SAFE EXECUTIONS OF RECOGNIZABLE TRACE LANGUAGES BY ASYNCHRONOUS AUTOMATA

Wiesław Zielonka

Institute of Computer Science
Polish Academy of Sciences
00-901 Warsaw
PKiN P.O. Box 22
POLAND

1. Introduction

In this paper we are interested in concurrent systems on the level of action occurrences. As was shown by Mazurkiewicz[7] the behaviour of these systems can be conveniently described by trace languages, i.e. subsets of partially commutative monoids in the same way as the behaviour of sequential systems is described by formal languages.

Here we consider the class of recognizable trace languages.

We introduce a new kind of asynchronous finite devices - finite asynchronous cellular automata that recognize exactly all recognizable trace languages. We show that every recognizable trace language can be realized by a safe and unambiguous finite asynchronous cellular automaton, when safety means that every computation starting from an initial state can be extended to an accepted computation.

The same result holds also for finite asynchronous automata defined in [10] because every finite asynchronous cellular automaton can be transformed easily to a finite asynchronous automaton in a way that preserves all interesting properties such as safety and unambiguity.

2. Traces and recognizable trace languages

In this section we fix the notation and recall some elementary properties of traces. A great part of this material is folklore.

We begin with notational remarks. If $i,j \in \mathbb{N}$ are integers then $[i,j]$ denotes the set $\{ k \in \mathbb{N} : i \leq k \leq j \}$. For a set X by $\mathcal{P}(X)$ is denoted the family of all subsets of X. Finally, for a word x over an alphabet A and a letter $a \in A$ we assume that $|x|$ is the length of x, $alph(x)$ is the set of letters that occur in x, $|x|_a$ is the number

of occurrences of a in x. The empty word is denoted by **1**.

The pair (A,θ) is a *concurrent alphabet* if A is a finite set of actions and $\theta\subseteq A\times A$ is a symmetric and irreflexive binary relation over A, *the independency relation*. We introduce an equivalence relation $\simeq_\theta$ over A^*. For two words $u,v\in A^*$, $u\simeq_\theta v$ if there exists a sequence of words $w_1,\ldots,w_{k+1}\in A^*$ such that $u=w_1$, $v=w_{k+1}$ and $\forall\, i\in[1,k]$, $\exists\, x_i,y_i\in A^*$, $\exists\,(a,b)\in\theta$, $w_i=x_iaby_i$ & $w_{i+1}=x_ibay_i$.

We can establish easily that $\simeq_\theta$ is a congruence over A^*, i.e. if $x\simeq_\theta y$ and $u\simeq_\theta v$ then $xu\simeq_\theta yv$ for all $x,y,u,v\in A^*$.

By definition the quotient $M(A,\theta)=A^*/\simeq_\theta$ of A^* by $\simeq_\theta$ is the *partially commutative monoid* over (A,θ). Its elements, i.e. equivalence classes of $\simeq_\theta$ are called *traces*. If $u\in A^*$ then $[u]_\theta$ denotes the trace represented by u, i.e. the equivalence class of u. Thus the mapping $[\ ]_\theta : A^*\longrightarrow M(A,\theta)$ that maps $x\in A^*$ to $[x]_\theta$ is the canonical homomorphism from A^* to $M(A,\theta)$.

If $L\subseteq A^*$, $T\subseteq M(A,\theta)$ then $[L]_\theta=\{\,[u]_\theta : u\in L\,\}$ is the image of L under $[\ ]_\theta$ while $[T]_\theta^{-1}=\{\,u\in A^* : [u]_\theta\in T\,\}$ is the inverse image of T.

Subsets of $M(A,\theta)$ will be called *trace languages*.

Let $t=[x]_\theta\in M(A,\theta)$ be a trace, then we define $|t|=|x|$, $|t|_a=|x|_a$, $alph(t)=alph(x)$ for $a\in A$. The traces $[a]_\theta$ for $a\in A$ and $[1]_\theta$ will be denoted simply by a and **1** respectively. This does not lead to a confusion because it will be clear from the context if we are in the domain of words or traces.

