambiguous regular trace languages* over the concurrent alphabet $\langle \Sigma, C \rangle$ is the least class of subsets of $F(\Sigma, C)$ containing finite trace languages and closed with respect to $\sqcup$, $\circ$, and $(-)^{@}$.

The class of *recognizable trace languages* is usually defined through the notion of automaton on a monoid M or M-automaton. In the following we briefly recall the definition of M-automaton and subsequently that of recognizable trace language.

DEFINITION 2.8. Let M be a monoid with unit 1. An *M-automaton* A is a quadruple $A = \langle Q, q_0, \delta, F \rangle$, where:

— Q is a finite set of states;

— $\delta: Q \times M \to Q$ is a transition function such that

$$\delta(q, 1) = q \qquad \text{for every} \quad q \in Q$$

$$\delta(q, mm') = \delta(\delta(q, m), m') \qquad \text{for every} \quad m, m' \in M, q \in Q$$

— $q_0 \in Q$ is the initial state

— $F \subseteq Q$ is the set of final states.

DEFINITION 2.9. The language recognized by an M-automaton A is the set $L = \{m \in M \mid \delta(q_0, m) \in F\}$.

We recall that a state $q \in Q$ is reachable in A iff there exists $m \in M$ s.t. $\delta(q_0, m) = q$; the automaton A is *reachable* iff every state in Q is reachable.

We observe that given an $F(\Sigma, C)$-automaton $A = \langle Q, q_0, \delta, F \rangle$, for every state $q \in Q$ and for every pair $(a, b) \in C$, it holds that $\delta(q, ab) = \delta(q, ba)$. This means that in $F(\Sigma, C)$-automata the concurrency among independent actions is reduced to their interleaving.

DEFINITION 2.10. A trace language $T \subseteq F(\Sigma, C)$ is called *recognizable* iff there exists an $F(\Sigma, C)$-automaton which recognizes T. The class of recognizable trace languages on the concurrent alphabet $\langle \Sigma, C \rangle$ is denoted by $\mathbf{Rec}(\Sigma, C)$.

While regular languages on Σ^* exactly characterize the recognizing power of finite state automata, the class of trace languages recognizable by $F(\Sigma, C)$-automata is a proper subclass of regular trace languages. Moreover, the following inclusions hold:

THEOREM 2.1 (Bertoni, *et al.*, 1981; Bertoni, *et al.*, 1985; Sakarovitch, 1987). *For every concurrent alphabet* $\langle \Sigma, C \rangle$ *the following relations hold*:

(1) $\mathbf{Rec}(\Sigma, C) \subseteq \mathbf{UR}(\Sigma, C) \subseteq \mathbf{Reg}(\Sigma, C)$

(2) $\mathbf{Rec}(\Sigma, C) = \mathbf{UR}(\Sigma, C)$ *iff* $C = \varnothing$

(3) $\mathbf{UR}(\Sigma, C) = \mathbf{Reg}(\Sigma, C)$ *iff* C *is transitive.*

We observe that (3) is the generalization to arbitrary transitive relations of the well known result of Eilenberg and Schützenberger (1968), given for totally commutative monoids.

Definitions 2.4 and 2.5 allow us to characterize the classes of regular, recognizable, and unambiguous regular trace languages in terms of languages on Σ:

THEOREM 2.2 (Szijarto, 1981; Bertoni *et al.*, 1985). *Given a trace language* $T \subseteq F(\Sigma, C)$ *it holds that*

— T *is regular iff there exists a regular language* $L \subseteq \Sigma^*$ *s.t.* $[L]_C = T$,

— T *is recognizable iff* $\mathrm{lin}(T)$ *is a regular language on* Σ,

— T *is unambiguously regular iff there exists a regular language* L *s.t.*:

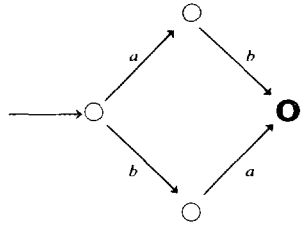
(1) $T = [L]_C$

(2) $x \in L$ *implies* $\#(\mathrm{lin}([x]_C) \cap L) \leqslant 1$.

An easily corollary following Theorem 2.2 is that a trace language $T \subseteq F(\Sigma, C)$ is unambiguous if there exists a regular language L such that $T = [L]_C$ and $t \in F(\Sigma, C)$ implies $\#(\mathrm{lin}(t) \cap L) \leqslant 1$, or equivalently $T = [L]_C$ and $t \in T$ implies $\#(\mathrm{lin}(t) \cap L) = 1$.

3. ASYNCHRONOUS AUTOMATA

Automata are useful mathematic models of sequential systems. As we remarked in the previous section, they only can represent concurrency as interleaving. For example, the automaton A given by the following transition diagram (where the bold circle denotes the final state) can represent two deeply different systems:



The first represents the execution of a sequence of two dependent actions, a and b; the second represents the execution of two independent actions, a and b. Thus, $F(\Sigma, C)$-automata do not distinguish concurrent systems from sequential systems. To achieve this goal, Zielonka (1987) has introduced the notion of asynchronous automata, whose formal definition and properties are recalled in this section.

An asynchronous automaton appears as a set of control units which can act independently or synchronized. Every action, represented by a symbol, is processed by a subset of control units; two actions are independent if and only if they are processed by disjoint sets of control units. Thus, asynchronous automata can be seen as abstractions of distributed systems.

Given a finite number of sets $A_1, ..., A_k$, let $\prod_{i=1}^{k} A_i$ denote the cartesian product of $A_1, ..., A_k$.

DEFINITION 3.1. A *finite state asynchronous automaton* (FSAA) A over a concurrent alphabet $\langle \Sigma, C \rangle$ with n processes is a tuple $A = \langle P_1, P_2, ..., P_n, \Delta, F \rangle$ such that:

— $\forall i,\ 1 \leqslant i \leqslant n,\ P_i = \langle \Sigma_i, S_i, s_{i0} \rangle$ is the ith process, where Σ_i is a finite non empty *local alphabet* such that $\Sigma_1, ..., \Sigma_n$ are the maximal cliques of the dependency relation C^c, S_i is a finite set of *local states* s.t. $S_i \cap S_j = \varnothing$ for $j \neq i$, and $s_{i0} \in S_i$ is the *initial state* of P_i;

— Δ is a set of *transition functions* containing exactly one transition function δ_σ for every action σ belonging to the *global alphabet* $\Sigma = \bigcup_{i=1}^{n} \Sigma_i$. The transition function δ_σ is a (possibly partial) function from $\prod_{i \in \mathrm{Dom}(\sigma)} S_i$ in itself; that is $\delta_\sigma : \prod_{i \in \mathrm{Dom}(\sigma)} S_i \to \prod_{i \in \mathrm{Dom}(\sigma)} S_i$, where $\mathrm{Dom}(\sigma) = \{ i \mid \sigma \in \Sigma_i \}$ is the *domain* of σ, that is the set of processes that can *execute* σ;

— $F \subseteq \prod_{i=1}^{n} S_i$ is the set of *final states*.

Every tuple $(s_1, ..., s_n) \in \prod_{i=1}^{n} S_i$ is called *global state* of A.

