Theoretical Computer Science 132 (1994) 179–207
Elsevier

Asynchronous automata versus asynchronous cellular automata*

Giovanni Pighizzini

Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, via Comelico 39, I-20135 Milano, Italy

Communicated by M. Nivat
Received September 1992
Revised September 1993

Abstract

Pighizzini G., Asynchronous automata versus asynchronous cellular automata, Theoretical Computer Science, 132 (1994) 179–207.

In this paper we compare and study some properties of two mathematical models of concurrent systems, *asynchronous automata* (Zielonka, 1987) and *asynchronous cellular automata* (Zielonka, 1989). First, we show that these models are "polynomially" related, exhibiting polynomial-time reductions between them. Subsequently, we prove that, in spite of that, the classes of asynchronous automata and of asynchronous cellular automata recognizing a given trace language are, in general, deeply different. In fact, we exhibit a recognizable trace language T with the following properties: there exists a unique minimum asynchronous automaton accepting T, does not exist a unique minimum asynchronous cellular automaton, but there are infinitely many minimal (i.e., unreducible) nonisomorphic asynchronous cellular automata accepting T. We characterize the class of concurrent alphabets for which every recognizable trace language admits a minimum finite state asynchronous cellular automaton as the class of alphabets with full concurrency relation. Finally, extending a result of (Bruschi et al., 1988), we show that for every concurrent alphabet with nontransitive dependency relation, there exists a trace language accepted by infinitely many minimal nonisomorphic asynchronous automata.

1. Introduction

Trace languages were introduced by Mazurkiewicz in 1977 [21, 22] in order to give a noninterleaving semantic of concurrent systems. In Mazurkiewicz's approach, the

Correspondence to: G. Pighizzini, Dipartimento di Scienze dell'Informazione, Università degli Studi di Milano, via Comelico 39, I-20135 Milano, Italy. Email: pighizzi@ghost.dsi.unimi.it.

* Supported in part by the ESPRIT Basic Research Action No. 3166: "Algebraic and Syntactic Methods in Computer Science (ASMICS)" and by MURST.

SSDI 0304-3975(93)E0161-V

structure of a system is described by a concurrent alphabet, that is by a finite set of actions (i.e., an alphabet) and by a binary relation over this set (i.e., a concurrency relation). This relation, used to specify those pairs of actions can be concurrently executed, permits to identify different sequential observations of the same behavior. In this way, a process is described by an equivalence class of strings. This class, called *trace*, can also be represented by a partially ordered set of actions.

It is possible to observe that traces are elements of free partially commutative monoids, algebraic structures introduced by Cartier and Foata [7] with combinatorial motivations. Using this fact, a theory of *trace languages* (i.e., subsets of free partially commutative monoids) has been developed as an extension of the classical theory of formal languages, as witnessed by many papers (see [1] for a review of many results in trace theory and for an annotated bibliography).

An interesting subject in trace theory is that of *recognizability* of trace languages. The interest for this subject is twofold. From the point of view of concurrent systems, the class of *recognizable* trace languages, introduced in [3] using the standard notion of finite state automata over free partially commutative monoids, is interesting since the behaviors of labeled condition-event Petri nets can be described by recognizable trace languages [4]. On the other hand, relevant algebraic properties of recognizable trace languages have been discovered. In particular, except in the case of empty concurrency relations, Kleene's theorem does not hold for trace languages. In fact, it is not difficult to show that the class of recognizable trace languages over a given concurrent alphabet is a *proper* subclass of that of rational (or regular) trace languages (defined using the usual rational operations). This immediate fact motivated a deeper analysis of recognizability phenomenon in free partially commutative monoids (e.g., [9, 26, 23]).

A main notion in trace theory, which allows to treat concurrent systems from an algebraic point of view, is that of *asynchronous automata* [28]. Asynchronous automata are recognizing devices for trace languages characterized by a distributed control; thus, they can be seen as mathematical abstractions of concurrent systems. Notwithstanding the distributed organization, finite state asynchronous automata characterize the class of recognizable trace languages, that is the same class of trace languages characterized by finite automata over free partially commutative monoids (i.e., by devices with a *centralized* control). This very surprising and nontrivial result was proved by Zielonka [28].

Another kind of distributed devices recognizing trace languages was proposed in [29], introducing *asynchronous cellular automata*. This model is closely related to the first model of parallel computation, the *cellular automaton* introduced by Von Neumann [25]. We recall that a cellular automaton consists of a collection of elementary automata, with local interconnections, evolving in a parallel and synchronous way. While all these automata change state at the same time, in asynchronous cellular automata only nonconnected automata can concurrently act. Then, although asynchronous cellular automata and Von Neumann's cellular automata have some similarity, they are different models of computation.

Asynchronous automata and asynchronous cellular automata have been extensively studied in literature (e.g. [15, 27, 6, 8, 10, 11, 30]); moreover some extensions of these modes have been proposed (e.g. [20, 2, 18]). In this paper we will compare asynchronous automata and asynchronous cellular automata. We recall that, as proved in [11], also asynchronous cellular automata characterize the class of recognizable trace languages. Then, this model has the same recognizing power of asynchronous automata.

In the first part of the paper, we give polynomial-time reductions between asynchronous automata and asynchronous cellular automata. The construction of an asynchronous automaton accepting the same trace language of a given asynchronous cellular automaton is quite trivial and it is given only for sake of completeness. On the other hand, the converse construction is not so immediate and requires the use of some algebraic properties of prefixes of traces. This fact suggests the idea that asynchronous cellular automata are in some sense "more complicated" than asynchronous automata. This idea is supported also by the immediate observation that monoid automata coincide with asynchronous automata for empty concurrency relations, while monoid automata coincide with asynchronous cellular automata only when the alphabet is a singleton.

We strengthen the idea that asynchronous cellular automata are "more complicated" than asynchronous automata in the second part of the paper, where we study the problem of the existence of minimal asynchronous automata and of minimal asynchronous cellular automata. The interest in this subject is related to the fact that all known algorithms for the synthesis of deterministic asynchronous automata and of deterministic asynchronous cellular automata accepting given trace languages produce very big automata. Then, it should be very useful to have some technique for reducing the number of states of these automata.

In [6] it was proved that there are recognizable trace languages over concurrent alphabets with nontransitive dependency relation for which the minimum asynchronous automaton does not exists.[1] In this paper, we extend that result, showing that for every concurrent alphabet with nontransitive dependency relation there exists a recognizable trace language T accepted by *infinitely many* nonisomorphic minimal asynchronous automata with a finite number of states and by infinitely many non isomorphic minimal asynchronous automata with an infinite number of states. We obtain a similar result also for asynchronous cellular automata. In fact, we show that for every concurrent alphabet containing at least two dependent letters, there exists a trace language T that does not admit a minimum asynchronous cellular automaton

[1] We informally explain the terminology used in the paper. A (monoid, asynchronous, asynchronous cellular) automaton $\mathscr{A}$ is said to be *minimal* if it cannot be reduced, that is when we try to identify some different states of $\mathscr{A}$, we obtain an automaton that does not recognize the language accepted by $\mathscr{A}$. An automaton $\mathscr{A}$ accepting a trace language T is said to be *minimum* if all automata accepting T can be "reduced" to it. Then, the minimum automaton $\mathscr{A}$ accepting a given language, if any, is unique up to isomorphism and every minimal automaton accepting T is isomorphic to $\mathscr{A}$.

but that admits infinitely many nonisomorphic minimal finite state asynchronous cellular automata. On the other hand, we point out that this language T admits a unique minimum finite state asynchronous automaton.

Finally, we show that, notwithstanding polynomial-time reducibility between asynchronous automata and asynchronous cellular automata, the class of concurrent alphabets for which every recognizable trace language admits a minimum finite state asynchronous automaton (characterized in [6]) is wider than the class of concurrent alphabets for which every recognizable trace language admits a minimum finite state asynchronous automaton. In fact, we show that this last class contains only concurrent alphabets with full concurrency relations.

The paper is organized as follows. Basic definitions and facts about trace languages are recalled in Section 2, while the notions of asynchronous automata and of asynchronous cellular automata are recalled in Section 3. Section 4 is devoted to study some properties of α-prefixes of traces. These properties are used in Section 5 to state the reductions between the two models of automata. Finally, in Section 6, we state our main results on the existence of minimal asynchronous automata and of minimal asynchronous cellular automata.

2. Preliminary definitions and results

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

Definition 2.1. A concurrent alphabet is a pair (A, θ), where
- $A=\{a_1, \ldots, a_m\}$ is a finite alphabet;
- $\theta \subseteq A \times A$ is a symmetric and irreflexive relation, the *concurrency* or *independency* relation.

The complementary relation of the independency relation θ is called the *dependency relation* and in the following it will be denoted by $\bar{\theta}$. For every $a \in A$, we denote by $\bar{\theta}(a)$ the set of all letters depending on a, i.e., the set

$$\bar{\theta}(a)=\{b \in A \mid (a, b) \in \bar{\theta}\}.$$

As usual, the relations θ and $\bar{\theta}$ will be represented as graphs. Observe that, for $a \in A$, $\bar{\theta}(a)$ is the set containing a and the neighbors of a in the dependency graph. Every clique of the dependency graph, i.e., every subset $\alpha \subseteq A$ such that $\alpha \neq \emptyset$ and $(a, b) \in \bar{\theta}$, $\forall a, b \in \alpha$, will be called *dependency clique*.

Definition 2.2. The *free partially commutative monoid* (fpcm) $M(A, \theta)$ generated by a concurrent alphabet (A, θ) is the quotient structure $M(A, \theta)=A^*/\equiv_\theta$, where $\equiv_\theta$ is

the least congruence over A^* which extends the set of "commutativity laws":

$$\{ab = ba \mid a, b \in A \text{ and } (a, b) \in \theta\}.$$

A trace *is an element of* $M(A, \theta)$, *a* trace language *is a subset of* $M(A, \theta)$.

We denote by $[w]_\theta$ (or $[w]$ if θ is understood) the trace containing the string $w \in A^*$. Then, the product of the traces $[w]_\theta$ and $[v]_\theta$, denoted as $[w]_\theta[v]_\theta$, is the trace $[wv]_\theta$ and the trace $[\varepsilon]_\theta$, i.e., the equivalence class containing only the empty string ε of A^*, is the neutral element of $M(A, \theta)$.

A trace x is said to be a *prefix* of a trace t if and only if there exists a trace z such that $t = xz$.

With every formal language it is possible to associate a trace language in the following way.

Definition 2.3. Given a concurrent alphabet (A, θ) and a language $L \subseteq A^*$, the *trace language generated by* L *under* θ is the set $[L]_\theta = \{[w]_\theta \mid w \in L\}$.

Conversely, it is possible to associate with every trace language a formal language as follows.

Definition 2.4. The *linearization* $\mathrm{lin}(t)$ *of a trace* $t \in M(A, \theta)$ is the set of all strings of A^* belonging to the equivalence class t, i.e., $\mathrm{lin}(t) = \phi^{-1}(t)$, where ϕ denotes the canonical morphism from A^* to $M(A, \theta)$. The linearization $\mathrm{lin}(T)$ of a trace language $T \subseteq M(A, \theta)$ is the set containing all linearizations of traces of T, i.e., $\mathrm{lin}(T) = \bigcup_{t \in T} \mathrm{lin}(t)$.

Of course, for every $L \subseteq A^*$, it holds $L \subseteq \mathrm{lin}([L]_\theta)$.

It is possible to introduce Chomsky-like hierarchies of trace languages [1]. In this paper, we are interested in the class of trace languages accepted by finite state devices. So, we now recall the notion of monoid automata and, subsequently, that of *recognizable trace language* [17, 3].

