Distributed Asynchronous Automata

Nicolas Baudru

Laboratoire d'Informatique Fondamentale de Marseille — eRISCS group
Aix-Marseille Universités, 163, avenue de Luminy, F-13288 Marseille Cedex 9, France
nicolas.baudru@lif.univ-mrs.fr

Abstract. Asynchronous automata are a model of communication processes with a distributed control structure, global initializations and global accepting conditions. The well-known theorem of Zielonka states that they recognize exactly the class of regular Mazurkiewicz trace languages. In this paper we study the particular case of *distributed* asynchronous automata, which require that the initializations and the accepting conditions are distributed as well: every process chooses an initial *local* state and stops in a final *local* state independently from each other. We characterize effectively the regular trace languages recognized by these automata. Also, we present an original algorithm to build, if it is possible, a non-deterministic distributed asynchronous automaton that recognizes a given regular trace language. Surprisingly, this algorithm yields a new construction for the more general problem of the synthesis of asynchronous automata from regular trace languages that subsumes all existing ones in terms of space complexity.

Introduction

Asynchronous automata [17] modelize concurrent systems that use a mechanism based on shared variables to communicate. They consist of a set of processes with a distributed control structure, global initializations and global accepting conditions. During an execution the processes synchronize on shared variables, which are called simply actions in our setting: all actions a are associated with a subset of processes which agree jointly on a move on reading a. On the other hand, the theory of Mazurkiewicz traces [4] provides mathematical tools for the formal study of concurrent systems. In this theory, the actions of a concurrent system are equipped with an independent relation between actions that do not share any process.

One of the major contributions in the theory of Mazurkiewicz traces characterizes regular trace languages by means of asynchronous automata [17]. This result, known as Zielonka's theorem, and related techniques are fundamental tools in concurrency theory. For instance they are useful to compare the expressive power of classical models of concurrency such as Petri nets, asynchronous systems, and concurrent automata [16,12]. These methods have been also adapted to the construction of communicating finite-state machines from regular sets of message sequence charts [8,1,7].

For twenty years, several constructive proofs of Zielonka's theorem have been developed. All these constructions, which build asynchronous automata from regular trace languages, are quite involved and yield an exponential explosion of the number of states in each process [6,13]. To our knowledge, the complexity of this problem is still unknown, and it was asked in [5] whether a simpler construction could be designed.

In this paper, we are interested in the particular case of *distributed* asynchronous automata (DAA for short). In a DAA, the initializations and the accepting conditions are also distributed: each process chooses a local initial state and stops in a local final state independently from other processes. We introduce them as an interesting tool to develop alternative proofs of Zielonka's result. In particular, we present a technique based on simple compositions/decompositions of DAAs that results in the construction of a "small" non-deterministic asynchronous automaton from any regular trace language given by means of a trace automaton $\mathcal{A}$. The size of each process built by our method is then polynomial in the size of $\mathcal{A}$ and only exponential in the number of processes and actions of the system. So our method reduces significantly the explosion of the number of states in each process

Contrary to asynchronous automata, non-deterministic DAAs and deterministic DAAs do not recognize the same class of trace languages. Moreover, some regular trace languages are recognized by no DAA. Therefore we characterize the trace languages that correspond precisely to the behaviours of non-deterministic DAAs: these are the ones that satisfy the condition of *distributivity*. We explain how to verify whether or not a regular trace language is distributed. If so, we show that our method gives directly a DAA recognizing this language. However it is still an open question to decide whether a regular trace language is recognizable by a deterministic DAA .

Overview of the Paper. The basic notions and definitions are presented in Section 1. Section 2 introduces the model of distributed asynchronous automata and a short comparison with the classical asynchronous automata. In particular, we show that these two models are not equivalent. In Section 3, a tool is introduced to compose hierarchically DAAs into a *high-level* DAA. The key results presented in this section will be used all along the paper. Then we define in Section 4 the concept of *roadmaps* and their associated *located* trace languages. Our first main result, Theorem 4.4, states that any located trace language is recognized by some DAA. We explain how to build this DAA by using high-level DAAs and present a first complexity result about our construction at the end of this section. Also, Theorem 4.4 yields a new construction for the synthesis of classical asynchronous automata that subsumes all existing ones in terms of complexity. This construction and a comparison with related works are presented in Section 5. Finally, Section 6 contains the second main result of the paper: Theorem 6.5 shows that distributed regular trace languages correspond precisely to the behaviours of non-deterministic distributed asynchronous automata.

1 Background

1.1 Mazurkiewicz Traces

In this paper we fix a finite alphabet Σ whose elements are called *actions*. A *word* u over Σ is a sequence of actions of Σ; the empty word is denoted by ε. The *alphabet* $\mathrm{alph}(u)$ of a word $u \in \Sigma^\star$ consists of all actions that appear in u. It is inductively defined by $\mathrm{alph}(\varepsilon) = \emptyset$ and $\mathrm{alph}(ua) = \mathrm{alph}(u) \cup \{a\}$ for all $u \in \Sigma^\star$ and all $a \in \Sigma$.

In addition to Σ, we fix an *independence relation* I over Σ, that is, a binary relation $\mathrm{I} \subseteq \Sigma \times \Sigma$ that is irreflexive and symmetric. We also define the *dependence relation* D as the complementary relation of I: $\mathrm{D} = \Sigma \times \Sigma \setminus \mathrm{I}$. The *trace equivalence* $\sim$ associated with the *independence alphabet* (Σ, I) is the least congruence over the free monoid $\Sigma^\star$ such that $ab \sim ba$ for all pairs of independent actions $a\mathrm{I}b$. For a word $u \in \Sigma^\star$, the *(Mazurkiewicz) trace* $[u]$ collects all words that are equivalent to u: $[u] = \{v \in \Sigma^\star \mid v \sim u\}$. We extend this notation to sets of words: for all $L \subseteq \Sigma^\star$, $[L] = \{v \in \Sigma^\star \mid \exists u \in L : v \sim u\}$. Finally a set of words L is called a *trace language* if $[L] = L$.

1.2 Trace Automata

An *automaton* $\mathcal{A}$ is a quadruple $(Q, \to, I, F)$ where Q is a finite set of states, $\to \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ is a set of transitions, and $I, F \subseteq Q$ are respectively a subset of initial states and a subset of final states. We write $q \xrightarrow{a} q'$ to denote $(q, a, q') \in \to$. Then the automaton $\mathcal{A}$ is *deterministic* if it satisfies the three next conditions: I is a singleton; if $q \xrightarrow{\varepsilon} q'$ then $q = q'$; if $q \xrightarrow{a} q'$ and $q \xrightarrow{a} q''$ then $q' = q''$.