$\theta_d=A\times A\setminus\theta$ will denote the *dependency* relation associated with θ.
For $\alpha\subseteq A$ we set $\theta_d(\alpha)=\{b\in A : \exists a\in\alpha,\ (a,b)\in\theta_d\}$.
Thus $\theta_d(\alpha)$ contains actions that depend on any action from α.
Observe that θ_d is reflexive and thus $\forall\, a\in A$, $a\in\theta_d(a)$.
The independency relation will sometimes be written in the infix form $a\theta b$ instead of $(a,b)\in\theta$. Consequently if $\alpha,\beta\subseteq A$ and $t,r\in M(A,\theta)$ then we write $\alpha\theta\beta$ and $t\theta r$ to denote that $\alpha\times\beta\subseteq\theta$ and $alph(t)\times alph(r)\subseteq\theta$ respectively.

If $t=ur$ for $t,u,r\in M(A,\theta)$ then u is a *prefix* of t and this fact is denoted by $u\ll t$.

Definition 2.1[4,6]
Let M *be a monoid with the unit element* 1. *A finite state* M*-automaton is a quintuple* $A=(M,Q,q_o,\delta,F)$, *where* Q *is a finite set of states* , q_o *is the initial state, and* $\delta : Q\times M \longrightarrow Q$ *is the transition mapping satisfying the following conditions*
$\delta(q,1)=q$ *for every* $q\in Q$ *and* $\delta(q,m_1m_2)=\delta(\delta(q,m_1),m_2)$ *for all* $q\in Q$ *and* $m_1,m_2\in M$. *A recognizes the subset* $L(A)=\{ m\in M : \delta(q_o,m)\in F \}$ *of* M *and a subset* $L\subseteq M$ *is recognizable iff there exists a finite state* M*-automaton recognizing* L. □

The following simple but very useful proposition establishes a link between recognizable trace languages and recognizable languages.

Proposition 2.2
T$\subseteq M(A,\theta)$ *is recognizable a trace language iff* $[T]_\theta^{-1}\subseteq A^*$ *is a recognizable language.* □

For other more sophisticated algebraic characterizations of recognizable trace languages we refer to Ochmański[9], Duboc[3] and Zielonka[10].

3. Finite Asynchronous Cellular Automata

A finite asynchronous cellular automaton $\mathbb{A}$ over a concurrent aphabet (A,θ) consists of
(i) a graph $\mathcal{G}$ with A as the set of vertices and where an edge joins $a\in A$ and $b\in A$ iff $(a,b)\in\theta_d$,
(ii) a family $\{ \mathbb{S}_a : a\in A \}$ of finite sets, where every $\mathbb{S}_a$ is interpreted as the set of states of $a\in A$,
(iii) a family $\{ \delta_a : a\in A \}$ of transition mappings, where for every $a\in A$, $\delta_a : \bigtimes_{b\in\alpha}\mathbb{S}_b \longrightarrow \mathcal{P}(\mathbb{S}_a)$, $\alpha=\theta_d(a)$.

Elements of the cartesian product $\mathbb{S}=\bigtimes_{a\in A}\mathbb{S}_a$ are *global states* of $\mathbb{A}$. We also distinguish the sets $\mathbb{I},\mathbb{F}\subseteq\mathbb{S}$ of *initial* and *final* states respectively.
Let $\emptyset\neq\alpha\subseteq A$, then $\Pi_\alpha : \bigtimes_{a\in A}\mathbb{S}_a \longrightarrow \bigtimes_{a\in\alpha}\mathbb{S}_a$ will denote the projection of the global states onto the set $\bigtimes_{a\in\alpha}\mathbb{S}_a$. If $\alpha=\{a\}$ then we write Π_a.

The bahaviour of $\mathbb{A}$ is explained in terms of a game on $\mathcal{G}$ with A as a set of players placed on the vertices of the graph $\mathcal{G}$. If a player $a\in A$ makes a move then he examines the states of all his neighbours from $\theta_d(a)$ and changes his own state in accordance with his transition mapping δ_a. This single move is viewed as atomic , which implies that two neighbouring players $a,b\in A$, $(a,b)\in\theta_d$, cannot move simultaneously. On the other hand, since there is no global synchronizing mechanism nonadjacent players are free to move concurrently.
Formally, if $s\in\mathbb{S}$ is a global state and the player a moves at s then he replaces his actual state $\Pi_a(s)$ by $s'\in\delta_a(\Pi_a(s))$, where $\alpha=\theta_d(a)$.