Definition 2.5. Let M be a monoid with unit 1. An *automaton* $\mathscr{A}$ over M, *or* M-automaton, is a quadruple (Q, δ, I, F), where

- Q is *a set of states*;
- $\delta : Q \times M \to Q$ is a *transition function* such that
  $\delta(q, 1) = q$, for every $q \in Q$;
  $\delta(q, mm') = \delta(\delta(q, m), m')$ for every $m, m' \in M$, $q \in Q$;
- $I \in Q$ is the *initial state*;
- $F \subseteq Q$ is the set of *final states*.

The automaton $\mathscr{A}$ is said to be a *finite state* M-automaton if the set Q of states is finite. The *language recognized* by the M-automaton $\mathscr{A}$ is the set $L = \{m \in M \mid \delta(I, m) \in F\}$.

We recall that a state $q \in Q$ is *reachable* in the automaton $\mathscr{A}$ if and only if there exists $m \in M$ such that $\delta(I, m) = q$; the automaton $\mathscr{A}$ is said to be reachable if and only if

every state in Q is reachable. Of course, removing all nonreachable states and all transitions from these states, every nonreachable M-automaton can be transformed in a reachable automaton recognizing the same language. Thus, in this paper we will consider only reachable automata.

We observe that given a $M(A, \theta)$-automaton $\mathscr{A}=(Q, \delta, I, F)$, for every state $q \in Q$ and for every pair $(a, b) \in \theta$, it holds: $\delta(q, ab)=\delta(q, ba)$. This means that in $M(A, \theta)$-automata the concurrency among independent actions is reduced to their interleaving.

Definition 2.6. A trace language $T \subseteq M(A, \theta)$ is called *recognizable* iff there exists a finite state $M(A, \theta)$-automaton which recognizes T. The class of *recognizable trace languages* over the concurrent alphabet (A, θ), will be denoted by $\mathrm{Rec}(A, \theta)$.

Through the paper, for every finite set of indices J, for every vector $s=(s_j)_{j \in J}$ and for every subset J' of J, we will denote by $s_{|J'}$ the restriction of s to the elements indexed by J', i.e., $s_{|J'}=(s_j)_{j \in J'}$.

3. Asynchronous automata and asynchronous cellular automata

As observed in Section 2, automata over free partially commutative monoids are devices with an unique central control, where the concurrency among actions is reduced to their interleaving.

A different kind of recognizing devices for trace languages was proposed by Zielonka [28], introducing *asynchronous automata*. The structures of asynchronous automata and of monoid automata are very different. In fact, in asynchronous automata the control is distributed on 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. Despite this main difference, the classes of trace languages accepted by automata over free partially commutative monoids and by asynchronous automata coincides. This result is very surprising. In fact, from the point of view of concurrent systems, this means that *commutativity* can be reduced to *concurrency* [4].

Recently, a different model of automata with distributed control, called *asynchronous cellular automata* was proposed by Zielonka [29]. Also this model characterizes the class of recognizable trace languages.

In this section, we recall definitions and some properties of these two kinds of devices.

3.1. Asynchronous automata

First, we recall the notion of *asynchronous automata*.

Definition 3.1. An *asynchronous automaton* with n processes, over a concurrent alphabet (A, θ), is a tuple $\mathcal{A}=(P_1, \ldots, P_n, \{\delta_a\}_{a\in A}, I, F)$, where

- for $i=1, \ldots, n$, $P_i=(A_i, S_i)$ is the ith *process*, where S_i is its set of *local states* and A_i is its *local alphabet*, such that $\{A_1, \ldots, A_n\}$ is a clique cover of the dependency graph;
- let $\text{Proc}=\{1, \ldots, n\}$; the domain of $a\in A$ is the set $\text{Dom}(a)=\{i\in\text{Proc} \mid a\in A_i\}$, i.e., the set of (indices of) processes that "execute" the action a;
  then $\delta_a: \prod_{i\in\text{Dom}(a)} S_i \to \prod_{i\in\text{Dom}(a)} S_i$ is the (partial) *local transition function* associated with the letter a;
- let $S=\prod_{i\in\text{Proc}} S_i$ be the set of *global states*; then $I=(I_1, \ldots, I_n)$ is the *initial state* and $F\subseteq S$ is the set of *final states*.

If for every $i\in\text{Proc}$, S_i is a finite set, then the asynchronous automaton $\mathcal{A}$ is said to be *a finite state asynchronous automaton.*

We underline that the domains $\text{Dom}(a)$ and $\text{Dom}(b)$ of two actions $a, b\in A$ are disjoint if and only if a and b are independent; in this case the transition functions δ_a and δ_b act on disjoint sets of local states and, consequently, the corresponding actions a and b can be concurrently executed. In this way, asynchronous automata over $M(A, \theta)$ represent all concurrency among actions, specified by the relation θ.

For describing the "global behavior" of a given asynchronous automaton $\mathcal{A}$, we introduce the *global transition function* $\Delta: S\times M(A, \theta)\to S$ of $\mathcal{A}$, extending local transition functions to global states, as follows. Given $s\in S$ and $a\in A$, $\Delta(s, a)$ is the global state u such that $u_{|\text{Dom}(a)}=\delta_a(s_{|\text{Dom}(a)})$ and $u_{|\overline{\text{Dom}(a)}}=s_{|\overline{\text{Dom}(a)}}$. Intuitively, this corresponds to the fact that a transition on the letter a acts only on the processes in $\text{Dom}(a)$. This function can be extended to traces, in the usual way, by defining $\Delta(s, [\varepsilon])=s$ and $\Delta(s, ta)=\Delta(\Delta(s, t), a)$, for $s\in S$, $t\in M(A, \theta)$ and $a\in A$.

Thus, the *language* $T(\mathcal{A})$ accepted by the asynchronous automaton $\mathcal{A}$ can be defined as the set

$$T(\mathcal{A})=\{t\in M(A, \theta) \mid \Delta(I, t)\in F\}.$$

It is not difficult to verify that the tuple (S, Δ, I, F) is a $M(A, \theta)$-automaton accepting $T(\mathcal{A})$. This monoid automaton will be called in the following *sequential version* of the asynchronous automaton $\mathcal{A}$ and will be denoted by $\text{SEQ}(\mathcal{A})$. Thus, with every finite asynchronous automaton can be associated a finite state automaton recognizing the same trace language. Conversely, given a finite automaton over the fpcm $M(A, \theta)$ it is possible to construct an asynchronous automaton over the same concurrent alphabet, accepting the same language. This result, not at all obvious, was obtained by Zielonka.

Theorem 3.2 (Zielonka [28]). *The class of trace languages accepted by finite state asynchronous automata over the concurrent alphabet (A, θ) coincides with the class $\text{Rec}(A, \theta)$ of trace languages recognized by finite state $M(A, \theta)$-automata.*

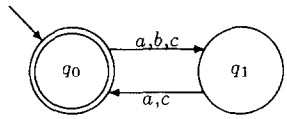

Fig. 1. $M(A, \theta)$-automaton accepting T.

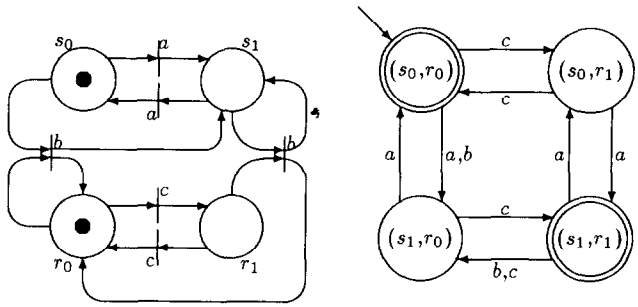

Fig. 2. An asynchronous automaton accepting T.

Example 3.3. Let (A, θ) be the concurrent alphabet with symbol set $A=\{a, b, c\}$ and concurrency relation $\theta=\{(a, c), (c, a)\}$. We consider the cliques $A_1=\{a, b\}$ and $A_2=\{b, c\}$ of the dependency graph and the trace language $T=[((a\cup b\cup c)(a\cup c))^*]_\theta$. The language T is recognizable. In fact, it is not difficult to see that the $M(A, \theta)$-automaton represented in Fig. 1 recognizes it. Let now $\mathscr{A}$ be the asynchronous automaton with two components $P_1=(A_1, S_1)$ and $P_2=(A_2, S_2)$, so defined:

- $S_1=\{s_0, s_1\}$ and $S_2=\{r_0, r_1\}$;
- $\delta_a(s_0)=s_1$, $\delta_a(s_1)=s_0$,
  $\delta_b(s_0, r_0)=(s_1, r_0)$, $\delta_b(s_1, r_1)=(s_1, r_0)$,
  $\delta_c(r_0)=r_1$, $\delta_c(r_1)=r_0$;
- $I=(s_0, r_0)$;
- $F=\{(s_0, r_0), (s_1, r_1)\}$.

As pointed out in [28], asynchronous automata can be represented as labeled Petri nets. In the following we will use this representation. In Fig. 2 the automaton $\mathscr{A}$ and its sequential version SEQ($\mathscr{A}$) are represented. Observe that the automaton of Fig. 1 cannot be the sequential version of any asynchronous automata over (A, θ).

3.2. Asynchronous cellular automata

We now recall the notion of asynchronous cellular automata introduced by Zielonka [29] and, independently, by Diekert [12].

Definition 3.4. An *asynchronous cellular automaton* over a concurrent alphabet (A, θ) is a tuple $\mathscr{A}=(\{S_a\}_{a\in A}, \{\delta_a\}_{a\in A}, I, F)$ such that

- for $a\in A$, S_a is the set of *local states* associated with the letter $a\in A$;
- for $a\in A$, δ_a is the (partial) *local transition function* associated with a, $\delta_a: \prod_{b\in\bar{\theta}(a)} S_b \to S_a$;
- let $S=\prod_{a\in A} S_a$ be the set of *global states* of $\mathscr{A}$; $I\in S$ is the *initial state* of $\mathscr{A}$ and $F\subseteq S$ is the set of *final states* of $\mathscr{A}$.

If, for every $a\in A$, S_a is a finite set, then the automaton $\mathscr{A}$ is said to be a *finite state* asynchronous cellular automaton.

By Definition 3.4, we can see an asynchronous cellular automaton as a net of automata $\{P_a\}_{a\in A}$. Every automaton P_a can execute only one action a and two automata are connected if and only if the corresponding actions do not commute. So, the graph of the net is an isomorphic copy of the dependency graph associated to the alphabet. The state that the automaton P_a of the net assumes after the execution of its action a depends on the states of its neighbors, that is the automata corresponding to letters noncommuting with a. For every pair of independent actions a and b, the transition function of the automaton P_a does not modify the states read as input by P_b and vice versa; then a and b can be concurrently executed. Moreover, P_a and P_b can read concurrently the states of their common neighbors.

As for asynchronous automata, we can associate with every asynchronous cellular automaton a *global transition function* $\Delta: S\times A\to S$ as follows: for $s\in S$, $a\in A$, $\Delta(s, a)$ is the global state u such that $u_b=s_b$ for all $b\in A$, $b\neq a$, and $u_a=\delta_a(s_{|\bar{\theta}(a)})$. The extension to traces can be obtained in a standard way.

Finally, the language *recognized* by the asynchronous cellular automaton $\mathscr{A}$ is the set $T(\mathscr{A})\subseteq M(A, \theta)$ so defined:

$$T(\mathscr{A})=\{t\in M(A, \theta)\mid \Delta(I, t)\in F\}.$$

As for asynchronous automata, it is possible to prove that cellular automata are "distributed" models characterizing the class of recognizable trace languages. In fact, it is easy to see that for every asynchronous cellular automaton $\mathscr{A}=(\{S_a\}_{a\in A}, \{\delta_a\}_{a\in A}, I, F)$, the $M(A, \theta)$-automaton defined by the tuple $\text{SEQ}(\mathscr{A})=(S, \Delta, I, F)$ and called *sequential version* of $\mathscr{A}$, accepts the same trace language $T(\mathscr{A})$ accepted by $\mathscr{A}$. Then, trace languages accepted by finite state asynchronous cellular automata over $M(A, \theta)$ are recognizable.