We underline that the domains $\mathrm{Dom}(\sigma)$ and $\mathrm{Dom}(\sigma')$ of two actions $\sigma, \sigma' \in \Sigma$ are disjoint if and only if σ and σ' are independent; in this case the functions δ_σ and $\delta_{\sigma'}$ act on disjoint sets of local states and, consequently, the actions σ and σ' can be executed concurrently. In this way, asynchronous automata over $\langle \Sigma, C \rangle$ represent *all* the concurrency among actions, specified by the relation C. Note that the two systems related to the $F(\Sigma, C)$-automaton introduced at the beginning of this section are represented by different asynchronous automata.

For describing the "global behavior" of a given asynchronous automaton A, we associate with every trace t a *t-reachability relation* $\overset{t}{\Longrightarrow}$ on the set of global states, such that a pair (q, p) belongs to $\overset{t}{\Longrightarrow}$ if and only if the automaton A starting from q and executing the trace t reaches the state p.

In order to formally define $\overset{t}{\Longrightarrow}$, first we define for every $\sigma \in \Sigma$ the reachability relation $\overset{\sigma}{\Longrightarrow}$ on the set of global states. Intuitively a pair (q, p) of global states belongs to $\overset{\sigma}{\Longrightarrow}$ if each local state in q corresponding to a process executing σ has a corresponding local state in p computed according to δ_σ, remaining local states in q and p are respectively identical. More formally:

DEFINITION 3.2. Given a finite state asynchronous automaton $A = \langle P_1, ..., P_n, \Delta, F \rangle$ over $\langle \Sigma, C \rangle$, for every $\sigma \in \Sigma$ the *σ-reachability relation* $\overset{\sigma}{\Longrightarrow} \subseteq (\prod_{i=1}^{n} S_i) \times (\prod_{i=1}^{n} S_i)$ is defined as follows:

$$\forall (s_1, ..., s_n), (u_1, ..., u_n) \in \prod_{i=1}^{n} S_i,$$

$$(s_1, ..., s_n) \overset{\sigma}{\Longrightarrow} (u_1, ..., u_n) \text{ iff } \forall i \notin \mathrm{Dom}(\sigma)\ s_i = u_i$$

$$\text{and } (u_{i_1}, ..., u_{i_k}) = \delta_\sigma(s_{i_1}, ..., s_{i_k})$$

where $\{i_1, ..., i_k\} = \mathrm{Dom}(\sigma)$.

Now, given the relation $\overset{\sigma}{\Longrightarrow}$, we inductively define a reachability relation $\overset{x}{\Longrightarrow}$ for strings, which will turn out to coincide with the t-reachability relation. More precisely,

DEFINITION 3.3. The ε-reachability relation $\overset{\varepsilon}{\Longrightarrow}$ is the identity relation, that is,

$$\overset{\varepsilon}{\Longrightarrow} = \left\{ (q, p) \in \left(\prod_{i=1}^{n} S_i \right) \times \left(\prod_{i=1}^{n} S_i \right) \middle| q = p \right\},$$

and, for every $x \in \Sigma^*$ and $\sigma \in \Sigma$, the $x\sigma$-reachability relation is the set

$$\overset{x\sigma}{\Longrightarrow} = \left\{ (q, p) \in \left(\prod_{i=1}^{n} S_i \right) \times \left(\prod_{i=1}^{n} S_i \right) \middle| \exists r \in \prod_{i=1}^{n} S_i \text{ s.t. } q \overset{x}{\Longrightarrow} r \text{ and } r \overset{\sigma}{\Longrightarrow} p \right\}.$$

Observe that $\overset{\sigma\sigma'}{\Longrightarrow} = \overset{\sigma'\sigma}{\Longrightarrow}$ for every $(\sigma, \sigma') \in C$, so we can define the t-reachability relation $\overset{t}{\Longrightarrow}$, for every trace t, as

$$\overset{t}{\Longrightarrow} = \overset{x}{\Longrightarrow},$$

where x is a string such that $[x]_C = t$.

Now, using the notion of reachability, we can define the *trace language* $T \subseteq F(\Sigma, C)$ *recognized* by a given asynchronous automaton A as the set of traces t such that a final state is t-reachable in A from the initial state; that is,

$$T = \{ t \in F(\Sigma, C) \mid (s_{10}, ..., s_{n0}) \overset{t}{\Longrightarrow} (s_1, ..., s_n) \text{ and } (s_1, ..., s_n) \in F \}.$$

The next two theorems show that the class of trace languages recognized by the usual monoid automata and the class of trace languages recognized by asynchronous automata coincide. Theorem 3.1 can easily be proved by observing that given an asynchronous automaton A over the concurrent alphabet $\langle \Sigma, C \rangle$ we can easily obtain an $F(\Sigma, C)$-automaton accepting the same trace language of A.

The converse result, stated in Theorem 3.2, has been proved by Zielonka (1987).

THEOREM 3.1. *Every trace language T recognizable by a finite state asynchronous automaton over the concurrent alphabet $\langle \Sigma, C \rangle$, belongs to the class* $\mathbf{Rec}(\Sigma, C)$; *i.e., T is a trace language recognizable by $F(\Sigma, C)$-automata.*

THEOREM 3.2. *For every trace language $T \in \mathbf{Rec}(\Sigma, C)$ there exists a finite state asynchronous automaton A over $\langle \Sigma, C \rangle$ that recognizes T.*

The last theorem is quite surprising because it implies the fact, not at all intuitive, that for every "commutative" system represented by an $F(\Sigma, C)$-automaton there exists a "concurrent" system (asynchronous automaton over $\langle \Sigma, C \rangle$) with the same behaviour (language accepted).

In the next section we are interested in minimizing asynchronous automata. For formalizing the concept of minimum asynchronous automaton, we have to introduce the preliminar notions of reachable state and of reachable finite state asynchronous automaton and some important properties of this type of automata. In particular from this point on *we consider only reachable automata.*

DEFINITION 3.4. Given an asynchronous automaton A over $\langle \Sigma, C \rangle$ and a set $\alpha = \{i_1, ..., i_k\} \subseteq \{1, ..., n\}$, a tuple of local states $(s_{i_1}, ..., s_{i_k})$, $s_{i_j} \in S_{i_j}$, $1 \leqslant j \leqslant k$, is said to be *reachable* if there exist a trace $t \in F(\Sigma, C)$ and a global state $(u_1, ..., u_n) \in \prod_{i=1}^{n} S_i$ such that $(s_{10}, ..., s_{n0}) \overset{t}{\Longrightarrow} (u_1, ..., u_n)$, and $u_{i_j} = s_{i_j}$, for $j = 1, ..., k$.

DEFINITION 3.5. An asynchronous automaton A is *reachable* if and only if the following conditions hold:

(1) every local state $s \in S_i$, $1 \leqslant i \leqslant n$, is reachable;

(2) for every $\sigma \in \Sigma$, with $\mathrm{Dom}(\sigma) = \{i_1, ..., i_k\}$ and $(s_{i_1}, ... s_{i_k}) \in \prod_{i \in \mathrm{Dom}(\sigma)} S_i$, if $(s_{i_1}, ..., s_{i_k})$ is not reachable, then $\delta_\sigma(s_{i_1}, ..., s_{i_k})$ is not defined.