Now we introduce the global transition mapping
$\Delta : \mathbb{S}\times A^* \longrightarrow \mathcal{P}(\mathbb{S})$ of $\mathbb{A}$
defined in the following way:
(Δ1) $\Delta(s,1)=\{s\}$ for all $s\in\mathbb{S}$,
(Δ2) if $s',s''\in\mathbb{S}$ and $a\in A$ then $s''\in\Delta(s',a)$ if
(i) $\forall b\in A\setminus\{a\}$, $\Pi_b(s'')=\Pi_b(s')$,and
(ii) $\Pi_a(s'')\in\delta_a(\Pi_a(s'))$, $\alpha=\theta_d(a)$.
We extend Δ to words as in the case of finite automata, $\Delta(s,xy)=\{ s'\in\mathbb{S} : \exists\, s''\in\mathbb{S},\ s''\in\Delta(s,x) \ \&\ s'\in\Delta(s'',y) \}$ for $s\in\mathbb{S}$ and $x,y\in A^*$.

Observe that if $a\theta b$ then $\Delta(s,ab)=\Delta(s,ba)$ and in general, for $u,v\in A^*$, if $u\simeq_\theta v$ then $\Delta(s,u)=\delta(s,v)$ for any $s\in\mathbb{S}$. This remark and the fact that certain moves can be performed concurrently by the players show that it is more natural to consider the sequences of moves up to $\simeq_\theta$, i.e. rather than words $u\in A^*$ we should take traces $[u]_\theta\in M(A,\theta)$ to describe the behaviour of $\mathbb{A}$.
We extend Δ to $\Delta : \mathbb{S}\times M(A,\theta) \longrightarrow \mathcal{P}(M(A,\theta))$ by setting
$\Delta(s,[x]_\theta)=\Delta(s,x)$ for all $x\in A^*$ and $s\in\mathbb{S}$.

Formally, the finite asynchronous cellular automaton is just the finite automaton $\mathbb{A}=(A,\mathbb{S},\mathbb{I},\Delta.\mathbb{F})$,where the set of states $\mathbb{S}$ is the cartesian product $\bigtimes_{a\in A}\mathbb{S}_a$ and the transition function Δ is represented by the family $\{ \delta_a : a\in A \}$ of mappings in the way described by the conditions (Δ1) and (Δ2).
$L(\mathbb{A})=\{ x\in A^* : \exists\, s_0\in\mathbb{I},\ \exists\, s_f\in\mathbb{F},\ s_f\in\Delta(s_0,x) \}$
is the language recognized by $\mathbb{A}$, while
$T(\mathbb{A})=\{ t\in M(A,\theta) : \exists\, s_0\in\mathbb{I},\ \exists\, s_f\in\mathbb{F},\ s_f\in\Delta(s_0,t) \}$

is the trace language recognized by $\mathbb{A}$.
Note that $T(\mathbb{A})=[L(\mathbb{A})]_\theta$ and $L(\mathbb{A})=[T(\mathbb{A})]_\theta^{-1}$. Thus $T(\mathbb{A})$ is a recognizable trace language.
As it turns out the inverse also holds:

Proposition 3.1
For every recognizable trace language $T\subseteq M(A,\theta)$ *there exists a deterministic finite asynchronous cellular automaton* $\mathbb{A}$ *over* $M(A,\theta)$ *recognizing* T. □

In Proposition 3.1 "deterministic" means that $\mathbb{A}$ is deteministic as a finite automaton, i.e. $card(\mathbb{I})=1$ and $\forall s\in\mathbb{S}$, $\forall a\in A$, $card(\Delta(s,a))\leq 1$.
The proof of Proposition 3.1 is not trivial. It is similar to the proof of the corresponding result in the case of finite asynchronous automata in [10] and will be given in a separate paper[2] . Some related ideas were introduced in [1]. Note also that there are some simpler interesting cases , e.g. if the graph $\mathcal{G}$ is acyclic[8].