Conversely, the analogous of Theorem 3.2 for asynchronous cellular automata, proved in [11], holds.

Theorem 3.5. *The class of languages accepted by finite state asynchronous cellular automata over the concurrent alphabet (A, θ) coincides with the class* $\text{Rec}(A, \theta)$ *of trace languages recognized by finite state $M(A, \theta)$-automata.*

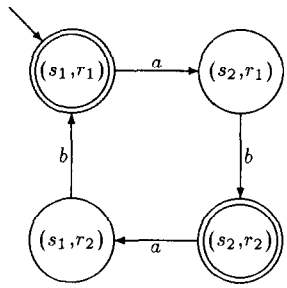

Fig. 3. The automaton SEQ($\mathscr{A}$).

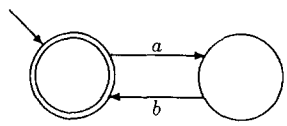

Fig. 4. A $M(A, \theta)$-automaton accepting $T(\mathscr{A})$.

Example 3.6. Let (A, θ) be the (degenerated) concurrent alphabet with $A=\{a, b\}$ and $\theta=\emptyset$. Then $\bar{\theta}(a)=\bar{\theta}(b)=\{a, b\}$.

Every asynchronous cellular automaton on this alphabet has two components P_a and P_b. The local transition functions δ_a and δ_b are applied to global states and return as value a local state of the corresponding component.

We consider the asynchronous cellular automaton $\mathscr{A}$ defined as follows:

- $S_a=\{s_1, s_2\}$ and $S_b=\{r_1, r_2\}$;
- $\delta_a(s_1, r_1)=s_2$, $\delta_a(s_2, r_2)=s_1$, $\delta_b(s_2, r_1)=r_2$, $\delta_b(s_1, r_2)=r_1$;
- $I=(s_1, r_1)$;
- $F=\{(s_1, r_1), (s_2, r_2)\}$.

It is immediate to see that such an automaton, whose sequential version is represented in Fig. 3, recognizes the language $T(\mathscr{A})=[(ab)^*]_\emptyset$.

Another monoid automaton accepting the language $T(\mathscr{A})$ is represented in Fig. 4. It is not difficult to see that this automaton cannot be the sequential version of any asynchronous cellular automaton.

Remarks

(a) It is obvious that if the independency relation θ is empty then every $M(A, \theta)$-automaton is also an asynchronous automaton over (A, θ) and vice versa. On the other hand, as shown in Example 3.6 this fact is not true for asynchronous cellular automata, except when $\# A=1$.

(b) If the independency is full, i.e., $\theta = A \times A - \{(a, a) \mid a \in A\}$, then every local alphabet of an asynchronous automaton contains exactly one letter. In this case every asynchronous is also an asynchronous cellular automaton and vice versa.

4. α-prefixes of traces

In this section we recall the notion of α-prefix of traces [28], related to properties of asynchronous and asynchronous cellular automata. This notion and its properties have been extensively studied in many papers (e.g. [10, 11, 28]) and will be useful to study the reduction from asynchronous cellular automata to asynchronous automata.

We recall, using the notation adopted in [5], that every trace t can be represented as a poset.

Definition 4.1. Given a trace $t \in M(A, \theta)$, let $x = x_1 \dots x_n$ be a representative of t, i.e., $[x]_\theta = t$; the *partial order* $\mathrm{ord}(t)$ associated with t is the pair $\mathrm{ord}(t) = (O_t, \leqslant_t)$, such that

(1) $O_t = \{(x_1, k_1), \dots, (x_n, k_n)\}$, where k_s denotes the number of symbols equal to x_s in the string $x_1 \dots x_s$;

(2) $\leqslant_t$ is the transitive closure of the relation L defined by:

$$(x_i, k_i)\, L(x_j, k_j) \quad \text{iff } (i \leqslant j \text{ and } (x_i, x_j) \notin \theta).$$

It is easy to show that there exists a bijection between prefixes of a trace t and order ideals of $\mathrm{ord}(t) = (O_t, \leqslant_t)$, i.e., the subsets of O_t closed with respect to the relation $\leqslant_t$ [5]. Let α be a subset of A. For every trace $t \in M(A, \theta)$ we consider the order ideal $\mathrm{Pref}_\alpha(t)$ containing exactly all symbol occurrences preceding the last occurrence in t of some letter in α, i.e.,

$$\mathrm{Pref}_\alpha(t) = \{(x_i, k_i) \mid \exists (x_j, k_j) \in O_t \colon (x_i, k_i) \leqslant_t (x_j, k_j) \text{ and } x_j \in \alpha\}.$$

The α-prefix $\partial_\alpha(t)$ is defined as the prefix of t corresponding to $\mathrm{Pref}_\alpha(t)$.

Our interest to the notion of α-prefix of a trace is motivated by the fact that processes of asynchronous automata and of asynchronous cellular automata work on prefixes of this kind. More precisely, it is immediate to verify that the state reached by the process P_a of an asynchronous cellular automaton $\mathscr{A}$, executing a trace t, depends on the prefix of t corresponding to the order ideal $\mathrm{Pref}_{\{a\}}(t)$ generated by the letter a. Then, we have the main equality $\Delta(I, t)_{|\{a\}} = \Delta(I, \partial_{\{a\}}(t))_{|\{a\}}$. Analogously, for every process P_i of an asynchronous automaton $\mathscr{A}$, we have $\Delta(I, t)_{|\{i\}} = \Delta(I, \partial_{A_i}(t))_{|\{i\}}$.

The properties stated in the following lemma are immediate consequences of previous definitions.

Lemma 4.2 *For every trace $t \in M(A, \theta)$, the following properties hold.*

(1) $\forall \alpha, \beta \subseteq A$, *if* $\alpha \subseteq \beta$ *then* $\partial_\alpha(\partial_\beta(t)) = \partial_\alpha(t)$;

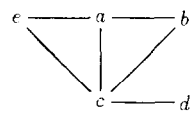

Fig. 5. Graph of θ.

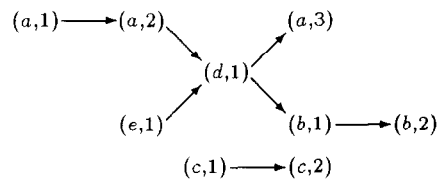

Fig. 6. The partial order ord(t).

(2) $\forall a \in A$, $\partial_a(ta) = \partial_{\bar{\theta}(a)}(t)a$;

(3) *for every dependency clique* $\alpha \subseteq A$, $\exists a \in \alpha$ *such that* $\partial_a(t) = \partial_\alpha(t)$.

It is possible to observe that if α is a dependency clique and $\partial_\alpha(t) \neq [\varepsilon]$, then there exists exactly one letter $a \in \alpha$ such that $\partial_a(t) = \partial_\alpha(t)$. In the following, this letter will be denoted as $\text{Last}_\alpha(t)$. Intuitively, $\text{Last}_\alpha(t)$ is the last letter of the set α executed in the trace t. If $\partial_\alpha(t) = [\varepsilon]$, then every letter $a \in \alpha$ verifies the equality $\partial_a(t) = \partial_\alpha(t)$; in this case, we will denote by $\text{Last}_\alpha(t)$ the minimal letter in the set α with respect to a fixed linear order on the alphabet A. It is immediate to see that $\text{Last}_\alpha(t) = \text{Last}_\alpha(\partial_\alpha(t))$.

Example 4.3. Given the concurrent alphabet (A, θ) with $A = \{a, b, c, d, e\}$, and the graphs of θ as in Fig. 5, we consider the clique cover constituted by the sets $A_1 = \{a, d\}$, $A_2 = \{b, d, e\}$ and $A_3 = \{c\}$.

The partial order ord(t) associated with the trace $t = [caaedabbc]$ is the transitive closure of the graph given in Fig. 6. It is not difficult to see that $\partial_a(t) = [aaeda]$, $\partial_b(t) = [aaedbb]$, $\partial_c(t) = [cc]$, $\partial_d(t) = [aaed]$ and $\partial_e(t) = [e]$. Moreover, $\text{Last}_{A_1}(t) = a$ $\text{Last}_{A_2}(t) = b$, and $\text{Last}_{A_3}(t) = c$.

5. Reductions between asynchronous cellular automata and asynchronous automata

In this section, we show that asynchronous automata and asynchronous cellular automata are polynomially related. While the reduction of asynchronous cellular automata to asynchronous automata is quite trivial, and it is presented only for completeness (see also [13]), the converse reduction is more complicated and it is obtained using the results presented in Section 4.

5.1. Construction of asynchronous cellular automata from asynchronous automata

The presentation of this construction is split in two parts. First, we define an asynchronous cellular automaton $\mathscr{B}$ with the property that for every dependency clique α and for every trace $t \in M(A, \theta)$, it is possible to recover the linear order among the last occurrences of symbols of α in t (and then the value of $\mathrm{Last}_\alpha(t)$) *only* comparing the local states reached by processes of $\mathscr{B}$ associated with the symbols belonging to α (i.e., without remembering all the trace t). This problem was previously solved with a similar construction using asynchronous automata in [8]. In the second part of the construction, we will extend the automaton $\mathscr{B}$ so defined in order to obtain an asynchronous cellular automaton $\mathscr{A}'$ simulating a given asynchronous automaton $\mathscr{A}$.

For the rest of this section, we fix a concurrent alphabet (A, θ) and a linear order $<$ on the set A. For every nonempty set $\alpha \subseteq A$, we denote by $\max(\alpha)$ and $\min(\alpha)$ the maximum and the minimum element of α with respect to the order relation $<$.

To define the asynchronous cellular automaton $\mathscr{B}$, we introduce a function G_a associating with every trace a boolean function, and we prove that for every dependency clique $\alpha \subseteq A$, from the set $\{G_a(\partial_a(t))\}_{a \in \alpha}$ it is possible to compute $\mathrm{Last}_\alpha(t)$.

Definition 5.1. Given a concurrent alphabet (A, θ) and a letter $a \in A$, fixed a linear order $<$ on A, we define inductively the function G_a associating with every trace a boolean function from $\bar{\theta}(a) - \{a\}$, i.e., $G_a : M(A, \theta) \to \{0, 1\}^{\bar{\theta}(a) - \{a\}}$, as follows.

$$\forall a' \in \bar{\theta}(a) - \{a\}, \quad \forall b \in A, \ b \neq a, \quad \forall t \in M(A, \theta),$$

$$G_a(\varepsilon)(a') = 0,$$

$$G_a(tb)(a') = \begin{cases} G_a(t)(a') & \text{if } b \neq a, \\ G_{a'}(t)(a) & \text{if } b = a \text{ and } a' > a, \\ 1 - G_{a'}(t)(a) & \text{if } b = a \text{ and } a' < a. \end{cases}$$

From this definition, it is immediate to see that for every $a \in A$ it holds $G_a(t) = G_a(\partial_a(t))$.

The main property of the mapping G is stated in the next lemma, and it is crucial to define the asynchronous cellular automaton $\mathscr{B}$.