An *execution* of $\mathcal{A}$ of length $n \geq 1$ is a sequence of transitions $(q_i \xrightarrow{a_i} q'_i)_{i \in [1,n]}$ such that $q'_i = q_{i+1}$ for each $i \in [1, n-1]$. An execution of length 0 is simply a state of $\mathcal{A}$. For any word $u \in \Sigma^\star$, we write $q \xrightarrow{u} q'$ if there is some execution $(q_i \xrightarrow{a_i} q'_i)_{i \in [1,n]}$ such that $q = q_1$, $q' = q'_n$ and $u = a_1 \cdots a_n$. The language $L(\mathcal{A})$ accepted by an automaton $\mathcal{A}$ consists of all words $u \in \Sigma^\star$ such that $q \xrightarrow{u} q'$ for some $q \in I$ and some $q' \in F$. A set of words $L \subseteq \Sigma^\star$ is *regular* if it is accepted by some automaton.

Definition 1.1. *An automaton $\mathcal{A} = (Q, \to, I, F)$ is called a* trace automaton *if for all states $q, q' \in Q$ and all words $u, v \in \Sigma^\star$ such that $u \sim v$, $q \xrightarrow{u} q'$ implies $q \xrightarrow{v} q'$.*

Clearly the language accepted by a trace automaton is a regular trace language. Conversely for any regular trace language L the minimal deterministic automaton that accepts L is a trace automaton.

1.3 Synthesis of Asynchronous Automata

We present next the model of asynchronous automata [17]. At first we introduce some additional notations. A finite family $\delta = (\Sigma_p)_{p \in \mathcal{P}}$ of subsets of Σ is called a *distribution* of (Σ, I) if we have $\mathrm{D} = \bigcup_{p \in \mathcal{P}} (\Sigma_p \times \Sigma_p)$. We fix an arbitrary distribution $\delta = (\Sigma_p)_{p \in \mathcal{P}}$ in the rest of this paper and call *processes* the elements of $\mathcal{P}$. The *location* $\mathrm{Loc}(a)$ of an action $a \in \Sigma$ consists of all processes $p \in \mathcal{P}$ such that $a \in \Sigma_p$. We extend this notation to set of actions $T \subseteq \Sigma$ and to words $u \in \Sigma^\star$ in a natural way: $\mathrm{Loc}(T) = \bigcup_{a \in T} \mathrm{Loc}(a)$ and $\mathrm{Loc}(u) = \mathrm{Loc}(\mathrm{alph}(u))$.

Definition 1.2. *An* asynchronous automaton *(*AA *for short) $\mathcal{S}$ consists of*

- *a family of local states $(Q_p)_{p \in \mathcal{P}}$;*
- *a set of initial global states $I \subseteq \prod_{p \in \mathcal{P}} Q_p$;*
- *a set of final global states $F \subseteq \prod_{p \in \mathcal{P}} Q_p$;*
- *a family of mute transitions $(\tau_p)_{p \in \mathcal{P}}$ where $\tau_p \subseteq Q_p \times Q_p$ for all $p \in \mathcal{P}$;*
- *and a family of transition relations $(\partial_a)_{a \in \Sigma}$ where for all $a \in \Sigma$,*

$$\partial_a \subseteq \prod_{p \in \mathrm{Loc}(a)} Q_p \times \prod_{p \in \mathrm{Loc}(a)} Q_p$$

We stress here that the mute transitions have been introduced to simplify the different constructions presented throughout this paper. They can be removed without loss of generality and without increasing the number of local states. For each process $p \in \mathcal{P}$, we define $\tau_p^\star$ as the reflexive and transitive closure of τ_p: for all $q_0, q \in Q_p$, $(q_0, q) \in \tau_p^\star$ iff $q_0 = q$ or else there exists a sequence of states $q_1, \ldots, q_n$ such that $(q_i, q_{i+1}) \in \tau_p$ for all $0 \leq i < n$ and $q_n = q$.

The *global automaton* $\mathcal{A}_\mathcal{S}$ of $\mathcal{S}$ is the automaton $(Q, \rightarrow, I, F)$ where Q is the set of global states $\prod_{p\in\mathcal{P}} Q_p$ and the set of global transitions $\rightarrow \subseteq Q \times (\Sigma \cup \{\varepsilon\}) \times Q$ is defined by the two next rules: for all $a \in \Sigma$, $(x_p)_{p\in\mathcal{P}} \xrightarrow{a} (y_p)_{p\in\mathcal{P}}$ if $x_p = y_p$ for all $p \notin \mathrm{Loc}(a)$ and $((x_p)_{p\in\mathrm{Loc}(a)}, (y_p)_{p\in\mathrm{Loc}(a)}) \in \partial_a$; $(x_p)_{p\in\mathcal{P}} \xrightarrow{\varepsilon} (y_p)_{p\in\mathcal{P}}$ if there is some $p \in \mathcal{P}$ such $(x_p, y_p) \in \tau_p$ and $x_k = y_k$ for all $k \neq p$. For convenience, an execution of the global automaton $\mathcal{A}_\mathcal{S}$ of $\mathcal{S}$ is simply called an execution of $\mathcal{S}$.

The language recognized by an asynchronous automaton $\mathcal{S}$ is simply the language of its global automaton $\mathcal{A}_\mathcal{S}$: $L(\mathcal{S}) = L(\mathcal{A}_\mathcal{S})$. Notice that $\mathcal{A}_\mathcal{S}$ is a trace automaton. Then $L(\mathcal{S})$ is a regular trace language. In [17], Zielonka shows that any regular trace language is recognized by an asynchronous automaton. Furthermore, this asynchronous automaton is *deterministic*, which means its global automaton is deterministic.

Theorem 1.3 (Zielonka [17]). *For all trace automata $\mathcal{A}$ there exists a deterministic asynchronous automaton $\mathcal{S}$ such that $L(\mathcal{S}) = L(\mathcal{A})$.*

2 Distributed Asynchronous Automata

Distributed asynchronous automata (DAA for short) differ slightly from asynchronous automata since we consider a *local* definition of initial and final states rather than a *global* one. We will show in Example 2.2 that the two models are not semantically equivalent: some regular trace languages are not recognizable by distributed asynchronous automata.