The defined here finite asynchronous cellular automata relate closely to the usual cellular automata. The set A of the vertices of the graph $\mathcal{G}$ corresponds to the set of cells. In the cellular automata the connection pattern is usually regular, e.g. this can be a grid and they work in the synchronized mode while in the case of the finite asynchronous cellular automata the connection pattern is given by the graph $\mathcal{G}$ of the dependency relation θ_d and actions (moves) are executed asychronously.

4. Safe implementation of recognizable trace languages by asynchronous cellular automata

As it turns out the implementations of recognizable trace languages by deterministic finite asynchronous cellular automata may sometimes show incurable defects.

Definition 4.1.
Let $\mathbb{A}=(A,\mathbb{S},\mathbb{I},\Delta,\mathbb{F})$ *be a finite asynchronous cellular automaton. A global state* $s\in\mathbb{S}$ *is accessible (coaccessible) if there exist an initial state* $s_0\in\mathbb{I}$ *(a final state* $s_f\in\mathbb{F}$ *respectively) and a trace* $t\in M(A,\theta)$ *such that* $s\in\Delta(s_0,t)$ ($s_f\in\Delta(s,t)$ *resp.*). $\mathbb{A}$ *is safe if*

every accessible state $s \in S$ *is coaccessible.*

□

Example 4.2.

Let $A=\langle a,b,c\rangle$, $\theta=\langle (a,b),(b,a)\rangle$, $L=((a^2b \cup ab^2)c)^*$ and $T=[L]_\theta$.
The trace language T is recognizable and we can construct a deterministic finite asynchronous cellular automaton recognizing T. An example of such an automaton is given below.
We set

$S_a=\langle s_a^0, s_a^1, s_a^2, r_a^1, r_a^2\rangle$, $S_b=\langle s_b^0, s_b^1, s_b^2, r_b^1, r_b^2\rangle$, $S_c=\langle s_c, r_c\rangle$,

$I=\langle (s_a^0, s_b^0, s_c)\rangle$,

$F=\langle (s_a^i, s_b^j, r_c) : i,j\in\langle 1,2\rangle,\ i\neq j\rangle \cup \langle (r_a^i, r_b^j, s_c) : i,j\in\langle 1,2\rangle,\ i\neq j\rangle$,

$\delta_a(s_a^0, s_c)=s_a^1$, $\delta_b(s_b^0, s_c)=s_b^1$

$\delta_a(s_a^1, s_c)=s_a^2$, $\delta_b(s_b^1, s_c)=s_b^2$,

$\delta_c(s_a^1, s_b^2, s_c)=r_c=\delta_c(s_a^2, s_b^1, s_c)$,

$\delta_a(s_a^1, r_c)=\delta_a(s_a^2, r_c)=r_a^1$, $\delta_b(s_b^1, r_c)=\delta_b(s_b^2, r_c)=r_b^1$,

$\delta_a(r_a^1, r_c)=r_a^2$, $\delta_b(r_b^1, r_c)=r_b^2$,

$\delta_c(r_a^1, r_b^2, r_c)=s_c=\delta_c(r_a^2, r_b^1, r_c)$.

$\delta_a(r_a^1, s_c)=\delta_a(r_a^2, s_c)=s_a^1$, $\delta_b(r_b^1, s_c)=\delta_b(r_b^2, s_c)=s_b^1$.

The reader is encouraged to draw the transition diagram for A.
Note that in this automaton we have $\Delta((s_a^0, s_b^0, s_c), a^2b^2)=(s_a^2, s_b^2, s_c)$. But from the global state (s_a^2, s_b^2, s_c) we cannot reach any final state and in fact we cannot move from this state at all. Thus the presented automaton is not safe. □

Thus although the automaton A from Example 4.2 recognizes the given trace language T this implementation seems to have a defect. In our distributed system there is no global control device and so we are not able to prevent the execution of the trace $[a^2b^2]_\theta$ by A. Intuitively this means that besides the desired computations A may perform some other computations which we can detect only by inspecting from time to time the current global state to see if it is coaccessible. This is due to the fact that the players have only a limited knowledge , if a and b are nonadjacent , $(a,b)\in\theta_d$, then they do not see their states one another. This holds in Example 4.2 where the players a,b may perform the sequence