Lemma 5.2. *For every dependency clique $\alpha \subseteq A$ and for all traces $t, r \in M(A, \theta)$, if $\forall a \in \alpha$ $G_a(t) = G_a(r)$ then $\mathrm{Last}_\alpha(t) = \mathrm{Last}_\alpha(r)$.*

Proof (*outline*). The reader can verify that for every pair of distinct letters $(a, b) \notin \theta$ and for every trace $u \in M(A, \theta)$, it holds:

$$\mathrm{Last}_{\{a,b\}}(u) = \begin{cases} \min\{a, b\} & \text{if } G_a(u)(b) = G_b(u)(a), \\ \max\{a, b\} & \text{otherwise.} \end{cases}$$

This permits to conclude that given two traces $t, r \in M(A, \theta)$ and a dependency clique $\alpha \subseteq A$ if $G_a(t) = G_a(r)$ for every $a \in \alpha$, then the linear order between the last occurrences of the letters of α in t coincides with the linear order of the last occurrences of the letters of α in r. However, by definition, the maximal elements of these linear orders are respectively $\text{Last}_\alpha(t)$ and $\text{Last}_\alpha(r)$. Thus, we can conclude that $\text{Last}_\alpha(t) = \text{Last}_\alpha(r)$. □

In other words, Lemma 5.2 shows that $\text{Last}_\alpha(t)$ can be computed, without knowing all the trace t, from the set $\{G_a(t)\}_{a \in \alpha}$; thus, we can write $\text{Last}_\alpha(\{G_a(t)_{a \in \alpha})$ instead than $\text{Last}_\alpha(t)$.

It is possible to observe that the time for computing $\text{Last}_\alpha(t)$ from the set $\{G_a(t)\}_{a \in \alpha}$, using the algorithm outlined in the proof of Lemma 5.2 is $O(\#(\alpha)^2) \leqslant O(d^2)$, where $d = \max\{\#(\bar{\theta}(\alpha)) - 1 \text{ s.t. } a \in A\}$ is the maximum degree of a vertex in the dependency graph.

Now, we are able to define the asynchronous cellular automaton $\mathscr{B}$. The set U_a of local states of the process associated with the letter a is

$$U_a = \{G_a(t) \mid t \in M(A, \theta)\}.$$

The initial global state is the tuple $J = (G_a(\varepsilon))_{a \in \alpha}$. The local transition function ξ_a corresponding to the letter a, associates with a tuple $(g_b)_{b \in \bar{\theta}(a)} \in \prod_{b \in \bar{\theta}(a)} U_b$ the local state $g'_a = \xi_a((g_b)_{b \in \bar{\theta}(a)})$ such that

$$g'_a(a') = \begin{cases} g_{a'}(a) & \text{if } a' > a, \\ 1 - g_{a'}(a) & \text{if } a' < a. \end{cases}$$

Comparing the definition of the transition function $(\xi_a)_{a \in A}$ with the definition of the function G_a, it is not difficult to conclude that for every trace $t \in M(A, \theta)$ it holds $\Xi(J, t) = (G_a(t))_{a \in A}$, where Ξ denotes the global transition function of $\mathscr{B}$.

At this point, the reader can verify that for every dependency clique $\alpha \subseteq A$ and for every trace $t \in M(A, \theta)$ it holds

$$\text{Last}_\alpha(t) = \text{Last}_\alpha(\Xi(J, t)_{|\alpha}),$$

i.e., $\text{Last}_\alpha(t)$ can be recovered from the local states reached by processes of $\mathscr{B}$ associated with symbols belonging to α.

Now, we can state the second part of the construction. Let $\mathscr{A} = (P_1, \ldots, P_n\{\delta_a\}_{a \in A}, I, F)$ be a finite state asynchronous automaton. We show how to build a finite state asynchronous cellular automaton $\mathscr{A}' = (\{S'_a\}_{a \in A}, \{\delta'_a\}_{a \in A}, I', F')$ accepting the same trace language of $\mathscr{A}$, using the above defined automaton $\mathscr{B}$.

The local states of the process P'_a of $\mathscr{A}'$ are pairs of the form (g, r), where $g \in U_a$ and $r \in \prod_{i \in \text{Dom}(a)} S_i$, and the transition functions are defined in such a way that the local state reached after the execution of a trace t is the pair $(G_a(t), \Delta(I, \partial_a(t))_{|\text{Dom}(a)})$. So, the second component of the local state of P'_a is used to simulate all processes of $\mathscr{A}$ that execute the action a. We observe that the same process P_i of $\mathscr{A}$ is simulated by all process P'_a of $\mathscr{A}'$ with $a \in A_i$, each one of them keeps in its internal state the local

state $\Delta(I, \partial_a t)_{|i}$; moreover:

$$\Delta(I, t)_{|i} = \Delta(I, \partial_{A_i}(t))_{|i} = \Delta(I, \partial_{\mathrm{Last}_{A_i}(t)}(t))_{|i}.$$

When a transition of $\mathscr{A}$ involving the process P_i has to be simulated, it is useful to get the correct value of $\Delta(I, t)_{|i}$. This can easily be done recovering the value of $\mathrm{Last}_{A_i}(t)$ from the first components of local states, i.e., the components simulating the automaton $\mathscr{B}$.

More formally, the initial global state of $\mathscr{A}'$ is $I' = (I'_a)_{a \in A}$, where $I'_a = (G_a(\varepsilon), (I_{|i})_{i \in \mathrm{Dom}(a)})$, and the local transition function associated with the symbol $a \in A$, where $\mathrm{Dom}(a) = \{i_1, \ldots, i_k\}$, is defined as

$$\delta'_a((g_b, s_b)_{b \in \bar{\theta}(a)}) = (\xi(g_b)_{b \in \bar{\theta}(a)}, \delta_a(s_{\mathrm{Last}_{A_{i_1}}((g_b)_{b \in A_{i_1}})}, \ldots, s_{\mathrm{Last}_{A_{i_k}}((g_b)_{b \in A_{i_k}})})),$$

for $g_b \in U_b$, $s_b \in \prod_{i \in \mathrm{Dom}(b)} S_i$, $b \in \bar{\theta}(a)$.

Using the standard algebraic manipulations, the reader can easily prove the following result.

Lemma 5.3. *For every trace $t \in M(A, \theta)$ the following equality holds:*

$$\Delta'(I', t) = (G_a(t), \Delta(I, \partial_a(t))_{|\mathrm{Dom}(a)})_{a \in A}.$$

Now we are able of completing the construction of the automaton $\mathscr{A}'$, stating the main result of this section.

Theorem 5.4. *Given an asynchronous automaton $\mathscr{A} = (P_1, \ldots, P_n, \{\delta_a\}_{a \in A}, I, F)$ over a concurrent alphabet (A, θ), it is possible to construct in polynomial time an asynchronous cellular automaton $\mathscr{A}' = (\{S'_a\}_{a \in A}, \{\delta'_a\}_{a \in A}, I', F')$, recognizing the same trace language T accepted by $\mathscr{A}$.*

Proof. We observe that, given two traces $t, t' \in M(A, \theta)$, if $\Delta'(I', t) = \Delta'(I', t')$ then also $\Delta(I, t) = \Delta(I, t')$. So, we can well define the set of final states of $\mathscr{A}'$ as $F' = \{\Delta'(I', t) \mid \Delta(I, t) \in F\}$. Of course, with this choice of final states, it turns out that the automaton $\mathscr{A}'$ recognizes just the trace language accepted by the given asynchronous automaton $\mathscr{A}$.

Now, we estimate the complexity of the reduction. Let d be the maximum degree of a node of the dependency graph, and let s be the maximum cardinality of the sets of local states of the automaton $\mathscr{A}$, i.e., $s = \max\{\#S_i \mid i \in \mathrm{Proc}\}$. Since $\#\{G_a(t) \mid t \in M(A, \theta)\} \leqslant 2^d$ and $\#\prod_{i \in \mathrm{Dom}(a)} S_i \leqslant s^n$, it turns out that the cardinality of the set of local states S'_a is at most $2^d s^n$; then it is polynomial in the cardinality of the sets of local states of $\mathscr{A}$. As observed in the previous section, the time for computing $\mathrm{Last}_\alpha(t)$, for every dependency clique α, is $O(d^2)$. So, given the table of δ_a, we can compute $\delta'_a((g_b, s_b)_{b \in \bar{\theta}(a)})$ in $O((\#\mathrm{Dom}(a))d^2) = O(nd^2)$ steps. This number is constant with respect to the dimension of the automaton $\mathscr{A}$. Hence, the time for

computing the table δ'_a is linear in its length $\#S'_a$, and, thus, it is polynomial with respect to the dimension of the automaton $\mathscr{A}$.

Finally, we explain how to compute in polynomial time the set F' of final states. We consider "global transition graphs" associated with automata $\mathscr{A}$ and $\mathscr{A}'$. The nodes of these graphs represent global states, while the arcs represent all possible transitions between global states. To the graph associated with $\mathscr{A}'$ we apply a depth-first visit, and we use the graph associated with $\mathscr{A}$ for choosing final states. More precisely, we use the following algorithm:

Procedure visit $(q' \in S'; q \in S)$
begin
  mark q' as visited
  if $q \in F$ **then** $F' := F' \cup \{q'\}$
(*) for every $a \in A$ s.t. $\Delta(q', a)$ is defined $\Delta(q', a)$ is not visited do
    visit$(\Delta'(q, a), \Delta(q, a))$
end.

The computation starts calling visit(I', I) with $F' = \emptyset$ and every global state of $\mathscr{A}'$ not visited. The more expensive step is the loop (*). It is executed at most $\#A \#S' \leqslant (\#A)^2 (O(S'))^{\#A} \leqslant O(s^{(\#A)^2})$ times. Then, we can conclude that, fixed the concurrent alphabet, the reduction is polynomial in the number of states of the given asynchronous automaton. □

5.2. *Construction of asynchronous automata from asynchronous cellular automata*

Now, we show that every asynchronous cellular automaton $\mathscr{A}$ over a concurrent alphabet (A, θ) can be reduced in polynomial time to an asynchronous automaton $\mathscr{A}'$, whose set of local alphabets $\{A_1, \ldots, A_n\}$ represents a clique cover of the dependency graph. The reduction is quite simple. The main idea is that the process P'_i, $i \in \text{Proc}$, of the automaton $\mathscr{A}'$ is obtained grouping together the processes P_a, $a \in A_i$, of $\mathscr{A}$.

Formally, the asynchronous automaton $\mathscr{A}'$ is defined as follows:

- for $i = 1, \ldots, n$, $S'_i = \prod_{a \in A_i} S_a$;
- for $a \in A$ with $\text{Dom}(a) = \{i_1, \ldots, i_k\}$, and for $(s_{i_1}, \ldots, s_{i_k}) \in \prod_{j=1}^{k} S'_{i_j}$, $\delta'_a(s_{i_1}, \ldots, s_{i_k})$ is defined if and only if there exists a vector $(r_b)_{b \in \bar{\theta}(a)} \in \prod_{b \in \bar{\theta}(a)} S_b$ of local states of $\mathscr{A}$ such that $s_{i_j} = (r_b)_{b \in A_{i_j}}$, $j = 1, \ldots, k$; in this case the value of $\delta'_a(s_{i_1}, \ldots, s_{i_k})$ is the vector $(u_{i_1}, \ldots, u_{i_k})$ where $u_{i_j} = (r'_b)_{b \in A_{i_j}}$ and

$$r'_b = \begin{cases} r_b & \text{if } b \neq a, \\ \delta_a((r_b)_{b \in \bar{\theta}(a)}) & \text{otherwise;} \end{cases}$$

- for $i \in \text{Proc}$, the component I'_i of the initial state is the tuple $(I_a)_{a \in A_i}$;
- $F' = \{s \in S' \mid \forall a \in A \ \exists r_a \in S_a \text{ s.t.} \forall i \in \text{Dom}(a) \, (s_i)_{\{a\}} = r_a, \text{ and } r \in F\}$.

Observe that every local state $s_i \in S_i'$ is a tuple of local states of $\mathscr{A}$; this tuple contains one state $(s_i)_{|\{a\}}$ for every $a \in A_i$. Thus, the transition function δ_a is defined only for "consistent" tuples of states.