Clearly, given a nonreachable asynchronous automaton A, it is possible to obtain a reachable automaton A' recognizing the same trace language, removing from A all nonreachable local states and all transitions from nonreachable tuples of local states.

4. MINIMUM ASYNCHRONOUS AUTOMATA

In this section we introduce the notion of minimum asynchronous automaton, in the category of reachable asynchronous automata which

recognize a given trace language, and subsequently we exhibit a trace language that does not admit such an automaton.

Furthermore, we prove that every trace language defined on a concurrent alphabet with transitive dependency relation admits the minimum asynchronous automaton and we give a construction of such an automaton.

This section is organized as follows. First we define the concept of morphism for asynchronous automata, and we prove its properties; using the notion of morphism, we give a categorical definition of the minimum asynchronous automaton recognizing a given trace language. Next, we prove that there exist trace languages that do not admit a minimum asynchronous automaton. Finally, using these results, we characterize the class of concurrent alphabets on which every recognizable trace language admits a minimum asynchronous automaton as the class of concurrent alphabets with transitive dependency relation.

DEFINITION 4.1. Given two finite state asynchronous automata $A = \langle P_1, ..., P_n, \Delta, F \rangle$ and $A' = \langle P'_1, ..., P'_n, \Delta', F' \rangle$ over $\langle \Sigma, C \rangle$, a *morphism* φ *from* A *to* A' ($\varphi: A \to A'$) is a family of functions $\langle \varphi_i: S_i \to S'_i \rangle_{i=1,...,n}$ such that:

(1) φ preserves the initial states; i.e., for every i, $1 \leqslant i \leqslant n$, $\varphi_i(s_{i0}) = s'_{i0}$;

(2) φ preserves the transitions; i.e., for every $\sigma \in \Sigma$ with $\mathrm{Dom}(\sigma) = \{i_1, ..., i_k\}$ and for every reachable tuple $(s_{i_1}, ..., s_{i_k}) \in S_{i_1} \times \cdots \times S_{i_k}$ and $\forall j$, $1 \leqslant j \leqslant k$,

$$\varphi_{i_j}(\mathrm{Pro}_j[\delta_\sigma(s_{i_1}, ..., s_{i_k})]) = \mathrm{Pro}_j[\delta'_\sigma(\varphi_{i_1}(s_{i_1}), ..., \varphi_{i_k}(s_{i_k}))]$$

(where $\mathrm{Pro}_i(x_1, ..., x_k) = x_i$);[1]

(3) φ preserves the set of final states; i.e., for every reachable global state $(s_1, ..., s_n) \in \prod_{i=1}^n S_i$, $(s_1, ..., s_n) \in F$ if and only if $(\varphi(s_1), ..., \varphi(s_n)) \in F'$.

The following technical lemma is used in the proof of our main result:

LEMMA 4.1. *For every pair A and A' of reachable finite state asynchronous automata, if there exists a morphism $\varphi: A \to A'$, then this morphism is unique.*

Proof. Let $\psi: A \to A'$ be a morphism different from φ. Consider a reachable global state $(s_1, ..., s_n) \in \prod_{i=1}^n S_i$ and suppose that $\psi_i(s_i) = \varphi_i(s_i)$,

[1] Since the local transition functions are partial, the previous equality holds if and only if both $\delta_\sigma(s_{i_1}, ..., s_{i_k})$ and $\delta'_\sigma(\varphi_{i_1}(s_{i_1}), ..., \varphi_{i_k}(s_{i_k}))$ are defined and the left-hand side and the right-hand side coincide, or both $\delta_\sigma(s_{i_1}, ..., s_{i_k})$ and $\delta'_\sigma(\varphi_{i_1}(s_{i_1}), ..., \varphi_{i_k}(s_{i_k}))$ are not defined.

$i = 1, ..., n$. Then, for every $\sigma \in \Sigma$ with $\mathrm{Dom}(\sigma) = \{i_1, ..., i_k\}$, $(s_{i_1}, ..., s_{i_k})$ is a reachable tuple and so, for $j = 1, ..., k$, by Definition 4.1(2), it holds that

$$\begin{aligned}\varphi_{i_j}(\mathrm{Pro}_j[\delta_\sigma(s_{i_1}, ..., s_{i_k})]) &= \mathrm{Pro}_j[\delta'_\sigma(\varphi_{i_1}(s_{i_1}), ..., \varphi_{i_k}(s_{i_k}))] \\ &= \mathrm{Pro}_j[\delta'_\sigma(\psi_{i_1}(s_{i_1}), ..., \psi_{i_k}(s_{i_k}))] \\ &= \psi_{i_j}(\mathrm{Pro}_j[\delta_\sigma(s_{i_1}, ..., s_{i_k})]).\end{aligned}$$

Now it turns out, by assumption, that A and A' are reachable automata and, by Definition 4.1, that $\psi_i(s_{i0}) = \varphi_i(s_{i0}) = s'_{i0}$, $i = 1, ..., n$. Thus starting from $(s_{10}, ..., s_{n0})$ and iterating the previous argument it is not difficult to see that $\psi_i(s_i) = \varphi_i(s_i)$ for every $s_i \in S_i$, $i = 1, ..., n$, and so $\psi = \varphi$. ∎

In the following we consider the category AA_T of the reachable finite state asynchronous automata recognizing a given trace language T, with their morphisms.

We briefly recall that an object F in a category $\mathscr{C}$ is said to be *final* if and only if for every object A in $\mathscr{C}$ there exists exactly one morphism $\varphi: A \to F$; the final object of a category $\mathscr{C}$, if any, is unique up to isomorphism. Moreover, if every object in $\mathscr{C}$ is a reachable object then every morphism is surjective.

The following categorical definition of minimal automaton and of minimum automaton in a category $\mathscr{C}$ of reachable automata is from Ehrig *et al.* (1974).

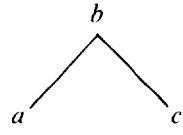
DEFINITION 4.2. An automaton A in a category $\mathscr{C}$ is called *minimal* or *reduced* if and only if, for every automaton A' in $\mathscr{C}$, every morphism $\varphi: A \to A'$ is an isomorphism. A is *minimum* if and only if A is the final object of $\mathscr{C}$.

From the previous definition, the *minimum finite state asynchronous automaton* recognizing a trace language T, if any, is the final object in AA_T. Since the final object is unique, the minimum asynchronous automaton recognizing T is unique.

It is known that for every recognizable trace language T, there exists a unique (up to isomorphism) minimum $F(\Sigma, C)$-automaton accepting T (Nerode, 1958). In Theorem 4.1 we show that this result, in general, is not true for asynchronous automata. In fact, we exhibit a trace language that *does not admit* a minimum asynchronous automaton.