$[a^2b^2]_\theta$ because a is unvisible directly for b and vice versa. On the other hand, the introduction of a global arbiter to our game restricts considerably the degree of parallelism in the system and creates immediately other problems, for example how to find a checkpoint at which $\mathbb{A}$ should be restarted.
This gives rise to the following central question of our paper.
Does there exist for every recognizable trace language T a safe finite asynchronous cellular automaton $\mathbb{A}$ recognizing T?
For the language given in Example 4.2 the answer is positive as it is shown below.

Example 4.3.
Let A,θ,T be as in Example 4.2. We set

$$\mathbb{S}_a=\{s_a^1,s_a^2,r_a^1,r_a^2\},\ \mathbb{S}_b=\{s_b^1,s_b^2,r_b^1,r_b^2\},\ \mathbb{S}_c=\{s_c^{12},s_c^{21},r_c^{12},r_c^{21}\}.$$

The initial and final states are
$$\mathbb{I}=\mathbb{F}=\{(s_a^i,s_b^j,r_c^{kl}),(r_a^i,r_b^j,s_c^{kl}) : i,j,k,l\in\{1,2\},\ i\neq j,\ k\neq l\},$$
and the transition mappings are given by

$$\left.\begin{array}{ll}\delta_a(s_a^i,r_c^{jk})=r_a^1 & \delta_b(s_b^i,r_c^{jk})=r_b^1,\\ \delta_a(r_a^i,s_c^{jk})=s_a^1 & \delta_b(r_b^i,s_c^{jk})=s_b^1,\end{array}\right\}\ i,j,k\in\{1,2\},\ i\neq k$$

$$\delta_a(s_a^1,s_c^{21})=s_a^2 \quad \delta_b(s_b^1,s_c^{12})=s_b^2,$$

$$\delta_a(r_a^1,r_c^{21})=r_a^2 \quad \delta_b(r_b^1,r_c^{12})=r_b^2,$$

$$\delta_c(s_a^1,s_b^2,s_c^{12})=\delta_c(s_a^2,s_b^1,s_c^{21})=\{r_c^{12},r_c^{21}\},$$

$$\delta_c(r_a^1,r_b^2,r_c^{12})=\delta_c(r_a^2,r_b^1,r_c^{21})=\{s_c^{12},s_c^{21}\}.$$

In this way we obtain a safe automaton recognizing T but it is no longer deterministic. □

The following proposition shows that it is not by accident that the automaton from Example 4.3 is not deterministic.

Proposition 4.4.
Let (A,θ) and $T\subseteq M(A,\theta)$ be as in Example 4.2 and let $\mathbb{A}=(A,\mathbb{S},\mathbb{I},\Delta,\mathbb{F})$ be a safe finite asynchronous cellular automaton recognizing T. *Then*
(i) $\mathbb{A}$ *has at least two input states, i.e.* $\mathrm{card}(\mathbb{I})\geq 2$, *and*
(ii) there exists an accesible state $s\in\mathbb{S}$ *and an action* $x\in A$ *such*

that card(Δ(s,x))≥2.

Proof

Suppose that 𝔄 recognizes T and there exists an accessible state s∈𝕊 such that

$\Delta(s,[a^2b]_\theta)\neq\emptyset\neq\Delta(s,[ab^2]_\theta)$.

Let $s=(s_a,s_b,s_c)$, $(s'_a,s'_b,s_c)\in\Delta(s,[ab^2]_\theta)$, $(s''_a,s''_b,s_c)\in\Delta(s,[a^2b]_\theta)$.

Since $a\theta b$ it is easy to see that $(s''_a,s'_b,s_c)\in\Delta(s,[a^2b^2]_\theta)$. If $t\in M(A,\theta)$ is such that $s\in\Delta(s_o,t)$ for some $s_o\in\mathbb{I}$ then $(s''_a,s'_b,s_c)\in\Delta(s_o,ta^2b^2)$. But ta^2b^2 is not a prefix for any trace from T and hence (s''_a,s'_b,s_c) is not coaccessible.