If starting from its initial state and executing a trace t, the automaton $\mathscr{A}'$ so constructed, reaches the global state s, then the automaton $\mathscr{A}$, starting from its initial state and executing t, reaches the global state $(r_a)_{\{a \in A\}}$, where $r_a = (s_i)_{|\{a\}}$, for $i \in \mathrm{Dom}(a)$. More precisely, by induction on the length of traces, the following result can be proved.

Lemma 5.5. *Given a trace $t \in M(A, \theta)$, an asynchronous cellular automaton $\mathscr{A}$, the asynchronous automaton $\mathscr{A}'$ obtained applying to $\mathscr{A}$ the construction stated above, and the states $s \in S'$, $r \in S$ such that $s = \Delta'(I', t)$, $r = \Delta(I, t)$, we have $s_i = (r_a)_{a \in A_i}$, $i = 1, \ldots, n$.*

As a consequence of previous lemma, the following result can be immediately stated.

Corollary 5.6. *The automata $\mathscr{A}$ and $\mathscr{A}'$ recognize the same trace language.*

Using arguments similar to those of the Section 5.1, it is possible to verify that the construction stated here is a polynomial time reduction.

6. Minimal automata

A classical problem in automata theory is that of finding the minimum automaton accepting a given language [19]. Then, it is quite natural to study this problem also for asynchronous automata. This is important also since the known constructions of asynchronous automata (e.g. [10, 28]) produce automata with a very high number of states. While for every recognizable language there exists a minimum (up to isomorphism) monoid automaton recognizing it, this is no true for asynchronous automata [6]. In fact, there exists a trace language accepted by two nonisomorphic minimal (i.e., unreducible) asynchronous automata. In this section, after recalling some basic definitions, we deepen this investigation and we extend these results to asynchronous cellular automata.

First of all, we have to introduce the notions of reachable asynchronous (cellular) automata and of morphism between asynchronous (cellular) automata. The interest for reachable automata is related to the fact that, in order to minimize asynchronous automata, the first trivial step consists in eliminating from automata all useless states and all useless transitions. Before recalling the formal definition of reachable asynchronous automaton, we give an example.

Example 6.1. Consider the asynchronous automaton $\mathscr{A}$ on the concurrent alphabet $(A, \theta) = (\{a, b, c\}, \{(a, c), (c, a)\})$ represented in Fig. 7, where $S_1 = \{u_0, u_1, u_2, u_3, u_4\}$,

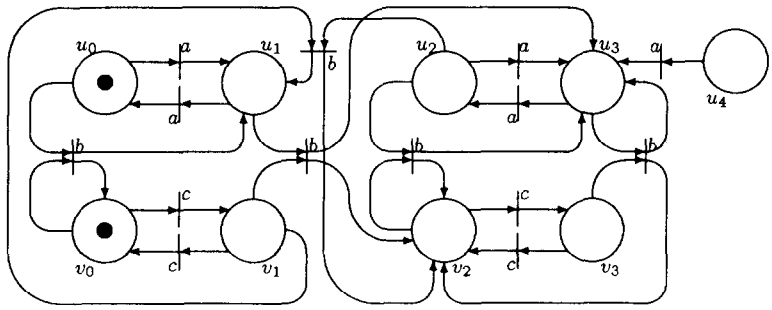

Fig. 7. The nonreachable asynchronous automaton $\mathscr{A}$.

$S_2=\{v_0, v_1, v_2, v_3\}$, $I=(u_0, v_0)$ and $F=\{(u_0, v_0), (u_1, v_1), (u_2, v_2), (u_3, v_3)\}$. It is clear that the component P_1 of $\mathscr{A}$ cannot reach the local state u_4. Thus, this state and the transition from it can be removed obtaining another automaton recognizing the same trace language. The local states $v_1 \in S_2$ and $u_2 \in S_1$ are reachable from the initial state (u_0, v_0), in fact $\Delta((u_0, v_0), c)=(u_0, v_1)$ and $\Delta((u_0, v_0), [acba])=(u_2, v_2)$. However, they are not simultaneously reachable, i.e., there is no trace t such that $\Delta(I, t)=(v_1, u_2)$, or, in other words, the global state (u_2, v_1) is not reachable. So, the transition $\delta_b(v_1, u_2)=(u_1, v_2)$ is never used, and then it can be removed, obtaining the automaton $\mathscr{A}'$ represented in Fig. 8. Observe that all local states and all transitions of the automaton $\mathscr{A}'$ are used in the computation over same trace. An automaton with such a property is said to be *reachable*. However, a reachable asynchronous automaton can have some nonreachable global state. For instance, the global state (u_2, v_1) of $\mathscr{A}'$ is not reachable.

Example 6.1 should be useful in understanding the meaning of the definition of reachable asynchronous automata, that can be formulated as follows.

Definition 6.2. Let $\mathscr{A}$ be an asynchronous automaton over a concurrent alphabet (A, θ). Given a set $\alpha \subseteq \text{Proc}$, a tuple of local states $s \in \prod_{i \in \alpha} S_i$ is said to be *reachable* whether there exists a trace $t \in (A, \theta)$ such that $\Delta(I, t)_{|\alpha}=s$.

The asynchronous automaton $\mathscr{A}$ is said to be reachable if and only if the following conditions hold.

- every local state s in S_i is reachable, $i=1, \ldots, n$;
- for every $a \in A$ and for every tuple $s \in \prod_{i \in \text{Dom}(a)} S_i$, if s is not reachable, then $\delta_a(s)$ is not defined.

As shown in Example 6.1, given an unreachable asynchronous automaton $\mathscr{A}$, it is easy to obtain a reachable automaton $\mathscr{A}'$ recognizing the same trace language, by removing all unreachable local states and all transitions from unreachable tuples of local states. Thus, from this point of *all asynchronous automata we will consider are supposed to be reachable.*

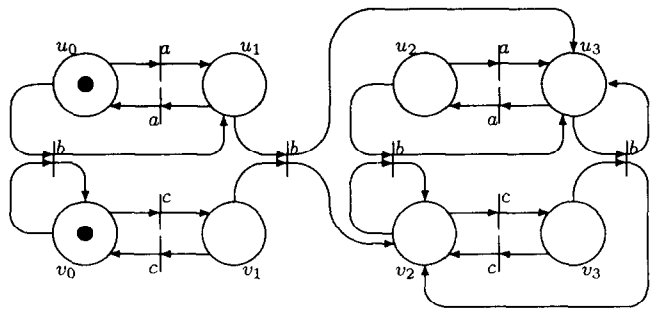

Fig. 8. The reachable asynchronous automaton obtained from $\mathscr{A}$.

The notion of *reachable* asynchronous cellular automaton is very similar to the notion of reachable asynchronous automaton given in Definition 6.2.

Definition 6.3. Let $\mathscr{A}$ be an asynchronous cellular automaton over a concurrent alphabet (A, θ). Given a set $\alpha \subseteq A$, a tuple $s \in \prod_{a\in\alpha} S_a$ of local states is said to be *reachable* if and only if there exists a trace $t \in M(A, \theta)$ such that $\Lambda(I, t)_{|\alpha} = s$.

The cellular automaton $\mathscr{A}$ is reachable if and only if the following conditions hold.

- every local state $s \in S_a$, $a \in A$, is reachable;
- for every $a \in A$, if a tuple of local state $s \in \prod_{b\in\bar{\theta}(a)} S_b$ is not reachable then $\delta_a(s)$ is not defined.

Of course, as for asynchronous automata, every unreachable asynchronous cellular automaton can be reduced to a reachable asynchronous cellular automaton. Thus, from this point on, *all asynchronous cellular automata we will consider are supposed to be reachable.*

Now, we introduce the notion of morphism between asynchronous (cellular) automata. Informally, a morphism between two automata $\mathscr{A}$ and $\mathscr{A}'$ is a family of maps from the sets of local states of $\mathscr{A}$ to the set of local states of $\mathscr{A}'$, mapping the initial state of $\mathscr{A}$ in the initial state of $\mathscr{A}'$, preserving the transitions, mapping final states in final states and nonfinal states in nonfinal states. Intuitively, a morphism describes how the states of $\mathscr{A}$ can be grouped together in order to obtain a "smaller" automaton $\mathscr{A}'$ recognizing the same trace language.

Definition 6.4. Given two finite state asynchronous automata $\mathscr{A} = (P_1, \ldots, P_n, \{\delta_a\}_{a\in A}, I, F)$ $\mathscr{A}' = (P'_1, \ldots, P'_n, \{\delta'_a\}_{a\in A}, I', F')$ over the same concurrent alphabet (A, θ),[2] a *morphism* ϕ *between* $\mathscr{A}$ *and* $\mathscr{A}'$ $(\phi : \mathscr{A} \to \mathscr{A}')$ is a family of functions $(\phi_i : S_i \to S'_i)_{i=1,\ldots,n}$ such that

[2] We suppose that processes of the automata $\mathscr{A}$ and $\mathscr{A}'$ have the same local alphabets, i.e., $A_1 = A'_1, \ldots, A_n = A'_n$; for instance, we can consider without loss of generality the maximal cliques of the dependency relation [14]. S_i and S'_i will denote the sets of local states of the processes P_i and P'_i, respectively.

- ϕ preserves the initial states, i.e., $\phi_i(I_i)=I'_i$, $i=1,\ldots,n$;
- preserves the transitions, i.e., for every $a\in A$ with $\mathrm{Dom}(a)=\{i_1,\ldots,i_k\}$ and for every reachable tuple $(s_{i_1},\ldots,s_{i_k})\in S_{i_1}\times\cdots\times S_{i_k}$, $\delta_a(s_{i_1},\ldots,s_{i_k})$ is defined if and only if $\delta'_a(\phi_{i_1}(s_{i_1}),\ldots,\phi_{i_k}(s_{i_k}))$ is, and, in this case, for $j=1,\ldots,k$, it holds

$$\phi_{i_j}((\delta_a(s_{i_1},\ldots,s_{i_k}))_{|\{i_j\}})=(\delta'_a(\phi_{i_1}(s_{i_1}),\ldots,\phi_{i_k}(s_{i_k})))_{|\{i_j\}};$$

- ϕ preserves the set of final states, i.e., for every reachable global state $s\in S$, $s\in F$ if and only if $(\phi_1(s_1),\ldots,\phi_n(s_n))\in F'$.

Example 6.5. Let $\mathscr{A}$ be the asynchronous automaton represented in Fig. 8 and $\mathscr{A}'$ be the asynchronous automaton represented in Fig. 2. Then, the pair of maps $(\phi_1:S'_1\to S_1,\ \phi_2:S'_2\to S_2)$ such that $\phi_1(u_0)=\phi_1(u_2)=s_0$, $\phi_1(u_1)=\phi_1(u_3)=s_1$, $\phi_2(v_0)=\phi_2(v_2)=r_0$ and $\phi_2(v_1)=\phi_2(v_3)=r_1$ defines a morphism from $\mathscr{A}$ to $\mathscr{A}'$.

We introduce now the notion of morphism for asynchronous cellular automata.