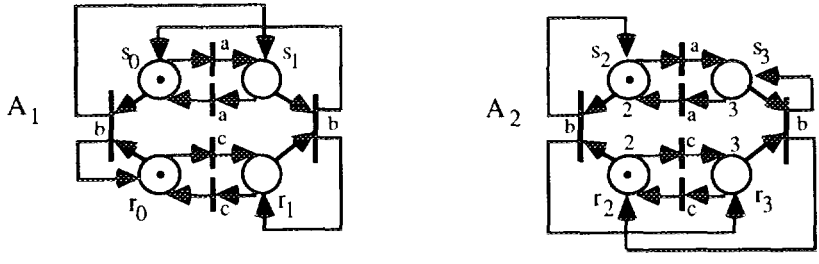
THEOREM 4.1. *Let* $\langle \Sigma, C \rangle$ *be the concurrent alphabet such that* $\Sigma = \{a, b, c\}$, $C = \{(a, c), (c, a)\}$. *Then the recognizable trace language* $T = [\{\{a, b, c\}\{a, c\}\}^*]_C$ *does not admit a minimum finite state asynchronous automaton; i.e., the category* AA_T *does not contain a final object.*

Proof. First, we observe that the dependency graph has the form



so the maximal cliques of the dependency relation, i.e., the local alphabets of asynchronous automata with concurrent alphabet $\langle \Sigma, C \rangle$, are $\{a, b\}$ and $\{b, c\}$.

Let us consider the following finite state asynchronous automata A_1 and A_2 over $\langle \Sigma, C \rangle$ (represented as Petri nets), with sets of final states $F_1 = \{(s_0, r_0), (s_1, r_1)\}$ and $F_2 = \{(s_2, r_2), (s_3, r_3)\}$, respectively:



It is not difficult to verify that both automata A_1 and A_2 recognize the trace language $T = [\{\{a, b, c\}\{a, c\}\}^*]_C$.

To prove the theorem we show that A_1 and A_2 are minimal non isomorphic automata in AA_T, and so the final object in AA_T cannot exist.

To achieve this goal, it is useful to show that every morphism φ from A_1 to another asynchronous automaton A' is an isomorphism.

In fact, suppose that $\varphi_1(s_0) = \varphi_1(s_1)$; then $(\varphi_1(s_0), \varphi_2(r_0)) = (\varphi_1(s_1), \varphi_2(r_0))$. But since $(s_0, r_0) \in F_1$, $(s_1, r_0) \notin F_1$ and, by Definition 4.1(3), it should hold that $(\varphi_1(s_0), \varphi_2(r_0)) \in F'$ and $(\varphi_1(s_1), \varphi_2(r_0)) \notin F'$. At this point we have a contradiction. Then $\varphi_1(s_0) \neq \varphi_1(s_1)$ and, in a similar way, $\varphi_2(r_0) \neq \varphi_2(r_1)$. So, recalling that φ is surjective, it follows that φ is an isomorphism and consequently A_1 is minimal in AA_T.

Using similar arguments we can verify that also A_2 is minimal in AA_T.

Now, we show that A_1 and A_2 are non isomorphic. Suppose that there exists a morphism $\psi: A_1 \to A_2$. Then, by Definition 4.1(1), $\psi_1(s_0) = s_2$ and $\psi_2(r_0) = r_2$ and, by Definition 4.1(2), the following equalities hold:

$$\psi_1(s_1) = \psi_1[\mathrm{Pro}_1(s_1, r_0)] = \psi_1[\mathrm{Pro}_1(\delta_{1b}(s_0, r_0))]$$
$$= \mathrm{Pro}_1[\delta_{2b}(\psi_1(s_0), \psi_2(r_0))]$$
$$= \mathrm{Pro}_1[\delta_{2b}(s_2, r_2)] = \mathrm{Pro}_1(s_2, r_3) = s_2 = \psi_1(s_0).$$

Then $\psi_1(s_1) = \psi_1(s_0)$, but, as observed above, for every morphism φ from A_1 to another asynchronous automaton A', it holds that $\varphi_1(s) \neq \varphi_1(s_1)$. So a morphism $\psi: A_1 \to A_2$ cannot exist. Thus the category AA_T contains two minimal non isomorphic objects A_1 and A_2, and the final object does not exist. ∎

While the previous theorem shows that in general it is not possible to find a minimum asynchronous automaton recognizing a given trace language, for the class of trace languages defined on alphabets with transitive dependency relation, such automaton always exists. The proof of this fact and the construction of the minimum automaton is given in the following theorem.

THEOREM 4.2. *Let $\langle \Sigma, C \rangle$ be a concurrent alphabet with transitive dependency relation. Then every recognizable trace language $T \subseteq F(\Sigma, C)$ admits a minimum finite state asynchronous automaton; i.e., the category AA_T contains a final object.*

Proof. Given $\langle \Sigma, C \rangle$ with C^c transitive, let $\Sigma_1, \ldots, \Sigma_n$ be the maximal cliques of C^c. Since $\Sigma_i \cap \Sigma_k = \varnothing$ for $i \neq k$, $F(\Sigma, C)$ is isomorphic to $\Sigma_1^* \times \cdots \times \Sigma_n^*$.

For every recognizable trace language $T \subseteq F(\Sigma, C)$ let R be the relation

$$\forall x, y \in F(\Sigma, C), \qquad xRy \text{ iff } (\forall z, xz \in T \text{ iff } yz \in T),$$

and for every $F(\Sigma, C)$-automaton $A = \langle Q, q_0, \delta, F \rangle$, let R_A be the relation

$$\forall x, y \in F(\Sigma, C), \qquad xR_A y \text{ iff } \delta(q_0, x) = \delta(q_0, y).$$

The following facts are known by results on Nerode equivalence (Nerode, 1958):

— R and R_A are right invariant equivalence relations;

— the set of the states of the minimum $F(\Sigma, C)$-automaton M recognizing T is isomorphic to the set of equivalence classes of R and $R = R_M$;

— A recognizes T if and only if the equivalence relation R_A is a refinement of R.

Now, let R_i, $i=1, ..., n$, be the restriction of R to $\Sigma_i^* \times \Sigma_i^*$; that is,

$$\forall x, y \in \Sigma_i^*, \qquad x R_i y \text{ iff } xRy.$$

Also, R_i, $i=1, ..., n$, is a right invariant equivalence relation.

Consider the finite state asynchronous automaton $A = \langle P_1, ..., P_n, \Delta, F \rangle$ defined as follows:

— $\forall i,\ 1 \leqslant i \leqslant n,\ P_i = \langle \Sigma_i, S_i, s_{i0} \rangle$,

— S_i is the set of equivalence class of R_i and $[x]_i$ is the element of S_i containing x;

— $s_{i0} = [\varepsilon]_i$;

— $\forall x \in \Sigma_1^*$ and $\forall \sigma \in \Sigma_i$, $\delta_\sigma([x]_i) = [x\sigma]_i$ (the definition is consistent since R_i is right invariant);

— $F = \{(s_1, ..., s_n) \in S_1 \times \cdots \times S_n \mid \exists t \in T(s_{10}, ..., s_{n0}) \overset{t}{\Longrightarrow} (s_1, ..., s_n)\}$.

Now, we prove that A is the minimum asynchronous automaton accepting T. First, we show that the relation R_A induced by the sequential version of A is a refinement of R; thus we can conclude that A recognizes T. Subsequently, we exhibit, for every asynchronous automaton A' recognizing T, a morphism from A' to A.

Let t, t' be two traces such that $(s_{10}, ..., s_{n0}) \overset{t}{\Longrightarrow} (s_1, ..., s_n)$ and $(s_{10}, ..., s_{n0}) \overset{t'}{\Longrightarrow} (s_1, ..., s_n)$ for some $(s_1, ..., s_n) \in S_1 \times \cdots \times S_n$. Then $t = t_1 \cdots t_n$, $t' = t'_1 \cdots t'_n$, where $t_i, t'_i \in \Sigma_i^*$, for $i=1, ..., n$.