This shows that if 𝔄 is safe then for any accessible state s∈𝕊 either $\Delta(s,[a^2b]_\theta)=\emptyset$ or $\Delta(s,[ab^2]_\theta)=\emptyset$.

Since there exists an initial state from which $[a^2b]_\theta$ can be excuted and the same holds for the trace $[ab^2]_\theta$, the preceding fact entails immediately (i).

We shall prove (ii). Let k=card(𝕀×𝕊) and $T_k=[((a^2b\cup ab^2)c)^k]_\theta\subseteq T$. Note hat T_k contains exactly 2^k traces. Since $2^k>k$ this implies that there exist different traces $t',t''\in T_k$, $t'\neq t''$ and states $s_o\in\mathbb{I}$, $s\in\mathbb{S}$ such that $\Delta(s_o,t')=\Delta(s_o,t'')=s$. Since $t'\neq t''$ there exist traces t_o,r',r'' such that $t'=t_o[ab^2]r'$ and $t''=t_o[a^2b]_\theta r''$. If (ii) does not hold then $\Delta(s_o,t_o)$ contains exactly one state, say s', and $\Delta(s',[a^2b]_\theta)\neq\emptyset\neq\Delta(s',[ab^2]_\theta)$. But we have just shown that in this case 𝔄 cannot be safe. □

Proposition 4.4 shows that the finite asynchronous cellular automata differ from the finite automata. Namely we can always construct a deterministic and safe finite automaton. For this it suffices to take any deterministic finite automaton recognizing a given regular language L and simply remove all states that are not coaccessible. This procedure is not applicable for the cellular asynchronous automata, if we remove the state (s_a^2,s_b^2,s_c) from the automaton 𝔄 in Example 4.2 then we should remove all the global states that contain s_a^2,s_b^2,s_c changing in this way the recognized language.

However there is one important case when a safe and deterministic finite asynchronous cellular automaton can be constructed.

Proposition 4.5.
Note that if the independency relation is empty, $\theta=\emptyset$, then $\forall u\in A^*$, $[u]_\theta=\{u\}$ and $M(A,\theta)$ is isomorphic with A^*. Thus any language $L\subseteq A^*$ can be viewed as a trace language in $M(A,\emptyset)$.
Let $L\subseteq A^*$ be a regular language. Then there exists a deterministic and safe finite asynchronous cellular automaton over the concurrent alphabet $(A,\emptyset)$ recognizing L. □

The proof of this proposition will be given in Appendix.
Now we introduce some definitions.

Definition 4.6.
An asynchronous cellular automaton $\mathbb{A}=(A,S,I,\Delta,F)$ is unambiguous if for all traces $t',t''\in M(A,\theta)$ if their composition is recognized by $\mathbb{A}$, i.e. $t't''\in T(\mathbb{A})$ then there exists exactly one triple of states $(s_o,s',s'')\in I\times S\times F$ such that $s'\in\Delta(s_o,t')$ and $s''\in\Delta(s',t'')$. □

Definition 4.6 is equivalent with the classical definition of unambiguity but it seems more adequate if we consider trace languages.

Definition 4.7.
Let $\mathbb{A}=(A,S,I,\Delta,F)$ be an asynchronous cellular automaton. A state $s_a\in S_a$ of a player $a\in A$ is dead if for every global state $s\in S$ such that $s_a=\Pi_a(s)$ we have $\Delta(s,a)=\emptyset$.
By $\mathbf{dead}_a\subseteq S_a$ we shall denote the subset of all dead states of S_a.
$\mathbb{A}$ is dead-regular iff $F\subseteq\bigtimes_{a\in A}\mathbf{dead}_a$. □

Thus if a player $a\in A$ is in a dead state $s_a\in\mathbf{dead}_a$ then he cannot move any more, regardless of the states of the other players.