Definition 6.6. Given two finite state asynchronous cellular automata $\mathscr{A}=(\{S_a\}_{a\in A}, \{\delta_a\}_{a\in A}, I, F)$, $\mathscr{A}'=(\{S'_a\}_{a\in A}, \{\delta'_a\}_{a\in A}, I', F')$ over the same concurrent alphabet (A,θ), a *morphism* ϕ *from* $\mathscr{A}$ *to* $\mathscr{A}'$ $(\phi:\mathscr{A}\to\mathscr{A}')$ is a family of functions $(\phi_a:S_a\to S'_a)_{a\in A}$ such that

- ϕ preserves the initial states, i.e., $\phi_a(I_a)=I'_a$, $a\in A$;
- ϕ preserves the transitions, i.e., for every $a\in A$ and for every reachable tuple $(s_b)_{b\in\bar\theta(a)}\in\prod_{b\in\bar\theta(a)}S_b$, $\delta_a((s_b)_{b\in\bar\theta(a)})$ is defined if and only if $\delta'_a((\phi_b(s_b))_{b\in\bar\theta(a)})$ is, and, in this case it holds

$$\phi_a(\delta_a((s_b)_{b\in\bar\theta(a)}))=\delta'_a((\phi_b(s_b))_{b\in\bar\theta(a)});$$

- ϕ preserves the set of final states, i.e., for every reachable global state $s\in S$, $s\in F$ if and only if $(\phi_a(s_a))_{a\in A}\in F'$.

Using previous definitions it is not difficult to verify that given two asynchronous (cellular) automata $\mathscr{A}$ and $\mathscr{A}'$, if there exists a morphism $\phi:\mathscr{A}\to\mathscr{A}'$, then $\mathscr{A}$ and $\mathscr{A}'$ recognize the same trace language. Moreover, we can prove that, for reachable automata, this morphism is unique.

Lemma 6.7. *Given two reachable asynchronous (cellular) automata* $\mathscr{A}$ *and* $\mathscr{A}'$, *if there exists a morphism* $\phi:\mathscr{A}\to\mathscr{A}'$, *then this morphism is unique. Moreover,* ϕ *is surjective.*

Proof. For asynchronous automata the proof is given in the revised version of [6]. We adapt such a proof to asynchronous cellular automata.

Let $\psi:\mathscr{A}\to\mathscr{A}'$ be another morphism. First, we study what happens when there is a reachable global state $s\in S$ such that $\phi_a(s_a)=\psi_a(s_a)$, for all $a\in A$. Since ϕ and ψ preserve transitions and, for every $a\in A$, the tuple $(s_b)_{b\in\bar\theta(a)}$ is reachable, if

$\delta_a((s_b)_{b\in\bar{\theta}(a)})$ is defined then we have

$$\phi_a(\delta_a((s_b)_{b\in\bar{\theta}(a)}))=\delta'_a((\phi_b(s_b))_{b\in\bar{\theta}(a)})=\delta'_a((\psi_b(s_b)_{b\in\bar{\theta}(a)})=\psi_a(\delta_a((s_b)_{b\in\bar{\theta}(a)})).$$

So, we can conclude that $\phi(\Delta(s,a))=\psi(\Delta(s,a))$. Using this argument and the fact that morphisms preserve initial states, i.e., $\phi(I_a)=\psi(I_a)=I'_a$, $a\in A$, it is not difficult to conclude that two morphisms ψ and ϕ coincide.

To prove that ϕ is surjective, it is sufficient to remember that $\mathscr{A}'$ is reachable and to observe that $\phi(\Delta(s,t))=\Delta'(\phi(s),t)$, for any trace t and global state s of $\mathscr{A}$. □

In the following, we will denote by AA_T and ACA_T the families of reachable asynchronous automata and of reachable asynchronous cellular automata, recognizing a trace language T.

Definition 6.8. An automaton $\mathscr{A}$ (asynchronous automaton, asynchronous cellular automaton, resp.) in a family $\mathscr{C}$ of automata is called *minimal* or *reduced* if and only if for every automaton $\mathscr{A}'$ belonging to $\mathscr{C}$, every morphism ϕ from $\mathscr{A}$ to $\mathscr{A}'$ is an isomorphism. $\mathscr{A}$ is *minimum* if and only if for every automaton $\mathscr{A}'$ belonging to $\mathscr{C}$ there exists exactly one morphism from $\mathscr{A}'$ to $\mathscr{A}$.[3]

It should be clear that if a family $\mathscr{C}$ contains at least two minimal not isomorphic automata, then it cannot contain the minimum automaton; on the other hand, if $\mathscr{C}$ contains a minimum automaton $\mathscr{A}$, then every minimal automaton $\mathscr{A}'$ of $\mathscr{C}$ is isomorphic to $\mathscr{A}$. By Nerode's results [24], for every trace language T, the family of monoid automata accepting T contains a minimum (up to isomorphism) automaton. This fact is no more true when we consider asynchronous automata. For instance, the trace language $T=\{a,b,c\}$ over the concurrent alphabet (A,θ) with $A=\{a,b,c\}$ and $\theta=\{(a,c),(c,a)\}$ is accepted by two minimal non isomorphic asynchronous automata. These automata are represented in Fig. 9 (the set of final states are $\{(s_1,r_0),(s_0,r_1)\}$ and $\{(u_1,v_0),(u_0,v_1)\}$.

More precisely, the following result proved in [6], holds.

Theorem 6.9. *Let (A,θ) be a concurrent alphabet. Then the following sentences are equivalent.*

- *every recognizable trace language $T\subseteq M(A,\theta)$ admits a unique (up to isomorphism) minimum finite state asynchronous automaton;*
- *the dependency relation $\bar{\theta}$ is transitive.*

The fact that when the dependency relation is transitive, for every recognizable trace language there exists a unique minimum finite state asynchronous automaton was proved in [6] using Nerode's equivalence relations.

In the following we will show that there exists a trace language $T\subseteq M(A,\theta)$ with $\bar{\theta}$ not transitive, such that the family AA_T of asynchronous automata accepting it

[3] This definition can be done using categories as, for instance, in [16].

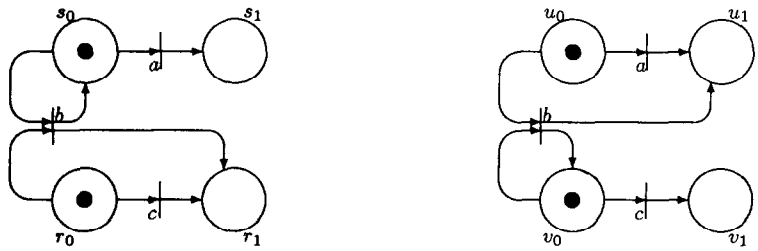

Fig. 9. Two minimal asynchronous automata.

contains *infinitely many* minimal *finite* state asynchronous automata and *infinitely many* minimal *infinite* state asynchronous automata.

6.1. Minimal asynchronous automata

Here, we deepen the analysis on the existence of minimal asynchronous automata started in [6] with Theorem 6.9.

To prove our results, it is useful the notion of *periodic* finite and infinite string, that now we recall.

Definition 6.10. Let Γ be a finite alphabet, and Γ^* the set of finite strings over Γ. We denote by Γ^ω the set of infinite strings over Γ, and by Γ^∞ the union of Γ^* and Γ^ω. A finite string $\gamma \in \Gamma^*$ is said to be periodic if and only if $\gamma = \eta^n$ for some finite string $\eta \in \Gamma^*$ and some integer $n > 1$.

An infinite string $\gamma \in \Gamma^\omega$ is said to be periodic if and only if $\gamma = \sigma\eta^\omega$ for some finite strings $\sigma, \eta \in \Gamma^*$ (where η^ω denotes the infinite string obtained concatenating infinitely many occurrences of the finite string η).

In the following, the ith symbol of a string $\gamma \in \Gamma^\infty$, will be denoted as γ_{i-1}. Then $\gamma = \gamma_0\gamma_1 \ldots \gamma_{m-1}$, $m = |\gamma|$, when γ is finite, and $\gamma = \gamma_0\gamma_1 \ldots$ when γ is finite.

The next lemma, whose proof is an immediate consequence of Definition 6.10, will be useful to obtain the main result of this section.

Lemma 6.11. *Let γ be a string in Γ^∞,*

(i) *if γ is a finite periodic string, then there exists an integer h, $0 < h < |\gamma|$, such that for every k, $0 \leqslant k \leqslant \gamma$, $\gamma_k = \gamma_{k \bmod h}$;*

(ii) *if γ is an infinite periodic string, then there exists two integers h, n, $n \geqslant 0$, $h \geqslant n$ such that for $k \geqslant n$, $\gamma_k = \gamma_{((k-n) \bmod (h-n))+n}$;*

(iii) *if γ is a finite or infinite non periodic string, then, for every pair (k, j) of integers, $0 \leqslant k, j < |\gamma|$, $k \neq j$, there exists an integer $h \geqslant 0$ such that $\gamma_{(k+h) \bmod |\gamma|} \neq \gamma_{(j+h) \bmod |\gamma|}$.*[4]

[4] With the convention that, when γ is infinite, $n \bmod |\gamma|$ is n, for every integer n.

We consider now the alphabet $\Gamma=\{0,1\}$ and with every (finite or infinite) string over Γ we associate an asynchronous automaton $\mathscr{A}_\gamma$. We will characterize the class of strings $\gamma\in\Gamma^\infty$ such that $\mathscr{A}_\gamma$ is a minimal asynchronous automaton, as the class of not periodic strings.

Let (A,θ) be the concurrent alphabet with $A=\{a,b,c\}$ and $\theta=\{(a,c),(c,a)\}$, and $T\subseteq M(A,\theta)$ be the trace language $T=[\{\{a,b,c\}\{a,c\}\}^*]_\theta$. Given $\gamma\in\{0,1\}^\infty$, with $|\gamma|=m$, we consider the asynchronous automaton $\mathscr{A}_\gamma=(P_1,P_2,\delta_a,\delta_b,\delta_c,I,F)$ defined in the following way:

$$A_1=\{a,b\},\qquad A_2=\{b,c\};$$

$$S_1=\{s_0,s_1\},\qquad S_2=\{r_{0k},r_{1k}\mid 0\leqslant k<m\};$$

$$\delta_a(s_0)=s_1,\ \delta_a(s_1)=s_0;$$

$$\text{for } 0\leqslant k<m\text{: } \delta_b(s_0,r_{0k})=\begin{cases}(s_0,r_{1k}) & \text{if } \gamma_k=0,\\ (s_1,r_{0k}) & \text{if } \gamma_k=1;\end{cases}$$

$$\text{for } 0\leqslant k<m\text{: } \delta_b(s_1,r_{1k})=(s_1,r_{0(k+1)\bmod m})$$

$$\text{for } 0\leqslant k<m\text{: } \delta_c(r_{0k})=r_{1k},\ \delta_c(r_{1k})=r_{0k};$$

$$I=(s_0,r_{00});$$

$$F=\{(s_0,r_{0k}),(s_1,r_{1k})\mid 0\leqslant k<m\}.$$

The automata $\{\mathscr{A}_\gamma\}$ have a particular structure. We can observe that considering the Petri Nets representation of the part of an automaton $\mathscr{A}_\gamma$, corresponding to states s_0, s_1, r_{0k}, r_{1k} and $r_{0k'}$, where $k'=k+1 \bmod |\gamma|$, $0\leqslant k<|\gamma|$, we obtain one of two patterns represented in Fig. 10. The asynchronous automaton $\mathscr{A}_1$ is that represented in Fig. 2, while the sequential version of the automaton $\mathscr{A}_{101}$ is represented in Fig. 11.

It is easy to see that every automaton $\mathscr{A}_\gamma$ recognizes the trace language $T=[((a\cup b\cup c)(a\cup c))^*]_\theta$. In fact, we can identify all final states of SEQ$(\mathscr{A}_\gamma)$ in a unique state q_0 and all nonfinal states of SEQ$(\mathscr{A}_\gamma)$ in a unique state q_1, obtaining in this way the $M(A,\theta)$-automaton of Fig. 1. Now, we will show that the minimal automata of the family $\{\mathscr{A}_\gamma\}$ are exactly those corresponding to non periodic strings of Γ^∞.