Definition 2.1. *An asynchronous automaton $((Q_p)_{p\in\mathcal{P}}, I, F, (\tau_p)_{p\in\mathcal{P}}, (\partial_a)_{a\in\Sigma})$ is* distributed *if there exist two families $(I_p)_{p\in\mathcal{P}}$ and $(F_p)_{p\in\mathcal{P}}$ such that $I_p, F_p \subseteq Q_p$ for all $p \in \mathcal{P}$, $I = \prod_{p\in\mathcal{P}} I_p$ and $F = \prod_{p\in\mathcal{P}} F_p$.*

For convenience we will denote a DAA $\mathcal{H}$ as $((Q_p, I_p, F_p, \tau_p)_{p\in\mathcal{P}}, (\partial_a)_{a\in\Sigma})$ where I_p and F_p represent the set of initial local states and the set of final local states of the process p, respectively. For next purpose, we also define the notion of *local executions* of $\mathcal{H}$. Let p be a process. Then a local execution of $\mathcal{H}$ for p is a sequence $(q_i, a_i, q'_i)_{i\in[1,n]}$ of tuples of $Q_p \times (\Sigma_p \cup \{\varepsilon\})) \times Q_p$ that satisfies the following statements: $q_1 \in I_p$; $q'_n \in F_p$; $q'_i = q_{i+1}$ for all $i \in [1, n-1]$; and for all $i \in [1, n]$, either $(q_i, q'_i) \in \tau_p$ and $a_i = \varepsilon$, or there is $((x_k)_{k\in\mathcal{P}}, (y_k)_{k\in\mathcal{P}}) \in \partial_{a_i}$ with $x_p = q_i$ and $y_p = q'_i$. For any word $u \in \Sigma_p^\star$, we write $q \overset{u}{\rightsquigarrow}_p q'$ if there is some local execution $(q_i, a_i, q'_i)_{i\in[1,n]}$ for p such that $q_1 = q$, $q'_n = q'$ and $a_1 \cdots a_n = u$.

Example 2.2. Consider the regular trace language $L = [aab] \cup [abb]$ over the alphabet $\{a, b\}$ with $a\mathrm{I}b$. L is recognized by the asynchronous automaton that consists of two processes p_a and p_b with $\Sigma_{p_a} = \{a\}$, $\Sigma_{p_b} = \{b\}$, $Q_{p_a} = \{q_a, q'_a, q''_a\}$, $Q_{p_b} =$

$\{q_b, q'_b, q''_b\}$, $I = \{(q_a, q'_b), (q'_a, q_b)\}$, $F = \{(q''_a, q''_b)\}$, $\partial_a = \{(q_a, q'_a), (q'_a, q''_a)\}$ and $\partial_b = \{(q_b, q'_b), (q'_b, q''_b)\}$. However L is recognized by no DAA. Indeed, suppose for the sake of contradiction that such a DAA $\mathcal{H}$ exists, that is $L(\mathcal{H}) = L$. Then $\mathcal{H}$ consists of two processes p_a and p_b. Since $L = [aab] \cup [abb]$ and $a\mathrm{I}b$, p_a is able to locally perform a and aa, and p_b is able to locally perform b and bb. Because initial and final states are local in a DAA, the process p_a carries out a or aa independently of the choice of the process p_b. Then the words ab and $aabb$ belong to $L(\mathcal{H})$. This contradicts $L(\mathcal{H}) = L$.

This example shows that some regular trace languages are not recognizable by DAAs. Since DAAs are a particular case of AAs, and because Theorem 1.3 states that any regular trace language is recognized by AAs, we conclude that DAAs are less expressive than AAs. Noteworthy the expressive power of asynchronous automata does not change if only the initializations or else only the accepting conditions are distributed.

Another interesting remark follows from Theorem 1.3. Let L be a regular trace language. Then L is recognized by some AA $\mathcal{S} = ((Q_p)_{p\in\mathcal{P}}, I_{\mathcal{S}}, F_{\mathcal{S}}, (\tau_p)_{p\in\mathcal{P}}, (\partial_a)_{a\in\Sigma})$. For each pair $(\imath, f)$ of initial and final states of $I_{\mathcal{S}} \times F_{\mathcal{S}}$, we can build another AA $\mathcal{S}_{\imath,f}$ that is identical to $\mathcal{S}$ except that the set of initial states and the set of final states are restricted to the singletons $\{\imath\}$ and $\{f\}$, respectively. Clearly, these new AAs are DAAs as well. Then the next corollary holds:

Corollary 2.3. *Let L be a regular trace language. There exists a finite set of* DAA*s $\mathcal{H}_1, \ldots, \mathcal{H}_n$ such that $L = \bigcup_{i\in[1,n]} L(\mathcal{H}_i)$.*

Thus, any regular trace language is the finite union of the languages of DAAs. The expressive power of DAAs will be studied in Section 6.

3 High-level Distributed Asynchronous Automata

In this section, we introduce a natural operation to compose DAAs similar to HMSC [8]. We show that the language of the DAA associated to a high-level DAA can be expressed in terms of the languages of its DAA components under some assumptions. As example, we show that some natural and intuitive operations over DAAs like concatenation and choice can be seen as particular high-level DAAs. The different results of this section constitute the heart of this paper. They will be used in Sections 4 and 6.

A *high-level* DAA $\mathcal{G}$ is a structure (V, E, I, F, Ψ) where V is a finite and nonempty set of vertices, $E \subseteq V \times V$ is a set of edges and $I, F \subseteq V$ are respectively a set of initial vertices and a set of final vertices. The function Ψ maps all vertices $v \in V$ to a DAA $\Psi(v)$, which is denoted by $((Q_{v,p}, I_{v,p}, F_{v,p}, \tau_{v,p})_{p\in\mathcal{P}}, (\partial_{v,a})_{a\in\Sigma})$. Moreover, we require that the sets of states $Q_{v,p}$ and $Q_{v',p}$ are disjoint for all $v, v' \in V$ and all $p \in \mathcal{P}$. A *path* π of $\mathcal{G}$ is a sequence $(v_0, \ldots, v_n)$ such that $v_0 \in I$, $v_n \in F$ and $(v_{i-1}, v_i) \in E$ for $i \in [1, n]$. We denote by $\Pi_{\mathcal{G}}$ the set of all paths of $\mathcal{G}$. We say that $\mathcal{G}$ has *no self-loop* if $(v, w) \in E \Rightarrow v \neq w$. This assumption will be useful for Proposition 3.5 to avoid some synchronization problems as presented in Remark 3.6.

Now we define the DAA $\langle\mathcal{G}\rangle$ associated with a high-level DAA $\mathcal{G}$. Roughly speaking, $\langle\mathcal{G}\rangle$ consists of putting the DAAs $\Psi(v)$ all together, and adding mute transitions from their final local states to their initial local states in accordance with the edges of $\mathcal{G}$.

Definition 3.1. *The* DAA $\langle\mathcal{G}\rangle = ((Q_p, I_p, F_p, \tau_p)_{p\in\mathcal{P}}, (\partial_a)_{a\in\Sigma})$ *associated with a high-level* DAA $\mathcal{G} = (V, E, I, F, \Psi)$ *is defined by*

- $Q_p = \bigcup_{v\in V} Q_{v,p}$ *for each process* $p \in \mathcal{P}$,
- $I_p = \bigcup_{v\in I} I_{v,p}$ *for each process* $p \in \mathcal{P}$,
- $F_p = \bigcup_{v\in F} F_{v,p}$ *for each process* $p \in \mathcal{P}$,
- $\partial_a = \bigcup_{v\in V} \partial_{v,a}$ *for each action* $a \in \Sigma$,
- $\tau_p = \bigcup_{v\in V} \tau_{v,p} \cup \bigcup_{(v,w)\in E}(F_{v,p} \times I_{w,p})$ *for each process* $p \in \mathcal{P}$.