For $x = t_i, t'_i$

$$(s_{10}, ..., s_{n0}) \overset{x}{\Longrightarrow} (s_{10}, ..., s_{i-10}, s_i, s_{i+10}, ..., s_{n0}),$$

and so $t_i R t'_i$, $t_i R_i t'_i$. If $k \leqslant n$ and $(t_1 \cdots t_{k-1}) R (t'_1 \cdots t'_{k-1})$ then, for the right invariancy of R,

$$(t_1 \cdots t_{k-1})\, t_k R (t'_1 \cdots t'_{k-1})\, t_k,\ t_k (t'_1 \cdots t'_{k-1})\, R t'_k (t'_1 \cdots t'_{k-1}).$$

Since for $i \neq j$ traces t_i, t'_i commute with t_j and t'_j, it holds that

$$(t'_1 \cdots t'_{k-1})\, t_k = t_k (t'_1 \cdots t'_{k-1}) \qquad \text{and} \qquad t'_k (t'_1 \cdots t'_{k-1})\, R (t'_1 \cdots t'_{k-1})\, t'_k.$$

Using the transitivity we obtain $(t_1 \cdots t_{k-1} t_k)\, R (t'_1 \cdots t'_{k-1} t'_k)$. For $k=n$, we can conclude that tRt'; so the relation R_A is a refinement of R and A recognizes T.

Let $A' = \langle P'_1, ..., P'_n, \Delta', F' \rangle$, with $P'_i = \langle \Sigma_i, S'_i, s'_{i0} \rangle$ for $i=1, ..., n$, be another reachable finite state asynchronous automaton recognizing the language T.

Since A' is reachable, then for every $s \in S_i'$ there exists a string $x \in \Sigma_i^*$ such that

$$\delta'((s'_{10}, ..., s'_{i-10}, s'_{i0}, s'_{i+10}, ..., s'_{n0}), x) = (s'_{10}, ..., s'_{i-10}, s, s'_{i+10}, ..., s'_{n0}).$$

For every string $y \in \Sigma_i^*$ such that

$$\delta'((s'_{10}, ..., s'_{i-10}, s'_{i0}, s'_{i+10}, ..., s'_{n0}), y) = (s'_{10}, ..., s'_{i-10}, s, s'_{i+10}, ..., s'_{n0}),$$

it holds that $xR_{A'}y$, and, R_A being a refinement of R, xRy, $[x]_i = [y]_i$. Defining $\varphi_i(s) = [x]_i$, it holds that $\varphi_i(\delta'_\sigma(s)) = [x\sigma]_i = \delta_\sigma([x]_i) = \delta_\sigma(\varphi_i(s))$ and $\varphi_i(s'_{i0}) = s_{i0}$.

Moreover, let $(s_1, ..., s_n)$ be a global state of A'. Then there exists a trace $t = t_1 \cdots t_n$, $t_i \in \Sigma_i^*$, such that $\delta'((s'_{10}, ..., s'_{n0}), t) = (s_1, ..., s_n)$ in the automaton A', and so, $(s_{10}, ..., s_{n0}) \overset{t}{\Longrightarrow} (\varphi_1(s_1), ..., \varphi_n(s_n))$ in the automaton A, by definition of φ_i. Hence, $(s_1, ..., s_n) \in F'$ if and only if $(\varphi_1(s_1), ..., \varphi_n(s_n)) \in F$.

Then the family of functions $\langle \varphi_i : S_i' \to S_i \rangle_{1 \leq i \leq n}$ is a morphism between A' and A. By Lemma 4.1 this morphism is unique; so A is the final object in the category AA_T. ∎

Using Theorems 4.1 and 4.2, we characterize the concurrent alphabets $\langle \Sigma, C \rangle$ such that every recognizable trace language $T \subseteq F(\Sigma, C)$ admits the minimum asynchronous automaton.

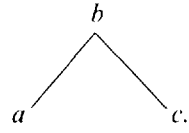
COROLLARY 4.1. *Let $\langle \Sigma, C \rangle$ be a concurrent alphabet. Then the following sentences are equivalent*:

(1) *Every recognizable trace language $T \subseteq F(\Sigma, C)$ admits a unique (up to isomorphism) minimum finite state asynchronous automaton.*

(2) *The dependency relation C^c is transitive.*

Proof. By Theorem 4.2, (2) implies (1).

For proving the converse, note that if C^c is not transitive then there is a subgraph of the dependency graph of the form

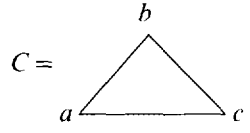


Thus by Theorem 4.1, there exists a language $T \subseteq F(\Sigma, C)$ that does not admit the minimum asynchronous automaton. ∎

5. Equivalence Problem for Unambiguous Regular Trace Languages

As anticipated in the Introduction, starting with this section we abandon the study of finite state asynchronous automata for the study of some decision problems related to regular trace languages.

Given two distinct regular expressions it may happen that they define the same trace language; this is the case for example of the regular expressions $(acb)^*$ and $(cab)^*$ defined on the concurrent alphabet $\langle \Sigma, C \rangle$ with $\Sigma = \{a, b, c\}$ and the concurrency relation



It becomes important then to have algorithms which can establish, given two regular expressions on the same concurrent alphabet, whether they define the same trace language. Bertoni *et al.* (1982) have shown that this problem, know as the *equivalence problem for regular trace languages*, is in its general form *undecidable*.

However, in Bertoni *et al.* (1982), it is also shown that if we restrict our attention to some particular subclasses of regular languages the equivalence problem turns out to be decidable. More precisely, Bertoni *et al.* (1982) have shown that the equivalence problem for regular trace languages defined over concurrent alphabets with a transitive concurrency relation is decidable. Subsequently Aalbersberg and Welzl (1986) have extended this result to universally regular trace languages, and Aalbersberg and Hoogeboom (1987) have shown that the equivalence problem for regular trace languages defined over a concurrent alphabet $\langle \Sigma, C \rangle$ is decidable *if and only if* the concurrency relation C is transitive.

In this section we deepen the analysis on the decidability of the equivalence problem restricting our considerations to unambiguous regular trace languages introduced and studied by Bertoni *et al.* (1985) and Sakarovitch (1987). We show that there exists a concurrent alphabet with a *non* transitive concurrency relation for which the equivalence problem for unambiguous regular trace languages can be decided in polynomial time by a deterministic Turing machine. (Note that the same problem for regular languages is undecidable in this monoid.)

We briefly recall that given two decision problems π and π', π is *polynomial time many one reducible* to π' if there exists a polynomial time computable function f which to each instance y of π associates an instance $f(y)$ of π' such that y is a positive instance of π *if and only if* $f(y)$ is a

positive instance of π'. Further, if π' is decidable in polynomial time then also π is decidable in polynomial time (see Garey and Johnson, 1979, for further details).

In order to prove this result we consider the following decision problems:

Equivalence Problem for $\mathbf{UR}(\Sigma, C)$.