Theorem 6.12. *Let γ be a string in $\{0,1\}^\infty$. Then γ is not periodic if and only if the automaton $\mathscr{A}_\gamma$ is minimal.*

Proof. We start outlining the proof of the fact that if γ is periodic then the automaton $\mathscr{A}_\gamma$ is not minimal. First, we consider the case $|\gamma|=m<\infty$. By definition, there exists a string $\eta\in\{0,1\}^*$ and an integer $n>1$ such that $\gamma=\eta^n$. Thus, it is not difficult to find a morphism from $\mathscr{A}_\gamma$ to $\mathscr{A}_\eta$.

Now, suppose that the periodic string γ is infinite. Then, by definition, there exists two finite strings σ, $\eta\in\Gamma^*$ such that $\gamma=\sigma\eta^\omega$. Let n and h denote respectively, the

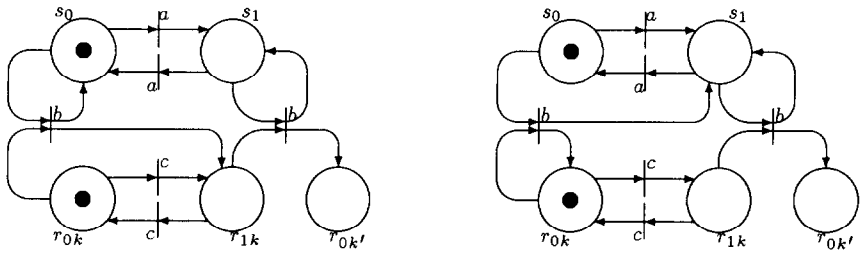

Fig. 10. Patterns corresponding to $\gamma_k=0$ and $\gamma_k=1$.

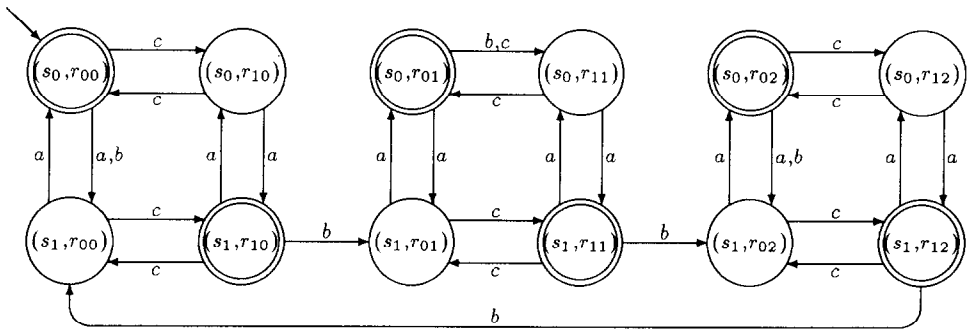

Fig. 11. Sequential version of the asynchronous automaton $\mathscr{A}_{101}$.

lengths of strings σ and $\sigma\eta$, i.e., $\sigma=\gamma_0 \dots \gamma_{n-1}$ and $\eta=\gamma_n \dots \gamma_{h-1}$. We consider the asynchronous automaton $\mathscr{A}'$ so defined:

$$A'_1=A_1=\{a,b\}, \qquad A'_2=A_2=\{b,c\};$$

$$S'_1=\{s'_0,s'_1\}, \qquad S'_2=\{r'_{00},r'_{10},r'_{01},r'_{11},\dots,r'_{0h-1},r'_{1h-1}\};$$

$$\delta'_a(s'_0)=s'_1, \qquad \delta'_a(s'_1)=s'_0;$$

$$\text{for } 0\leqslant k<h, \quad \delta'_b(s'_0,r'_{0k})=\begin{cases}(s'_0,r'_{1k}) & \text{if } \gamma_k=0,\\ (s'_1,r'_{0k}) & \text{if } \gamma_k=1;\end{cases}$$

$$\text{for } 0\leqslant k<h, \quad \delta'_b(s'_1,r'_{1k})=\begin{cases}(s'_1,r'_{0k+1}) & \text{if } k+1<h\\ (s'_1,r'_{0n}) & \text{otherwise;}\end{cases}$$

$$\delta'_c(r'_{0k})=r'_{1k}, \qquad \delta'_c(r'_{1k})=r'_{0k};$$

$$I'=(s'_0,r'_{00});$$

$$F'=\{(s'_0,r'_{0k}),(s'_1,r'_{1k})\mid 0\leqslant k<h\}.$$

We observe that $\sigma=\varepsilon$ the automaton $\mathscr{A}'$ coincides with the automaton $\mathscr{A}_\eta$; for $\sigma\neq\varepsilon$ we obtain an automaton very similar to the automaton $\mathscr{A}_{\sigma\eta}$: the only difference is in the transition on the letter b from the global state (s'_1,r'_{1h-1}).

We define now a pair of functions $\phi=(\phi_1, \phi_2)$ from local states of $\mathscr{A}_\gamma$ to local states of $\mathscr{A}'$, in the following way:

$$\phi_1(s_0)=s'_0 \quad \text{and} \quad \phi_1(s_1)=s'_1;$$

$$\text{for } i\in\{0,1\}, \quad \phi_2(r_{ik})=\begin{cases} r'_{ik} & \text{if } k<n, \\ r'_{i((k-n) \bmod (h-n))+n} & \text{otherwise.} \end{cases}$$

Using the second statement of Lemma 6.11 it is possible to verify that ϕ is a morphism from $\mathscr{A}_\gamma$ to $\mathscr{A}$.

Now, we show that if γ is not a periodic string then the automaton $\mathscr{A}_\gamma$ is minimal.

Let $m=|\gamma|$ and ϕ be a morphism from $\mathscr{A}_\gamma$ to an asynchronous automaton $\mathscr{A}$. We have to prove that ϕ is an isomorphism. Since all automata considered are reachable and then all morphisms are surjective, if, by contradiction, we suppose that ϕ is not an isomorphism, then we can find two local states q, $p\in S_i$, for some $i\in\{1,2\}$, whose images by ϕ coincide, i.e., $\phi_i(q)=\phi_i(p)$.

We consider all possible cases.

- $\phi_1(s_0)=\phi_1(s_1)$.
  Since ϕ preserves final and nonfinal states, we have the contradiction $(\phi_1(s_0), \phi_2(r_{00}))\in F'$, $(\phi_1(s_1), \phi_2(r_{00}))\notin F'$ and $(\phi_1(s_0), \phi_2(r_{00}))=(\phi_1(s_1), \phi_2(r_{00}))$.
- $\phi_2(r_{0k})=\phi_2(r_{1j})$, for some $0\leqslant k, j<m$.
  As in the previous case, using the fact that ϕ preserves final and nonfinal states, we obtain a contradiction: $(\phi_1(s_0), \phi_2(r_{0k}))\in F'$, $(\phi_1(s_0), \phi_2(r_{1j}))\notin F'$ and $(\phi_1(s_0), \phi_2(r_{0k}))=(\phi_1(s_0), \phi_2(r_{1j}))$.
- $\phi_2(r_{0k})=\phi_2(r_{0j})$, for some k, j with $k\neq j$, and the equivalent case $\phi_2(r_{1k})=\phi_2(r_{1j})$.
  The fact that these two cases are equivalent can be proved recalling that ϕ preserves transitions. Then, starting from $\phi_2(r_{0k})=\phi_2(r_{0j})$, we obtain $\phi_2(r_{1k})=\phi_2(\delta_c(r_{0k}))=\delta'_c(\phi_2(r_{0k}))=\delta'_c(\phi_2(r_{0j}))=\phi_2(\delta_c(r_{0j}))=\phi_2(r_{1j})$; in a similar way, starting from $\phi_2(r_{1k})=\phi_2(r_{1j})$, we obtain $\phi_2(r_{0k})=\phi_2(r_{0j})$.
  We have to consider two subcases: $\gamma_k\neq\gamma_j$ and $\gamma_k=\gamma_j$.

– $\gamma_k\neq\gamma_j$ (without loss of generality $\gamma_k=1$ and $\gamma_j=0$).
  From $\phi(r_{0k})=\phi(r_{0j})$, using the fact that ϕ preserves transitions, we obtain

$$(\phi_1(s_0), \phi_2(r_{1j}))=\delta'_b(\phi_1(s_0), \phi_2(r_{0j}))=\delta'_b(\phi_1(s_0), \phi_2(r_{0k}))=(\phi_1(s_1), \phi_2(r_{0k})).$$

  Then, $\phi_1(s_0)=\phi_1(s_1)$. But, as shown above, this is contradictory.

– $\gamma_k=\gamma_j$.
  From $\phi_2(r_{1k})=\phi_2(r_{1j})$, using the fact that ϕ preserves transitions it turns out that

$$\begin{aligned}(\phi_1(s_1), \phi_2(r_{0(k+1) \bmod m}))&=\delta'_b(\phi_1(s_1), \phi_2(r_{1k})) \\ &=\delta'_b(\phi_1(s_1), \phi_2(r_{1j}))=(\phi_1(s_1), \phi_2(r_{0(j+1) \bmod m})).\end{aligned}$$

  Thus, we can conclude that $\phi_2(r_{0(k+1) \bmod m})=\phi_2(r_{0(j+1) \bmod m})$. Iterating this proof for h times we obtain $\phi_2(r_{0(k+h) \bmod m})=\phi_2(r_{0(j+h) \bmod m})$. But, by the last statement of

Lemma 6.11 there exists an integer $h \geqslant 0$ such that $\gamma_{(k+h) \bmod m} \neq \gamma_{(j+h) \bmod m}$. Thus, we return to previous case.

In all considered possible cases we have obtained a contradiction. Then, we can conclude that the automaton $\mathscr{A}_\gamma$ is minimal. □

Using Theorem 6.12, we can easily state the following result.

Corollary 6.13. *Let (A, θ) be a concurrent alphabet with nontransitive dependency relation. Then there exists a trace language $T \subseteq M(A, \theta)$ accepted by infinitely many minimal non isomorphic asynchronous automata with a finite number of states and by infinitely many minimal nonisomorphic asynchronous automata with an infinite number of states.*

6.2. Minimal asynchronous cellular automata

In this section, we extend to asynchronous cellular automata the results of [6] and of Section 6.1. We will show that, despite the existence of polynomial time reductions between asynchronous automata and asynchronous cellular automata, the class of concurrent alphabets for which every recognizable trace language admits a minimum asynchronous cellular automaton is different from the class characterized in Theorem 6.9 for asynchronous automata. In fact, we will prove that for every concurrent alphabet (A, θ) containing at least two dependent letters, there exists a recognizable language over (A, θ) accepted by infinitely many nonisomorphic minimal finite states asynchronous cellular automata.