To illustrate this definition, we pay attention to the particular case of a high-level DAA with two vertices and one edge, which leads to the definition of the concatenation of two DAAs. The related Proposition 3.3 will be useful for the proof of Proposition 3.5.

Definition 3.2. *Let* $\mathcal{H}_1$ *and* $\mathcal{H}_2$ *be two* DAA*s and let* $\mathcal{G}$ *be the high-level distributed asynchronous automaton* (V, E, I, F, Ψ) *where* $V = \{v_1, v_2\}$, $(v_1, v_2) \in E$, $I = \{v_1\}$, $F = \{v_2\}$, $\Psi(v_1) = \mathcal{H}_1$ *and* $\Psi(v_2) = \mathcal{H}_2$. *Then the* concatenation $\mathcal{H}_1 \odot \mathcal{H}_2$ *is* $\langle\mathcal{G}\rangle$.

Intuitively, $\mathcal{H}_1 \odot \mathcal{H}_2$ behaves as follows: each process p of $\mathcal{H}_1 \odot \mathcal{H}_2$ starts to behave like the process p of $\mathcal{H}_1$ until it reaches a final local state of $\mathcal{H}_1$. Thereafter it behaves like the process p of $\mathcal{H}_2$. So, it should be clear that the following proposition holds:

Proposition 3.3. $L(\mathcal{H}_1 \odot \mathcal{H}_2)$ *is the trace language* $[L(\mathcal{H}_1) \cdot L(\mathcal{H}_2)]$.

We come to the key result of the paper. The next proposition shows that, under some assumptions, the language of the DAA associated with a high-level DAA $\mathcal{G}$ can be expressed in terms of the languages of the DAA components of $\mathcal{G}$. This result requires that $\mathcal{G}$ has no self-loop and relies on the synchronization condition defined below. The latter uses the notion of local executions introduced in Section 2. We recall that local executions always start in an initial local state and end in a final local state.

Definition 3.4. *Let* $T \subseteq \Sigma$. *A* DAA $\mathcal{H}$ *is* T-synchronizable *if for all processes* $p \in \mathcal{P}$ *and all local executions* $q \overset{u}{\rightsquigarrow}_p q'$ *of* $\mathcal{H}$, *we have* $(\Sigma_p \cap T) = \mathrm{alph}(u)$.

For a distributed asynchronous automaton $\mathcal{H}$, the T-synchronization condition will imply the next fact: if the dependence graph (T, D) is connected, then all processes of $\mathrm{Loc}(T)$ always synchronize with each other along *any* execution of $\mathcal{H}$ that leads its processes from local initial states to final ones. Thus, whenever we compose several T-synchronizable DAAs in a high-level DAA $\mathcal{G}$, all processes of $\mathrm{Loc}(T)$ are forced to travel along the same sequence of DAA components. Consequently, the behaviours of the associated DAA $\langle\mathcal{G}\rangle$ can easily be characterized as unions and compositions of the languages of the DAA components.

Proposition 3.5. *Let* $\mathcal{G}$ *be a high-level* DAA *with* Ψ *as labelling function and* $T \subseteq \Sigma$. *If* $\mathcal{G}$ *has no self-loop,* Ψ *maps all vertices of* $\mathcal{G}$ *to* T*-synchronizable* DAA*s and the dependence graph* (T, D) *is connected, then* $\langle\mathcal{G}\rangle$ *is a* T*-synchronizable* DAA *that recognizes*

$$L(\langle\mathcal{G}\rangle) = \bigcup_{(v_0,\ldots,v_n)\in\Pi_{\mathcal{G}}} [L(\Psi(v_0)) \cdot \ldots \cdot L(\Psi(v_n))] \ .$$

Proof. Let $\mathcal{G} = (V, E, I, F, \Psi)$ be a high-level DAA. First, $\langle \mathcal{G} \rangle$ is T-synchronizable because any process has to travel along at least one T-synchronizable DAA component to reach a final local state from an initial local state. Then the processes of $\mathcal{P} \setminus \mathrm{Loc}(T)$ take part in mute transitions of $\langle \mathcal{G} \rangle$ only. For this reason, we suppose without loss of generality than $\mathcal{P} = \mathrm{Loc}(T)$. It is not hard to prove that $\bigcup_{(v_0,\dots,v_n)\in \Pi_{\mathcal{G}}} [L(\Psi(v_0)) \cdot \ldots \cdot L(\Psi(v_n))] \subseteq L(\langle \mathcal{G} \rangle)$. So, we prove only the backward inclusion. From now on, we fix an execution s of $\langle \mathcal{G} \rangle$ that leads the global automaton $\mathcal{A}_{\langle \mathcal{G} \rangle}$ from some global initial state to some global final state and we let u the word yielded by s. Then $u \in L(\langle \mathcal{G} \rangle)$.

We start with some additional notations. Let p be any process of $\mathrm{Loc}(T)$. By construction of $\langle \mathcal{G} \rangle$, along s, there are some vertices $v_1, \dots, v_n \in V^\star$ such that: p starts from some initial state $\imath_1 \in I_{v_1,p}$ from which it reaches some state $f_1 \in F_{v_1,p}$ *by using exclusively* $\tau_{v_1,p}$ *or* $(\partial_{v_1,a})_{a\in\Sigma_p}$ *transitions*; then an added τ_p-transition leads p from f_1 to some state $\imath_2 \in I_{v_2,p}$ from which it reaches some $f_2 \in F_{v_2,p}$ *by using exclusively* $\tau_{v_2,p}$ *or* $(\partial_{v_2,a})_{a\in\Sigma_p}$ *transitions*; and so on, until p reaches a local final state of $F_{v_n,p}$. We denote by $\nu_p(s)$ this sequence of vertices $(v_1, \dots, v_n)$ relied on p along s. In addition, for all $1 \leq i \leq n$ and all $a \in T$, we denote by $|s,p|_{i,a}$ the number of $\partial_{v_i,a}$ transitions in which p takes part when it goes from $\imath_i$ to f_i along s.

Two remarks are useful for the rest of this proof. Let $p \in \mathcal{P}$ and $\nu_p(s) = (v_1, \dots, v_n)$. First, it should be clear that $|s,p|_{i,a} > 0$ for all $a \in \Sigma_p \cap T$ and all $i \in [1,n]$ because all the $\Psi(v_i)$ are T-synchronizable. Secondly, two consecutive vertices of $\nu_p(s)$ are always distinct because there is no self-loop in $\mathcal{G}$: $v_i \neq v_{i+1}$ for all $1 \leq i < n$.