Instance: Two deterministic Σ^*-automata A_1 and A_2, s.t. for every $t \in F(\Sigma, C)$ it holds that $\#(\text{lin}(t) \cap L(A_1)) \leqslant 1$ and $\#(\text{lin}(t) \cap L(A_2)) \leqslant 1$.

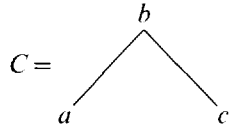
Question: $[L(A_1)]_C = [L(A_2)]_C$?

Rational Formal Series Emptiness.

Instance: A finite representation for a rational formal series φ.

Question: Is φ the null function?

We show that when $\Sigma = \{a, b, c\}$ and



the Equivalence Problem for $\mathbf{UR}(\Sigma, C)$ is polynomial time many one reducible to Rational Formal Series Emptiness, a problem decidable in polynomial time (see Berstel and Reutenauer, 1988, for definitions and properties of rational power series).

For proving our main result we also use the following representation (Salomaa and Soittola, 1978) of M-automata. Given an M-automaton $A = \langle Q, q_0, \delta, F \rangle$ with $Q = \{q_1, ..., q_n\}$, we consider a tuple $\langle \pi, \{A_m\}_{m \in M}, \eta \rangle$, where

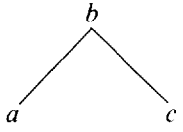
- π is a unitary n-dimensional row vector (where a vector is unitary iff its elements belong to $\{0, 1\}$, and exactly one element is equal to 1);
- for every $m \in M$, A_m is an $n \times n$-dimensional matrix whose rows are unitary vectors, such that A_1 is the identity matrix and $A_{mm'} = A_m A_{m'}$ for $m, m' \in M$;
- η is an n-dimensional column vector on $\{0, 1\}$.

Then matrix A_m can be interpretated as the adjacency matrix of the function $\delta(-, m)$, i.e., $A_m(i, j) = 1$ iff $\delta(q_i, m) = q_j$; the vector π as the characteristic vector of the initial state, i.e., $\pi(i) = 1$ iff $q_i = q_0$; and the vector η as the characteristic vector of final states, i.e., $\eta(i) = 1$ iff $i \in F$. So for every $m \in M$, the following relation holds:

$$\pi \cdot A_m \cdot \eta = 1 \qquad \text{if and only if } \delta(q_0, m) \in F.$$

Then the characteristic function of the language recognized by the automaton A can be rewritten as $\pi \cdot A_m \cdot \eta$.

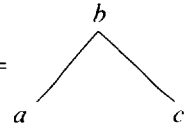
LEMMA 5.1. *Let $\langle \Sigma, C \rangle$ be a concurrent alphabet where $\Sigma = \{a, b, c\}$ and*

$C =$ 

Then the Equivalence Problem for $\mathbf{UR}(\Sigma, C)$ is many one reducible to Rational Formal Series Emptiness.

Proof. The proof is split in two parts. In the first part we show how to reduce the Equivalence Problem for $\mathbf{UR}(\Sigma, C)$ to the problem of deciding emptiness for a generic formal series G; subsequently we prove that G is in effect a rational formal series.

Part I. Given $\Sigma = \{a, b, c\}$ and

$C =$ 

every trace $t \in F(\Sigma, C)$ can be denoted by $[xb^n]_C$ with $x \in \{a, c\}^*$, and it contains exactly all strings of the form $b^{n_0} y_1 b^{n_1} \cdots y_k b^{n_k}$ where $k = |x|$, $n_i \geqslant 0$, $1 \leqslant i \leqslant k$, $n_0 + \cdots + n_k = n$, $y_i \in \{a, c\}$, $1 \leqslant i \leqslant k$, and $y_1 \cdots y_k = x$. Let $T \subseteq F(\Sigma, C)$ be an unambiguous regular trace language and consider for each $x \in \{a, c\}^*$ the generating function $f_T(x, z)$ in the complex variable z defined as follows:

$$f_T(x, z) = \sum_{n=0}^{\infty} \chi_T([xb^n]_C)\, z^n$$

where χ_T is the characteristic function of T.

By Theorem 2.2, given T, there exists a regular language $L \subseteq \sum^*$ such that $T = [L]_C$ and $\forall t \in T$, $\#(\mathrm{lin}(t) \cap L) = 1$. Thus, if we denote by χ_L the characteristic function of L and by $A = \langle \pi, A_a, A_b, A_c, \eta \rangle$ an automaton accepting L, we can rewrite the generating function $f_T(x, z)$ in the following way:

$$f_T(x, z) = \sum_{n=0}^{\infty} \chi_T([xb^n]_C)\, z^n = \sum_{n=0}^{\infty} \left(\sum_{[\alpha]_C = [xb^n]_C} \chi_L(\alpha) \right) z^n$$

$$= \sum_{n_0 \cdots n_k} \chi_L(b^{n_0} y_1 b^{n_1} y_2 \cdots b^{n_{k-1}} y_k b^{n_k})\, z^{n_0 + n_1 + \cdots + n_k}$$

$$= \pi \left[\sum_{n_0, \ldots, n_k} (A_b \cdot z)^{n_0} (A_{y_1}) \cdots (A_b \cdot z)^{n_{k-1}} (A_{y_k}) (A_b \cdot z)^{n_k} \right] \eta$$

$$= \pi \left[\sum_{n_0} (A_b \cdot z)^{n_0} (A_{y_1}) \cdots \sum_{n_{k-1}} (A_b \cdot z)^{n_{k-1}} (A_{y_k}) \sum_{n_k} (A_b \cdot z)^{n_k} \right] \eta$$

$$= \pi \hat{A}_{y_1} \cdots \hat{A}_{y_k} \hat{\eta}.$$

If we denote by I the identity matrix, the matrices $\hat{A}_{y_i}$ and $\hat{\eta}$ can be written in the following way:

$$\hat{A}_{y_i} = \sum_{n_i} (A_b \cdot z)^{n_i} (A_{y_i}) = \frac{1}{(I - A_b z)} A_{y_i}$$

and

$$\hat{\eta} = \sum_{n_k} (A_b \cdot z)^{n_k} \eta = \frac{1}{(I - A_b z)} \eta.$$

We observe that $\hat{A}_{y_i}$ and $\hat{\eta}$ are matrices whose components are rational functions in z.

Now consider two unambiguous regular trace languages T_1 and T_2 and the following function in two complex variables z, y:

$$G_{T_1 T_2}(z, y) = \sum_{x \in \{a, c\}^*} (f_{T_1}(x, z) - f_{T_2}(x, z))^2\, y^{|x|}.$$

It is easy to verify that with respect to $G_{T_1 T_2}(z, y)$ the following properties hold:

$$T_1 = T_2 \text{ iff } G_{T_1 T_2}(z, y) = 0 \text{ iff } (\forall x \in \{a, c\}^*, f_{T_1}(x, z) = f_{T_2}(x, z)).$$

Thus, for the concurrent alphabet $\langle \Sigma, C \rangle$ just considered, we have reduced via many one reduction the Equivalence Problem for $\mathbf{UR}(\Sigma, C)$ to the problem of verifying whether $G_{T_1 T_2}(z, y) = 0$. We now show that $G_{T_1 T_2}(z, y)$ is rational.