Let us examine the case when $\mathbb{A}$ is both safe and dead-regular. We say that a global state $s\in S$ is *dead* if $\forall a\in A$, $\Delta(s,a)=\emptyset$, i.e. none of the players can move at s.
Let $t'\in M(A,\theta)$ be a trace and let $s'\in S$ be a state such that $s'\in\Delta(s_o,t')$ for some initial state $s_o\in I$ of $\mathbb{A}$. If $\mathbb{A}$ is safe then there exists a trace r such that $\Delta(s,r)\in F$, i.e. $tr\in T(\mathbb{A})$. If $\mathbb{A}$ is also dead-regular then we can deduce that $alph(r)=\{\, a\in A : \Pi_a(s)\in S_a\setminus\mathbf{dead}_a \,\}$. Thus in this case if a state s is accessible and dead then it belongs to F and we can even assume that $F=\bigtimes_{a\in A}\mathbf{dead}_a$. Moreover, note that every player $a\in A$ can

verify locally if he will move any more or not by inspecting if he is in a dead state while in general, if $\mathbb{A}$ is not safe and dead-regular as for instance in Example 4.2, the players have no local criterion to decide when to stop their moves.
Now the main result can be presented.

Theorem 4.8.
Let $T \subseteq M(A,\theta)$ *be a recognizable trace language. Then there exists a finite asynchronous cellular automaton* $\mathbb{A}$ *that is safe, unambiguous, dead-regular , and recognizes* T. □

Thus even if it is not possible to implement safely and deterministicaly all recognizable trace languages it is always possible to do this safely and unambiguosly by means of a dead-regular automaton.

5. Appendix

Proof of Proposition 4.5 (hint)
The main idea is to include in the states of the players special labels (time-stamps) . By means of these time-stamps the player executing the last move can be found at any moment. We refer the reader to [5] when such a bounded sequential time-system is constructed. □

The full proof of the main Theorem 4.8 is long and for this reason it will be published elsewhere. But as we do not want to leave the reader without hints we present here the main ideas and point out encoutered difficulties.
The proof is carried on by induction on the number of players card(A).
From this moment we fix a player $a \in A$ and we set $A' = A \setminus \{a\}$, $\theta' = \theta \cap (A' \times A')$.
Let $GT_a = \{ r \in M(A,\theta) : |r|_a = 0 \ \& \ \forall t \in M(A,\theta) \setminus \{1\},\ t \ll r \implies \neg(t \theta a) \}$.
The trace language GT_a is recognizable. Moreover, for every trace $r \in M(A,\theta)$ there exists a unique sequence of traces $(r_0,\ldots,r_n)$, denoted by $fact_a(r)$, such that
(i) $r = r_0 a r_1 a \ldots a r_n$,
(ii) $|r_0|_a = 0$,
(iii) $\forall i \in [1,n],\ r_i \in GT_a$.

Note that since for every r_i in $fact_a(r)$ we have $|r_i|_a=0$ the traces r_i can be considered as elements of the monoid $M(A',\theta')$. Intuitively, the trace r_i , $i\in[1,n-1]$, contains the moves that *follow strictly* the i-th move of the player a but *do not follow* the $i+1$st move of a. The suffix r_n contains the moves that strictly follow the last move of a. Finally, the prefix r_0 contains all the other moves. Note that the moves in r_i precede or may be *concurrent* with the $i+1$st move of a, $i\in[0,n-1]$.
The main step in the construction is provided by the following proposition , which shows how a recognizable trace language is coded by a regular sequential language.

Proposition 5.1
Let $T\subseteq M(A,\theta)$ be a recognizable trace language and $a\in A$ be the fixed player. There exist finite alphabets G and H , a recognizable language $L_T\subseteq GH^*$, and a mapping $\varphi : G\cup H \longrightarrow \mathcal{P}(M(A',\theta'))$ such that

(i) $\forall\, e\in G\cup H$, $\varphi(e)$ *is a recognizable trace language*

(ii) *for every* $t\in T$, $fact_a(t)=(t_0,t_1,\ldots,t_n)$ *there exists exactly one word* $x\in L_T$, $x=e_0e_1\ldots e_n$, *such that* $t_0\in\varphi(e_0)$, $t_1\in\varphi(e_1),\ldots,t_n\in\varphi(e_n)$, (*we say that this* x *codes* t)