Let $a \in T$ and $p, k \in \mathrm{Loc}(a)$. Let $\nu_p(s) = (v_1, \dots, v_n)$ and $\nu_k(s) = (w_1, \dots, w_m)$ with $n \leq m$. We prove by contradiction that the following property (P) holds:

(P) for all $i \in [1,n]$, $v_i = w_i$ and $|s,p|_{i,a} = |s,k|_{i,a}$

Suppose that (P) fails. Then we denote by i the least integer of $[1,n]$ such that $v_i \neq w_i$ or $|s,p|_{i,a} \neq |s,k|_{i,a}$. Let $c = 1 + \sum_{j=1}^{i-1} |s,p|_{j,a}$. By hypothesis, $c = 1 + \sum_{j=1}^{i-1} |s,k|_{j,a}$ as well. This means that the c-th a-transition in which p takes part results from the transition relation $\partial_{v_i,a}$ whereas the c-th a-transition in which k takes part results from the transition relation $\partial_{w_i,a}$. This implies that $v_i = w_i$. Since we have supposed that (P) fails for i, we should have $|s,p|_{i,a} \neq |s,k|_{i,a}$. We prove it is impossible. Assume that $|s,p|_{i,a} < |s,k|_{i,a}$ and let $c' = c + |s,p|_{i,a}$. If $i = n$ then p takes part in $c' - 1$ a-transitions along s whereas k takes part in at least c' a-transitions, which is impossible because p and k must synchronize on all a-transitions. So $i < n$. However in this case the c'-th transition in which p takes part results from the transition relation $\partial_{v_{i+1},a}$ whereas the c'-th transition in which k takes part results from the transition relation $\partial_{w_i,a}$. Hence $v_i = w_i = v_{i+1}$, which is impossible because there is no self-loop in $\mathcal{G}$.

Since (P) holds for each $a \in T$ and since each process of $\mathrm{Loc}(a)$ takes part in the same number of a-transitions along s, we deduce that for all $p, k \in \mathrm{Loc}(a)$, $\nu_p(s) = \nu_k(s)$. Then, step by step, considering all actions of T together with similar arguments, we conclude that for all processes p, k of $\mathrm{Loc}(T)$ we have $\nu_k(s) = \nu_p(s)$ because (T, D) is connected. Consequently s is an execution of the DAA $\Psi(v_1) \odot \ldots \odot \Psi(v_n)$ as well. To conclude, Proposition 3.5 implies that $u \in [L(\Psi(v_1) \cdot \ldots \cdot L(\Psi(v_n)]$. ∎

Remark 3.6. We stress here the importance of the absence of self-loop in $\mathcal{G}$. Consider the $\{a, c\}$-synchronizable DAA $\mathcal{H}$ that consists of two processes p_1 and p_2 such that:

$\Sigma_{p_1} = \{a, c\}$ and $\Sigma_{p_2} = \{c\}$; $Q_{p_i} = \{q_i, q_i'\}$, $I_{p_i} = \{q_i\}$, $F_{p_i} = \{q_i'\}$ and $\tau_{p_i} = \emptyset$ for all $i \in \{1, 2\}$; $((q_1, q_2), (q_1, q_2')) \in \partial_c$ and $(q_1, q_1') \in \partial_a$. Clearly $\mathcal{H}$ accepts only the word ca. Now, consider the high-level DAA $\mathcal{G}$ that consists of a single vertex v, which is both initial and final, a single edge from v to v and a labelling function Ψ that maps v to $\mathcal{H}$. Then $\langle\mathcal{G}\rangle$ accepts the word cca, so that $L(\langle\mathcal{G}\rangle)$ cannot be expressed in terms of a rational expression that uses exclusively $L(\mathcal{H})$. The problem with the presence of self-loops is the following. In spite of the T-synchronization condition of $\mathcal{H}$, some processes (p_2 in our example) can travel along the DAA $\mathcal{H}$ several times, without the other processes (p_1 in our example) being informed of it. This kind of problem justifies why we require that a high-level DAA has no self-loop.

To illustrate Proposition 3.5, we introduce a second natural operation based on high-level DAA. Let $\mathcal{H}_1$ and $\mathcal{H}_2$ be two DAAs. The choice operation $\mathcal{H}_1 \oplus \mathcal{H}_2$ yields a DAA where each process p of $\mathcal{H}_1 \oplus \mathcal{H}_2$ can choose, independently from the other processes, to behave either like the process p of $\mathcal{H}_1$ or else like the process p of $\mathcal{H}_2$. Formally:

Definition 3.7. *The* choice $\mathcal{H}_1 \oplus \mathcal{H}_2$ *of* $\mathcal{H}_1$ *and* $\mathcal{H}_2$ *is the* DAA $\langle\mathcal{G}\rangle$ *where* $\mathcal{G}$ *consists of exactly two vertices, which are both initial and final, and no edges. As for the labelling function, it maps these vertices to* $\mathcal{H}_1$ *and* $\mathcal{H}_2$.

In this definition, $\mathcal{G}$ has no self-loop. So Proposition 3.5 can be applied as follows:

Corollary 3.8. *Let* $\mathcal{H}_1$ *and* $\mathcal{H}_2$ *be two* T*-synchronizable* DAAs*. If the dependence graph* (T, D) *is connected, then* $\mathcal{H}_1 \oplus \mathcal{H}_2$ *is* T*-synchronizable and recognizes* $L(\mathcal{H}_1) \cup L(\mathcal{H}_2)$.

4 Located Trace Languages

In this section we fix a trace automaton $\mathcal{A} = (Q, \rightarrow, I, F)$ over (Σ, I). We show in Theorem 4.4 that any *located* trace language of $\mathcal{A}$ (Definition 4.1) corresponds to the language of a synchronizable DAA.

An analogy can be drawn between our technique of proof and the technique used by McNaughton and Yamada in [10] in the framework of the free monoid. In this paper, the authors define languages $R_{i,j}^K$ that correspond to the set of words that label a path from the state $i \in Q$ to the state $j \in Q$ and that use the states in $K \subseteq Q$ as intermediate states. Then they use a recursive algorithm on the size of K to build rational expressions that correspond to the languages $R_{i,j}^K$. Here we have adapted this method for the trace monoid (Σ, I), that is, to care of concurrency. The defined languages are called *located trace languages*. They are based on *roadmaps* instead of intermediate states. Then we use a recursive algorithm on the size n of the roadmaps to build, by using high-level DAAs, the DAAs that correspond to the located trace languages on roadmaps of size n.

4.1 Roadmaps and Located Trace Languages

From now on, we fix a total order $\sqsubseteq$ over the actions of Σ. This order is naturally extended to sets of actions: for $T_1, T_2 \subseteq \Sigma$, $T_1 \sqsubseteq T_2$ if the least action in T_1 is smaller than the least action in T_2 w.r.t. $\sqsubseteq$.