Part II. To accomplish this step of the proof we must first introduce the definition and some properties of the Kronecker, or tensor, product between matrices.

Let $A(m\times n)=(a_{ij})$ and $B(p\times q)=(b_{ij})$ be two matrices. Then the Kronecker product of A and B, denoted by $A\otimes B$, is defined as

$$A\otimes B=(a_{ij}B),$$

or equivalently it is an $(mp\times nq)$ matrix defined as follows:

$$A\otimes B=\begin{bmatrix} a_{11}B & \cdots & a_{1n}B \\ \vdots & \vdots & \vdots \\ a_{m1}B & \cdots & a_{mn}B \end{bmatrix}.$$

In this paper we are interested in the property of the tensor product of matrices

$$(A\otimes B)\cdot(C\otimes D)=(A\cdot C)\otimes(B\cdot D),$$

where by the symbol $\cdot$ we denote the usual matrix product. We also denote by $A^{(2)}$ the term $A\otimes A$.

Now, consider two unambiguous regular trace languages T_1, T_2. Let $A_1=\langle \pi_1, A_{1a}, A_{1b}, A_{1c}, \eta_1\rangle$, $A_2=\langle \pi_2, A_{2a}, A_{2b}, A_{2c}, \eta_2\rangle$ be the automata recognizing the related languages L_1 and L_2 and $G_{T_1T_2}(z, y)$ the related generating function as previously defined, i.e.,

$$G_{T_1T_2}(z, y)=\sum_{x\in\{a,c\}^*}(f_{T_1}(x, z)-f_{T_2}(x, z))^2\, y^{|x|}.$$

Then the term $(f_{T_1}(x, z)-f_{T_2}(x, z))^2$, given $x=y_1\cdots y_k$, can be rewritten as follows:

$$\begin{aligned}
&(f_{T_1}(x, z)-f_{T_2}(x, z))^2 \\
&\quad=(\pi_1\hat{A}_{1_{y_1}}\cdots\hat{A}_{1_{y_k}}\hat{\eta}_1-\pi_2\hat{A}_{2_{y_1}}\cdots\hat{A}_{2_{y_k}}\hat{\eta}_2)^2 \\
&\quad=\left([\pi_1\pi_2]\begin{bmatrix}\hat{A}_{1_{y_1}} & 0\\ 0 & \hat{A}_{2_{y_1}}\end{bmatrix}\cdots\begin{bmatrix}\hat{A}_{1_{y_k}} & 0\\ 0 & \hat{A}_{2_{y_k}}\end{bmatrix}\begin{bmatrix}\hat{\eta}_1\\ -\hat{\eta}_2\end{bmatrix}\right)^2 \\
&\quad=\left([\pi_1\pi_2]\begin{bmatrix}\hat{A}_{1_{y_1}} & 0\\ 0 & \hat{A}_{2_{y_1}}\end{bmatrix}\cdots\begin{bmatrix}\hat{A}_{1_{y_k}} & 0\\ 0 & \hat{A}_{2_{y_k}}\end{bmatrix}\begin{bmatrix}\hat{\eta}_1\\ -\hat{\eta}_2\end{bmatrix}\right)^{(2)}
\end{aligned}$$

the last step follows from the observation that the left term of the above identity represents a matrix with just one component so that, in this particular case, the usual matrix product coincide with the Kronecker product. Now applying the property of tensor product, above cited, to the right term of the above identity we obtain

$$[\pi_1\pi_2]^{(2)}\begin{bmatrix}\hat{A}_{1_{y_1}} & 0\\ 0 & \hat{A}_{2_{y_1}}\end{bmatrix}^{(2)}\cdots\begin{bmatrix}\hat{A}_{1_{y_k}} & 0\\ 0 & \hat{A}_{2_{y_k}}\end{bmatrix}^{(2)}\begin{bmatrix}\hat{\eta}_1\\ -\hat{\eta}_2\end{bmatrix}^{(2)}.$$

Setting $y_i = a, c$ and

$$C_{y_i} = \begin{bmatrix} \hat{A}_{1_{y_i}} & 0 \\ 0 & \hat{A}_{2_{y_i}} \end{bmatrix}^{(2)}, \qquad \pi'[\pi_1 \pi_2] \otimes [\pi_1 \pi_2], \qquad \eta' = \begin{bmatrix} \hat{\eta}_1 \\ -\hat{\eta}_2 \end{bmatrix} \otimes \begin{bmatrix} \hat{\eta}_1 \\ -\hat{\eta}_2 \end{bmatrix},$$

the function $G_{T_1 T_2}(z, y)$ becomes

$$\begin{aligned} G_{T_1 T_2}(z, y) &= \sum_{k \geq 0} \sum_{y_1 \cdots y_k \in \{a,c\}^*} (\pi' C_{y_1} \cdots C_{y_k} \eta')\, y^k \\ &= \pi' \left(\sum_{k \geq 0} \sum_{y_1 \cdots y_k \in \{a,c\}^*} (C_{y_1} \cdots C_{y_k})\, y^k \right) \eta' \\ &= \pi' \left(\sum_{k \geq 0} (C_a + C_c)^k\, y^k \right) \eta' = \pi' \frac{1}{(I - (C_a + C_c)\, y)} \eta'. \end{aligned}$$

At this point we can observe that $G_{T_1 T_2}(z, y)$ is a rational function in z, y so we have reduced the Equivalence Problem for $\mathbf{UR}(\Sigma, C)$ to Rational Formal Series Emptiness. ∎

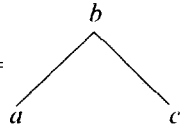
LEMMA 5.2. *The reduction showed in Lemma* 5.1 *can be computed by a deterministic Turing machine in polynomial time.*

Proof. The reduction presented in Lemma 5.1 can be computed by an algorithm which given as inputs two automata, $A_1 = \langle \pi_1, A_{1a}, A_{1b}, A_{1c}, \eta_1 \rangle$ and $A_2 = \langle \pi_2, A_{2a}, A_{2b}, A_{2c}, \eta_2 \rangle$, computes the explicit form of the function

$$G_{T_1 T_2}(z, y) = \pi' \frac{1}{(I - C_a + C_c)\, y)} \eta'.$$

The most expensive operation required for computing such an explicit form is the computation of the inverse of the multivariable polynomial matrix $(I - (C_a + C_c)\, y)$. We recall that C_a and C_c are functions of z. A method for realizing such a computation in polynomial time is suggested by Krishnamurty (1985). ∎

THEOREM 5.1. *Given* $\Sigma = \{a, b, c\}$ *and*

$C =$ 

the Equivalence Problem for $\mathbf{UR}(\Sigma, C)$ *is decidable in polynomial time.*

6. On Regular Expressions Expressiveness

The result presented in Section 5 stresses the importance of having algorithms to distinguish whether or not a given regular expression defines an unambiguous regular trace language. In this section we study the computational difficulty of the following decision problem and we prove that generally it is undecidable, by proving that it is undecidable for the concurrent alphabet considered in Theorem 5.1.

Regular Expressions for $\mathbf{UR}(\Sigma, C)$.

Instance: a regular expression r on $\langle \Sigma, C \rangle$.

Question: does r denote a language in $\mathbf{UR}(\Sigma, C)$?