To state this result, we consider the (degenerated) concurrent alphabet (A, θ) where $A = \{a, b\}$ and $\theta = \emptyset$, and, for $n \geqslant 1$, the asynchronous cellular automaton $\mathscr{A}_n = (S_a, S_b, \delta_a, \delta_b, I, F)$ where

$$S_a = \{s_0, s_1, s_2\},\ S_b = \{r_0, r_1, \ldots, r_{2n}\};$$

$$\delta_a(s_0, r_{2k}) = s_1, \text{ for } 0 \leqslant k < n,$$

$$\delta_a(s_1, r_{2k+1}) = s_0, \text{ for } 0 \leqslant k < n-1,$$

$$\delta_a(s_1, r_{2n-1}) = s_2,$$

$$\delta_a(s_2, r_{2n}) = s_0,$$

$$\delta_b(s_0, r_{2k+1}) = r_{2k+2}, \text{ for } 0 \leqslant k < n-1,$$

$$\delta_b(s_0, r_{2n}) = r_0,$$

$$\delta_b(s_1, r_{2k}) = r_{2k+1}, \text{ for } 0 \leqslant k \leqslant n-1,$$

$$\delta_b(s_2, r_{2n-1}) = r_{2n};$$

$$I = (s_0, r_0);$$

$$F = \{(s_0, r_{2i}) \mid 0 \leqslant i < n\} \cup \{(s_1, r_{2i+1}) \mid 0 \leqslant i < n\} \cup \{(s_2, r_{2n})\}.$$

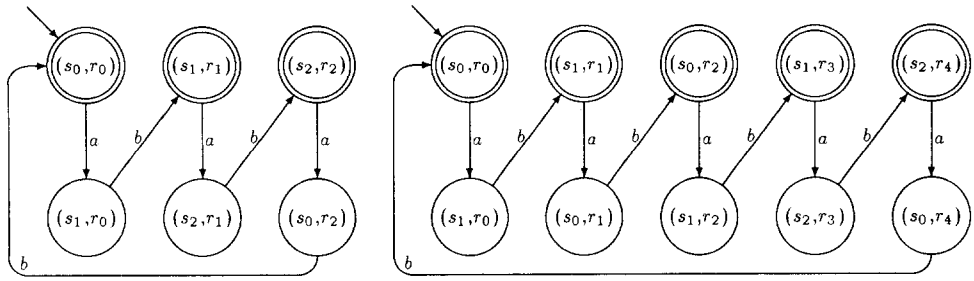

Fig. 12. Sequential versions of $\mathscr{A}_1$ and $\mathscr{A}_2$.

The sequential versions of automata $\mathscr{A}_1$, $\mathscr{A}_2$ are represented in Fig. 12.
We prove now the following result.

Theorem 6.14. *For every integer $n \geqslant 1$, the asynchronous cellular automaton $\mathscr{A}_n$ is minimal.*

Proof. By contradiction, consider a morphism ϕ from $\mathscr{A}_n$ to an asynchronous cellular automaton $\mathscr{A}'$ and suppose that ϕ is not an isomorphism. Thus, there exists a symbol $c \in \{a, b\}$ and two states $s, s' \in S_c$ such that $\phi_c(s) = \phi_c(s')$. We consider all possible cases.

- $\phi_a(s_i) = \phi_a(s_j)$, $i, j \in \{0, 1, 2\}$ with $i \neq j$.
  In the case $i=0, j=1$, using the fact that ϕ preserves final states it is immediate to obtain the contradiction $(\phi_a(s_0), \phi_b(r_0)) = (\phi_a(s_1), \phi_b(r_0)), (\phi_a(s_0), \phi_b(r_0)) \in F'$ and $(\phi_a(s_1), \phi_b(r_0)) \notin F'$.
  For the other pairs (i, j) the proof is similar.
- $\phi_b(r_i) = \phi_b(r_j)$, $i, j \in \{0, 1, \ldots, 2n\}$, with $i \neq j$.
  We have to consider the following subcases.

– i is even and $j = 2n$.
  From $\phi_b(r_i) = \phi_b(r_{2n})$, we obtain $(\phi_a(s_0), \phi_b(r_i)) = (\phi_a(s_0), \phi_b(r_{2n}))$. Since ϕ preserves final and nonfinal states, it turns out that the first pair is a final state of $\mathscr{A}'$ while the second is not. Thus, we obtain a contradiction.
– i is odd and $j = 2n-1$. From $(\phi_a(s_1), \phi_b(r_i)) = (\phi_a(s_1), \phi_b(r_{2n-1}))$, using the fact that ϕ preserves transitions on a, we obtain $\phi_a(s_0) = \phi_a(s_2)$, but, as shown above, this is contradictory.
– Both i and j are even, or both i and j are odd and $i < j < 2n-1$.
  If i and j are even (odd, resp.) then the global states (s_1, r_i) and (s_1, r_j) ((s_0, r_i) and (s_0, r_j), resp.) are reachable. Using the fact that ϕ preserves transitions on the letter b, we obtain $\phi_b(r_{i+1}) = \phi_b(r_{j+1})$, and iterating this argument, $\phi_b(r_{i+2n-1-j}) = \phi_b(r_{2n-1})$, i.e., the previous case.
– i is even and j is odd.
  If $j = 2n-1$ and $i < 2n$, then the global states (s_1, r_i) and (s_1, r_{2n-1}) are reachable, but $\delta_b(s_1, r_i)$ is defined, while $\delta_b(s_1, r_{2n-1})$ is not defined. This is contradictory, since morphisms preserve transitions.

Analogously, if $j=2n-1$ and $i=2n$ then $\delta_a(s_2, r_{2n})$ is defined and $\delta_a(s_2, r_{2n-1})$ is not defined, and if $j<2n-1$ and $i<2n$ then $\delta_b(s_0, r_i)$ is not defined, while $\delta_b(s_0, r_j)$ is defined. Finally, if $j<2n-1$ and $i=2n$, then $\delta_b(s_0, r_{2n})=r_0$, $\delta_b(s_0, r_j)=r_{j+1}$ and $\phi(r_0)=\phi(r_{j+1})$, where $j+1$ is even, but, as above shown, this is contradictory. □

Now, we obtain the main result of this section.

Theorem 6.15. *Given a concurrent alphabet (A, θ) the following sentences are equivalent:*

- *every recognizable trace language $T \subseteq M(A, \theta)$ admits a minimum finite states asynchronous cellular automaton;*
- *$M(A, \theta)$ is a free totally commutative monoid.*

Moreover, when $M(A, \theta)$ is nontotally commutative, there are trace languages over (A, θ) accepted by infinitely many minimal asynchronous cellular automata.

Proof. If the concurrency relation is full, i.e., $\theta = A \times A - \{(a, a) \mid a \in A\}$, then every asynchronous cellular automaton over $M(A, \theta)$ is also an asynchronous automaton over $M(A, \theta)$ and vice versa. Then, as a consequence of Theorem 6.9, every recognizable trace language $T \subseteq M(A, \theta)$ admits a minimum finite state asynchronous (cellular) automaton.

Conversely, if $M(A, \theta)$ is not totally commutative, then there are two letters $a, b \in A$ such that $a \neq b$ and $(a, b) \in \bar{\theta}$. By Theorem 6.14, the language $[(ab)^*]_\theta$ is accepted by infinitely many minimal non isomorphic asynchronous cellular automata; then, the minimum does not exist. □

Acknowledgment

I am indebted to the anonymous referees for helpful comments.

References

[1] I.J.J. Aalbersberg and G. Rozenberg, Theory of traces, *Theoret. Comput. Sci.* **60** (1988) 1–82.

[2] A. Arnold, An extension of the notions of traces and of asynchronous automata, *RAIRO Inform. Theor. Appl.* **25** (1991) 355–393.

[3] A. Bertoni, M. Brambilla, G. Mauri and N. Sabadini, An application of the theory of free partially commutative monoids: asymptotic densities of trace languages, in: *Proc. 10th MFCS*, Lecture Notes in Computer Science Vol. 118 (Springer, Berlin, 1981) 205–215.

[4] A. Bertoni, G. Mauri and N. Sabadini, Concurrency and commutativity, Tech. Report, University of Milan, 1982. Presented at *3rd European Workshop on Applications and Theory of Petri Nets*, Varenna, 1982.

[5] A. Bertoni, G. Mauri and N. Sabadini, Membership problem for regular and context-free trace languages, *Inform. and Comput.* **82** (1989) 135–150.

[6] D. Bruschi, G. Pighizzini and N. Sabadini, On the existence of the minimum asynchronous automaton and on decision problems for unambiguous regular trace languages, in: *Proc. 5th STACS*, Lecture Notes in Computer Science Vol. 294, (Springer, Berlin, 1988) 334–346. A revised version will appear in *Inform. and Comput.*

[7] P. Cartier and M. Foata, Problèmes combinatories de commutation et réarrangements, Lecture Notes in Mathematics Vol. 85 (Springer, Berlin 1969).

[8] R. Cori, M. Latteux, Y. Roos and E. Sopena, 2-asynchronous automata, *Theoret. Comput. Sci.* **61** (1988) 93–102.

[9] R. Cori and Y. Métivier, Recognizable subsets of some partially abelian monoids, *Theoret. Comput. Sci.* **35** (1985) 179–189.

[10] R. Cori and Y. Métivier, Approximation of a trace, asynchronous automata and the ordering of events in a distributed system, in: *Proc. 15th ICALP*, Lecture Notes in Computer Science, Vol. 317 (Springer, Berlin, 1988) 147–161.

[11] R. Cori, Y. Métivier and W. Zielonka, Asynchronous mappings and asynchronous cellular automata, Tech. Report 89-97, LaBRI, Université de Bordeaux I, 1989.

[12] V. Diekert, Combinatorial rewriting on traces, in: *Proc. 7th STACS*, Lecture Notes in Computer Science, Vol. 415 (Springer, Berlin, 1990) 138–151.

[13] V. Diekert, Combinatorics on traces, Lecture Notes in Computer Science, Vol. 454 (Springer, Berlin, 1990).

[14] C. Duboc, Commutations dans les monoïdes libres: Un cadre théorique pour l'étude du parallélism, Ph.D. Thesis, Université de Rouen, 1986.

[15] C. Duboc, Mixed product and asynchronous automata, *Theoret. Comput. Sci.* **48** (1986) 183–199.

[16] H. Ehrig, K. Kiermeier, H. Kreowski and W. Kühnel, *Universal Theory of Automata* (Teubner, Stuttgart, 1974).

[17] S. Eilenberg, *Automata, Languages and Machines*, Vol. A (Academic Press, New York, 1974).

[18] P. Gastin and A. Petit, Asynchronous automata for infinite traces, in: *Proc. 19th ICALP*, Lecture Notes in Computer Science, Vol. 623 (Springer, Berlin, 1992) 583–594.

[19] J. Hopcroft and J. Ullman, *Introduction to Automata Theory, Languages and Computations* (Addison–Wesley, Reading, MA, 1979).

[20] S. Jesi, G. Pighizzini and N. Sabadini, Probabilistic asynchronous automata, in: *Proc. Workshop: Free Partially Commutative Monoids*, Institüt für Informatik – Technische Universität München – TUM-I9002 (1990) 99–114. A revised version will appear in *Math. Systems Theory.*

[21] A. Mazurkiewicz, Concurrent program schemes and their interpretations, Tech. Report DAIMI Rep. PB-78, Aarhus University, 1977.

[22] A. Mazurkiewicz, Trace theory, in: *Advances in Petri Nets 1986*, Lecture Notes in Computer Science, Vol. 255 (Springer, Berlin, 1986) 279–324.

[23] Y. Métivier, On recognizable subsets of free partially commutative monoids, *Theoret. Comput. Sci.* **58** (1988) 201–208.

[24] A. Nerode, Linear automaton transformations, *Proc. Amer. Math. Soc.* **9** (1958) 541–544.

[25] J. Von Neumann, *Theory of Self-reproducing Automata* (Univ. of Illinois Press, Champaign, IL, 1966) Revised by A.W. Burks.

[26] E. Ochmański, Regular behaviour of concurrent systems, *EATCS Bull.* **27** (1985) 56–67.

[27] Y. Roos, Automates virtuellement asynchrones, Tech. Report IT-93, Laboratoire d'Informatique fondamentale de Lille, Univ. Lille Flandres Artois, 1987.

[28] W. Zielonka, Notes on finite asynchronous automata, *RAIRO Inform Theor. Appl.* **21** (1987) 99–135.

[29] W. Zielonka, Safe executions of recognizable trace languages by asynchronous automata, in: *Proc. Logic at Botik '89*, Lecture Notes in Computer Science, Vol. 363 (Springer, Berlin, 1989) 278–289.

[30] W. Zielonka, Asynchronous automata, in: *Proc. Workshop: Free Partially Commutative Monoids*, Institüt für Informatik – Technische Universität München – TUM-I9002 (1990) 183–197.