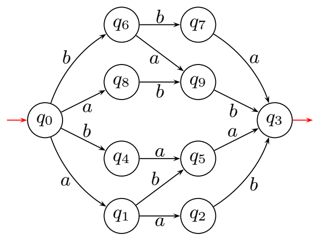

Fig. 1. $L(\mathcal{A})$ is recognized by no DAA

Let $T \subseteq \Sigma$. We denote by $\#\mathrm{cc}(T)$ the number of connected components of the dependence graph (T, D) and by $\mathrm{cc}(T)$ the sequence $(T_1, \ldots, T_{\#\mathrm{cc}(T)})$ of the $\#\mathrm{cc}(T)$ connected components of (T, D) sorted according to $\sqsubseteq$: $T_i \sqsubseteq T_{i+1}$ for all $1 \leq i < \#\mathrm{cc}(T)$. Note that $\#\mathrm{cc}(T) = 1$ if and only if (T, D) is connected.

Definition 4.1. *A* roadmap r *of the trace automaton* $\mathcal{A}$ *consists of a subset of actions* $T \subseteq \Sigma$ *together with a vector* $\boldsymbol{q}$ *of* $\#\mathrm{cc}(T) + 1$ *states of* $\mathcal{A}$*.*

The located trace language $L_r(\mathcal{A})$ *on a roadmap* $r = (T, \boldsymbol{q})$ *of* $\mathcal{A}$ *with* $\mathrm{cc}(T) = (T_1, \ldots, T_n)$ *and* $\boldsymbol{q} = (q_1, \ldots, q_{n+1})$ *comprises all the words contained in any trace* $[u_1 \cdots u_n]$ *such that* $\mathrm{alph}(u_i) = T_i$ *and* $q_i \xrightarrow{u_i} q_{i+1}$ *for all* $1 \leq i \leq n$, i.e.*:*

$$L_r(\mathcal{A}) = [\{u_1 \cdots u_n \in \Sigma^\star \mid \forall i \in [1, n], \mathrm{alph}(u_i) = T_i \wedge q_i \xrightarrow{u_i} q'_{i+1}\}].$$

Given a roadmap $r = (T, \boldsymbol{q})$, $\mathrm{start}(r)$ is the first component of $\boldsymbol{q}$, $\mathrm{stop}(r)$ is the last component of $\boldsymbol{q}$ and $\mathrm{alph}(r)$ is T. Then we denote by $\mathcal{R}(\mathcal{A})$ the set of all roadmaps of $\mathcal{A}$, and by $\mathcal{R}_f(\mathcal{A})$ the subset of roadmaps r of $\mathcal{R}(\mathcal{A})$ such that $\mathrm{start}(r) \in I$, $\mathrm{stop}(r) \in F$ and $L_r(\mathcal{A}) \neq \emptyset$.

Example 4.2. Consider the automaton $\mathcal{A}$ over $\{a, b\}$ with $a \mathrm{I} b$ and $a \sqsubseteq b$ depicted in Fig. 1, in which q_0 and q_3 are respectively the single initial state and the single final state. There are exactly two roadmaps r in $\mathcal{R}_f(\mathcal{A})$, that is, such that $\mathrm{start}(r) = q_0$, $\mathrm{stop}(r) = q_3$ and $L_r(\mathcal{A}) \neq \emptyset$: $r_1 = (\{a, b\}, q_0, q_2, q_3)$ and $r_2 = (\{a, b\}, q_0, q_8, q_3)$. The corresponding located trace languages are $L_{r_1}(\mathcal{A}) = [aab]$ and $L_{r_2}(\mathcal{A}) = [abb]$. We can check that $\mathcal{A}$ describes the trace language $[aab] \cup [abb]$ of Example 2.2.

As illustrated by Example 4.2, any regular trace language can be represented as the finite union of all the located trace languages on roadmaps of $\mathcal{R}_f(\mathcal{A})$:

Proposition 4.3. $L(\mathcal{A}) = \bigcup_{r \in \mathcal{R}_f(\mathcal{A})} L_r(\mathcal{A})$.

Now we turn to our first main result. The latter states that any located trace language is recognized by a DAA. Its constructive proof is detailed in Section 4.2.

Theorem 4.4. *Let* $\mathcal{A}$ *be a trace automaton and* r *be a roadmap of* $\mathcal{A}$*. There exists a* $\mathrm{alph}(r)$*-synchronizable* DAA *that recognizes* $L_r(\mathcal{A})$*.*

Note that Proposition 4.3 together with Theorem 4.4 give another proof of Corollary 2.3: any regular trace language can be seen as a finite union of languages of DAAs.

4.2 Constructive Proof of Theorem 4.4

We devote this subsection to the proof of Theorem 4.4. Our proof is constructive, that is, given a located trace language $L_r(\mathcal{A})$ of $\mathcal{A}$, we build a $\mathrm{alph}(r)$-synchronizable DAA and show that it recognizes L. To do this, we use an induction over the size of $\mathrm{alph}(r)$.

The base case is simple. Let r be a roadmap with $\mathrm{alph}(r) = \emptyset$. Either $L_r(\mathcal{A})$ is the empty language. Then it is recognized by the $\emptyset$-synchronized DAA where each process consists of one initial and non-final state and no transition. Or else $L_r(\mathcal{A})$ consists of the empty word ε only. So, it is recognized by the $\emptyset$-synchronized DAA where each process consists of one state, both initial and final, and all transition relations are empty.

We distinguish two inductive cases according to whether the dependence graph $(\mathrm{alph}(r), \mathrm{D})$ is connected or not.

Inductive Case 1. Let r be a roadmap of $\mathcal{R}(\mathcal{A})$ such that $\mathrm{alph}(r) \neq \emptyset$ and $(\mathrm{alph}(r), \mathrm{D})$ is unconnected. Let $r = (T, \boldsymbol{q})$ with $\mathrm{cc}(T) = (T_1, \ldots, T_n)$ and $\boldsymbol{q} = (q_1, \ldots q_{n+1})$. Then we set $r_i = (T_i, (q_i, q_{i+1}))$ for all $1 \leq i \leq n$. We can easily check that for all $1 \leq i \leq n$, r_i is a roadmap of $\mathcal{R}(\mathcal{A})$ and that $L_r(\mathcal{A}) = [L_{r_1}(\mathcal{A}) \cdot \ldots \cdot L_{r_n}(\mathcal{A})]$. By inductive hypothesis, for every $L_{r_i}(\mathcal{A})$ there exists a $\mathrm{alph}(r_i)$-synchronizable DAA $\mathcal{H}_i$ such that $L(\mathcal{H}_i) = L_{r_i}(\mathcal{A})$. Then Theorem 4.4 follows from Lemma 4.5. Note that Proposition 3.3 implies immediately that $L(\mathcal{H}_1 \odot \cdots \odot \mathcal{H}_n) = L_r(\mathcal{A})$.