To obtaining this result we prove that the following decision problem,

Regular Expressions Fullness for $\mathbf{F}(\Sigma, C)$.

Instance: a regular expression r on $\langle \Sigma, C \rangle$

Question: does r denote the string language $[L]_C = F(\Sigma, C)$?

which has been proved undecidable by Aalbersberg and Hoogeboom (1987), for all trace monoids with concurrency relation C not transitive, is Turing reducible to Regular Expression for $\mathbf{UR}(\Sigma, C)$, when

$$\langle \Sigma, C \rangle = \left\langle \{a, b, c\}, \begin{array}{ccc} & b & \\ a & & c \end{array} \right\rangle.$$

We briefly recall that given two problems π and π', π is Turing reducible to π' if there exists an algorithm that solves π using a hypothetical subroutine for solving π'. It is not difficult to see that if π is undecidable and it is Turing reducible to a problem π' then also π' has to be undecidable.

Lemma 6.1. *For every concurrent alphabet* $\langle \Sigma, C \rangle$ *for which the Equivalence Problem for* $\mathbf{UR}(\Sigma, C)$ *is decidable, the problem Regular Expressions Fullness for* $F(\Sigma, C)$ *is Turing reducible to Regular Expressions for* $\mathbf{UR}(\Sigma, C)$.

Proof. First of all, we recall that given a regular expression r, and where the string language denoted by r and the trace language denoted by r are called L and T, respectively, $[L]_C = T$. For every regular expression r over $\langle \Sigma, C \rangle$, the following algorithm, given a subroutine for deciding

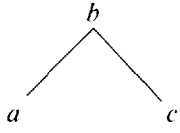
Regular Expressions for $\mathbf{UR}(\Sigma, C)$, can solve the problem Regular Expressions for $F(\Sigma, C)$:

if r denotes a language $T \notin \mathbf{UR}(\Sigma, C)$
then $T \neq F(\Sigma, C)$, since $F(\Sigma, C)$ is unambiguous;
else let r denote a language T s.t. $T \in \mathbf{UR}(\Sigma, C)$.

Since for Theorem 5.1 the Equivalence Problem for $\mathbf{UR}(\Sigma, C)$ is decidable, it is possible to decide if $T = F(\Sigma, C)$. ▮

THEOREM 6.1. *Given* $\Sigma = \{a, b, c\}$ *and*

$C =$



the problem Regular Expression for $\mathbf{UR}(\Sigma, C)$ *is recursively undecidable.*

7. RECENT DEVELOPMENTS

After this paper was been submitted for publication, a paper by Harju and Karhuämaki (1991) appeared containing results which make it possible, using techniques completely different from ours, to generalize Theorem 5.1 and Theorem 6.1. More precisely, the following theorem can be obtained:

THEOREM 7.1. *The Equivalence Problem for* $\mathbf{UR}(\Sigma, C)$ *is decidable for every concurrent alphabet* $\langle \Sigma, C \rangle$.

This result is also contained in Varricchio (1991).

Referring to Theorem 2.1, Theorem 7.1, and Lemma 6.1, the following theorem can be proved:

THEOREM 7.2. *The problem Regular Expression for* $\mathbf{UR}(\Sigma, C)$ *is undecidable if and only if* C *is not transitive.*

ACKNOWLEDGMENTS

We are indebted to Professor Alberto Bertoni for valuable advice and encouragement and for extremely useful comments about the paper's earlier draft. Thanks also to the referees for many valuable suggestions which contributed to improve the exposition. In particular, Section 7 and the results contained in it were suggested to the authors by one of the referees.

RECEIVED May 18, 1990; FINAL MANUSCRIPT RECEIVED September 27, 1991

REFERENCES

AALBERSBERG, IJ. AND HOOGEBOOM, H. (1987), Decision problems for regular trace languages, *in* "Proceedings, 14th ICALP," pp. 251–259, Lecture Notes in Computer Science, Vol. 267, Springer-Verlag, Berlin/New York.

AALBERSBERG, IJ. AND ROZENBERG, G. (1988), Theory of traces, *Theoret. Comput. Sci.* **60**, pp. 1–82.

AALBERSBERG, IJ. AND WELZL, E. (1986), Trace languages defined by regular string languages, *RAIRO Inform. Théor. Appl.* **20**, pp. 103–119.

BERSTEL, J. AND REUTENAUER, C. (1988), "Rational Series and Their Languages," Springer-Verlag, Berlin.

BERTONI, A., BRAMBILLA, M., MAURI, G., AND SABADINI, N. (1981), An application of the theory of free partially commutative monoids: Asymptotic densities of trace languages, *in* "Proceedings, 10th Symposium MFCS," pp. 205–215, Lecture Notes in Computer Science, Vol. 118, Springer-Verlag, Berlin/New York.

BERTONI, A., MAURI, G., AND SABADINI, N. (1982), Equivalence and membership problems for regular trace languages, *in* "Proceedings, 9th ICALP," pp. 61–71, Lecture Notes in Computer Science, Vol. 140, Springer-Verlag, Berlin/New York.

BERTONI, A., MAURI, G., AND SABADINI, N. (1985), Unambiguous regular trace languages, *in* "Algebra, Combinatorics, and Logic in Computer Science," pp. 113–123, Colloquia Mathematica Societatis János Bolyai, Vol. 42, North-Holland, Amsterdam.

DUBOC, C. (1986), "Commutations dans les monoides libres: Une cadre theorique pour l'étude du parallélisme," Thèse de Doctorat, Univesité du Rouen.

EILENBERG, S. AND SCHÜTZENBERGER, M. (1968), Rational sets in commutative monoids, *J. Algebra* **13**, 173–191.

EHRIG, H., KIERMEIER, K., KREOWSKI, H., AND KÜHNEL, W. (1974), "Universal Theory of Automata," Teubner, Stuttgart.

GAREY, M. AND JOHNSON, D. (1979), "Computers and Intractability," Freeman, San Francisco.

HARJU, T. AND KARHUÄMAKI, J. (1991), The equivalence problem of multitape finite automata, *Theoret. Comput. Sci.* **78**, 347–355.

KRISHNAMURTHY, E. V. (1985), "Error-Free Polynomial Matrix Computations," Springer-Verlag, Berlin/New York.

MAZURKIEWICZ, A. (1977), "Concurrent Program Schemes and Their Interpretations," DAIMI Report PB-78, Aarhus University.

NERODE, A. (1958), Linear automaton transformations, *Proc. Amer. Math. Soc.* **9**, 541–544.

SAKAROVITCH, J. (1987), On regular trace languages, *Theoret. Comput. Sci.* **52**, 59–75.

SALOMAA, A. AND SOITTOLA, M. (1978), "Automata Theoretic Aspects of Formal Power Series," Springer-Verlag, Berlin/New York.

SZIJARTO, M. (1981), A classification and closure properties of languages for descibing concurrent systems behaviours, *Fund. Inform.* **4**, 531–549.

VARRICCHIO, S. (1991), On the decidability of the equivalence problem for partially commutative rational power series, manuscript.

ZIELONKA, W. (1987), Notes on finite asynchronous automata, *RAIRO Inform. Theor. Appl.* **21** (2), 99–135. [first appeared as Technical Report, Institute of Mathematics, Warsaw Technical University, 1984]