(iii) *let* $x=e_0e_1\ldots e_n\in L_T$ *be any element of* L_T *and let* $(t_0,t_1,\ldots,t_n)$ *be any sequence of traces such that* $\forall\, i\in[0,n]$, $t_i\in\varphi(e_i)$. *Then there exists a trace* $t\in T$ *such that* $(t_0,t_1,\ldots,t_n)=fact_a(t)$. □

Let L_T, G,H,φ be as in Proposition 5.1. Since for $e\in G\cup H$ $\varphi(e)$ is a recognizable trace language and $\varphi(e)\subseteq M(A',\theta')$, by the induction hypothesis there exists a safe, unambigouos, dead-regular finite asynchronous cellular automaton $\mathbb{A}^e$ over (A',θ') recognizing $\varphi(e)$.
Let $\mathcal{A}=(G\cup H,Q,q_0,\delta,F)$ be a safe, deterministic finite automaton recognizing L_T (as we have mentioned, this automaton can be obtained from any deterministic automaton recognizing L_T by removing the states that are not coaccessible).
Let $t\in T$, $fact_a(t)=(t_0,t_1,\ldots,t_n)$ and let $x=e_0e_1\ldots e_n\in L_T$ be the word in L_T coding t. At the beginning the player a is in the state $q_1=\delta(q_0,e_0)$ and the players from A' have the automaton $\mathbb{A}^e0$ switched on , which enables them to execute the trace t_0. In general, for $i\in[1,n]$, the player a making its i-th move choses $e_i\in H$, changes its state to $q_{i+1}=\delta(q_i,e_i)$ using the automaton $\mathcal{A}$, and , finally, switches on the automaton $\mathbb{A}^e i$ (recognizing $\varphi(e_i)$) for all the other players.

We see that intuitively the execution of the trace t_i, $i\in[1,n]$, is controlled by the i-th move of the player α while the execution of t_0 is controlled by the choice of the initial state.

Let us examine the following situation. We want to execute the trace $t\in T$ and we have already executed its prefix u, $u\ll t$. Let $fact_\alpha(u)=(u_0,u_1,\ldots,u_m)$. Then for every $i\in[0,m]$, $u_i\ll t_i$ and the automaton $\mathbb{A}^e i$ has been switched on and it has executed the prefix u_i. If $u_i\neq t_i$ then $\mathbb{A}^e i$ is still active, it has not yet arrived to its final state. As it turns out, at the given moment not only the last switched on automaton $\mathbb{A}^e m$ may be active but some preceding automata may be active as well. Fortunately, the number of these automata is bounded by a constant. Now it remains to coordinate the work of the active automata, which , we admit ,is the most difficult part of the construction.

References

[1] R.Cori, Y.Metivier, Approximation of a trace, asynchronous automata and the ordering of events in a distributed system, 15 ICALP, LNCS 317,147-161, Springer Verlag, 1988

[2] R.Cori,Y.Metivier,W.Zielonka, Applications of trace approximation to asynchronous automata and message passing with bounded time-stamps in asynchronous networks, in preparation

[3] C.Duboc, Mixed product and asynchronous automata, Theor. Comp. Sci.,48(1986),183-199

[4] S.Eilenberg, Automata, Languages and Machines,vol. A, 1974, Academic Press

[5] Amos Israeli, Ming Li, Bounded Time-Stamps, Proceedings of 28th Annual FOCS Symposium ,1987,371-382

[6] G.Lallement, Semigroups and Combinatorial Applications,1979, J.Wiley and Sons

[7] A.Mazurkiewicz, Concurrent Program Schemes and Their Interpretations,DAIMI-PB-78,Aarchus University,1977

[8] Y.Metivier, An algorithm for computing asynchronous automata in the case of acyclic non-commutation graphs, 14 ICALP,LNCS 267

[9] E.Ochmański, Regular Behaviour of Concurrent Schemes, EATCS Bulletin, vol.27,1985,56-67

[10] W.Zielonka, Notes on Finite Asynchronous Automata, RAIRO Inf. Theor. et Appl.,21(2),1987,99-135