Lemma 4.5. *$\mathcal{H}_1 \odot \cdots \odot \mathcal{H}_n$ is $\mathrm{alph}(r)$-synchronizable and recognizes $L_r(\mathcal{A})$.*

Inductive Case 2. Let r be a roadmap with $\mathrm{alph}(r) = T \neq \emptyset$ such that $(\mathrm{alph}(r), \mathrm{D})$ is connected. We denote by S_r the set of all roadmaps s such that $\mathrm{alph}(s) \subsetneq T$ and $\mathrm{stop}(s) = \mathrm{stop}(r)$. For $s \in S_r$, $\bar{s}$ denotes the roadmap $(T, \mathrm{start}(r), \mathrm{start}(s))$ and R_s consists of all the words u in $L_{\bar{s}}(\mathcal{A})$ that are also contained in some trace $[u_1 a_1 \cdots u_n a_n]$ such that $n \in \mathbb{N}$, $a_i \in T$ and $\mathrm{alph}(u_i) = T \setminus \{a_i\}$ for all $1 \leq i \leq n$: $R_s = L_{\bar{s}}(\mathcal{A}) \cap [\{u_1 a_1 \cdots u_n a_n \in \Sigma^\star \mid n \in \mathbb{N} \wedge \forall i \in [1, n], (a_i \in T \wedge \mathrm{alph}(u_i) \in T \setminus \{a_i\})\}]$. Clearly the following Lemma holds:

Lemma 4.6. $L_r(\mathcal{A}) = \bigcup_{s \in S_r} [R_s \cdot L_s(\mathcal{A})]$.

Then we express each R_s as a high-level DAA $\mathcal{G}_s = (V_s, E_s, I_s, F_s)$ defined below:

- $V_s = V \times \{0, 1\}$ where V is the set of all roadmaps v of $\mathcal{R}(\mathcal{A})$ such that $\mathrm{alph}(v) = T \setminus \{a_v\}$ for some action $a_v \in T$.
- for all $(v, i) \in V_s$, $(v, i) \in I_s$ if $\mathrm{start}(v) = \mathrm{start}(r)$.
- for all $(v, i) \in V_s$, $(v, i) \in F_s$ if $\mathrm{stop}(v) \xrightarrow{a_v} \mathrm{start}(s)$ is a transition of $\mathcal{A}$.
- for all $(v, i), (w, j) \in V_s$, $((v, i), (w, j)) \in E_s$ if $\mathrm{stop}(v) \xrightarrow{a_v} \mathrm{start}(w)$ is a transition of $\mathcal{A}$ and $i \neq j$.

It remains to define the labelling function Ψ_s. Let (v, i) be a vertex of $\mathcal{G}_s$. By definition of $\mathcal{G}_s$, $\mathrm{alph}(v)$ is the set $T \setminus \{a_v\}$ for some action $a_v \in T$. Clearly, the singleton $\{a_v\}$ is recognized by the $\{a_v\}$-synchronizable DAA $\mathcal{H}_{a_v}$ that consists of one state (both initial and final) for each process $p \notin \mathrm{Loc}(a_v)$, two states q_p (which is initial) and q'_p (which is final) for all processes $p \in \mathrm{Loc}(a_v)$, and one unique transition

$((q_p)_{p\in\mathrm{Loc}(a_v)}, (q'_p)_{p\in\mathrm{Loc}(a_v)}) \in \partial_{a_v}$. Then, using the inductive hypothesis on the located trace language $L_v(\mathcal{A})$, there is a $\mathrm{alph}(v)$-synchronizable DAA $\mathcal{H}_v$ that recognizes $L_v(\mathcal{A})$. Together with Proposition 3.3, we get a T-synchronizable DAA $\mathcal{H}_v \odot \mathcal{H}_{a_v}$ that recognizes $[L_v(\mathcal{A}) \cdot \{a_v\}]$. So, we set $\Psi_s(v,i) = \mathcal{H}_v \odot \mathcal{H}_{a_v}$.

Remark 4.7. At this step of the proof, we have the following facts: $\mathcal{G}_s$ as no self-loop because of the definition of E_s; each $\Psi(v,i)$ is a T-synchronizable DAA; we have assumed that the dependence graph (T, D) is connected. Then, all the hypotheses are satisfied to apply Proposition 3.5 on $\mathcal{G}_s$, which is useful to get the next result.

Lemma 4.8. *$\langle\mathcal{G}_s\rangle$ is $\mathrm{alph}(r)$-synchronizable and recognizes R_s.*

We come to the last step of the proof. First we recall that $L_r(\mathcal{A})$ describes the trace language $\bigcup_{s\in S_r}[R_s \cdot L_s(\mathcal{A})]$ (Lemma 4.6). Furthermore, for any $s \in S_r$, R_s is recognized by the T-synchronizable DAA $\langle\mathcal{G}_s\rangle$ (Lemma 4.8), and there is some $\mathrm{alph}(s)$-synchronizable DAA $\mathcal{H}_s$ that recognizes $L_s(\mathcal{A})$ (by the inductive hypothesis). Then Proposition 3.3 and Corollary 3.8 yield the next lemma from which Theorem 4.4 results.

Lemma 4.9. *$\oplus_{s\in S_r}(\langle\mathcal{G}_s\rangle \odot \mathcal{H}_s)$ is $\mathrm{alph}(r)$-synchronizable and recognizes $L_r(\mathcal{A})$.*

4.3 Complexity Analysis of the Construction

Theorem 4.4 shows that any located trace language is the language of some DAA. We explained in Subsection 4.2 how to build inductively this DAA. Here we analyse the complexity of our construction. In the next proposition, k denotes the number of processes in $\mathcal{P}$ and q denotes the number of states of the trace automaton $\mathcal{A}$.

Proposition 4.10. *Let $\mathcal{A}$ be a trace automaton, r be a roadmap of $\mathcal{A}$ with $|\mathrm{alph}(r)| = n$ and $\mathcal{H}_r$ be the DAA built by the algorithm described at Subsection 4.2. Then the number of local states in each process p of $\mathcal{H}_r$ is in $2^{O(n^2)}$ for q fixed, and $O(q^{2nk})$ for n fixed.*

Proof. For all roadmaps r of $\mathcal{A}$, $\mathcal{H}_r$ denotes the DAA inductively built by the construction presented along Subsection 4.2. We write $q_{r,p}$ to refer to the number of local states in the process p of $\mathcal{H}_r$. Then, $c_p(n)$ corresponds to the size of the biggest process p among all the DAAs $\mathcal{H}_r$ such that $|\mathrm{alph}(r)| \le n$: $c_p(n) = \max\{q_{r,p} \mid r \in \mathcal{R}(\mathcal{A}) \wedge |\mathrm{alph}(r)| \le n\}$. Clearly we have $c_p(n_1) \ge c_p(n_2)$ as soon as $n_1 \ge n_2$. We will compute an upper bound for $c_p(n)$. From now on, we fix a roadmap $r = (T, \boldsymbol{q}) \in \mathcal{R}(\mathcal{A})$ such that $|T| = n$ and a process $p \in \mathcal{P}$. We proceed in two cases according to whether the dependence graph (T, D) is connected.

Suppose that (T, D) is unconnected and let $\mathrm{cc}(T) = (T_1, \ldots, T_l)$. Clearly the number l of connected components of (T, D) is less than $\min\{n, k\}$. By Lemma 4.5, $\mathcal{H}_r = \mathcal{H}_{r_1} \odot \cdots \odot \mathcal{H}_{r_l}$ where each r_i is some roadmap with $\mathrm{alph}(r_i) = T_i \subsetneq T$. Therefore, we have $q_{r,p} \le nc_p(n-1)$.

Suppose now that (T, D) is connected. By Lemma 4.9, $\mathcal{H}_r = \oplus_{s\in S_r}(\langle\mathcal{G}_s\rangle \odot \mathcal{H}_s)$. Let $s \in S_r$. Then $\mathrm{alph}(s) \subsetneq T$, which implies that $|\mathrm{alph}(s)| < n$. Consequently, $c_p(|\mathrm{alph}(s)|) \le c_p(n-1)$. We now compute the number of local states in the process p of $\langle\mathcal{G}_s\rangle$. The latter is built from the graph $\mathcal{G}_s = (V_s, E_s, I_s, F_s, \Psi_s)$ where $V_s =$

$V \times \{0,1\}$ and V consists of all roadmaps v such that $\mathrm{alph}(v) = T \setminus \{a_v\}$ for some $a_v \in T$. Then V_s contains at most $2nq^k$ vertices. Each vertex v of $\mathcal{G}_s$ is labelled by the DAA $\mathcal{H}_v \odot \mathcal{H}_{a_v}$. The number of local states in the process p of $\mathcal{H}_v$ is at most $c_p(n-1)$ (by inductive hypothesis) and the one of $\mathcal{H}_{a_v}$ is at most 2. Consequently the number of local states in the process p of $\mathcal{H}_v \odot \mathcal{H}_{a_v}$ is at most $c_p(n-1)+2$ and the one of $\langle \mathcal{G}_s \rangle$ is at most $2nq^k(c_p(n-1)+2)$. Finally, we can compute the number of local states in the process p of $\mathcal{H}_r$: since the number of roadmaps in S_r is less than $2^n q^k$, we have that $q_{r,p} \leq 2^n q^k (2nq^k(c_p(n-1)+2) + c_p(n-1))$, that is, $q_{r,p} < n2^{n+3}q^{2k}c_p(n-1)$.

In both cases, for all roadmaps $r \in \mathcal{R}(\mathcal{A})$ such that $|\mathrm{alph}(r)| = n$, the number $q_{r,p}$ of local states in the process p of $\mathcal{H}_r$ is at most $n2^{n+3}q^{2k}c_p(n-1)$. In other words, $c_p(n) < n2^{n+3}q^{2k}c_p(n-1)$. Since $c_p(0) = 1$, we get $c_p(n) < n!2^{(n+4)(n+3)/2}q^{2kn}$. ■

5 Back to Zielonka's Theorem

Zielonka's theorem presented in Section 1 states that any regular trace automaton is recognized by an asynchronous automaton (AA). However, all known constructions of asynchronous automata from regular trace languages are quite involved and yield an exponential state explosion. In this section, we discuss how to apply Proposition 4.3 and Theorem 4.4 to get immediately a new algorithm for the construction of *non-deterministic* asynchronous automata that reduces significantly the state explosion.

Indeed, Theorem 4.4 yields for every $r \in \mathcal{R}_f(\mathcal{A})$ a DAA $\mathcal{H}_r$ that recognizes $L_r(\mathcal{A})$. We denote by $I_{r,p}$ and $F_{r,p}$ the set of initial local states and the set of final local states of the process p of $\mathcal{H}_r$, respectively. Now, consider the asynchronous automaton $\mathcal{S}$ that is identical to $\oplus_{r \in \mathcal{R}_f(\mathcal{A})} \mathcal{H}_r$ except that the set of *global* initial states is $I_\mathcal{S} = \bigcup_{r \in \mathcal{R}_f(\mathcal{A})} (\prod_{p \in \mathcal{P}} I_{r,p})$ and the set of *global* final states is $F_\mathcal{S} = \bigcup_{r \in \mathcal{R}_f(\mathcal{A})} (\prod_{p \in \mathcal{P}} F_{r,p})$. Then, by Proposition 4.3, it should be clear that $\mathcal{S}$ recognizes $L(\mathcal{A})$. This leads us to the next corollary where the complexity result follows from Proposition 4.10 together with the fact that $\mathcal{R}_f(\mathcal{A})$ contains at most $2^{|\Sigma|}|Q|^{|\mathcal{P}|}$ roadmaps.

Corollary 5.1. *Let $\mathcal{A}$ be a trace automaton over (Σ, I) with Q as set of states. There exists an asynchronous automaton $\mathcal{S}$ that recognizes $L(\mathcal{A})$ such that the number of local states in each process is in $2^{O(|\Sigma|^2)}$ for $|Q|$ fixed, and $O(|Q|^{2\cdot|\Sigma|\cdot|\mathcal{P}|+1})$ for $|\Sigma|$ fixed.*

Comparison with Existing Approaches. We mention the approaches that lead to deterministic AAs first. In [13] a complexity analysis of Zielonka's construction [17] is detailed. The number of local states $|Q_p|$ built by Zielonka's technique for each process $p \in \mathcal{P}$ is $|Q_k| \leq 2^{O(2^{|\mathcal{P}|}|Q|\log(|Q|))}$. The simplified construction by Cori et al. in [3] also suffers from this exponential state-explosion [4]. More recently in [6], Genest and Muscholl improve the construction of [17]. To do this, they add to this construction the concept of zones to reduce the amount of information to store in each process. This new algorithm, that yields a deterministic AA still, is thus exponential in $|Q|$ and $|\mathcal{P}|$.

As for the non-deterministic approaches, the construction of Pighizzini [15] build non-deterministic AAs from particular rational expressions. This simpler approach gives AAs whose number of local states in each process is exponential in the length of the rational expression. Another construction of non-deterministic asynchronous automata is presented in [2]. In this paper the number of local states built for each process is polynomial in $|Q|$ and double-exponential in $|\Sigma|$. In comparison, our construction builds non-deterministic AAs that are also polynomial in $|Q|$, but only exponential in $|\Sigma|$.

6 Expressive Power of Distributed Asynchronous Automata

In Corollary 2.3, we have shown that any regular trace language can be expressed as a finite union of languages of DAAs. However, as illustrated in Example 2.2, some of them are the language of no DAA. In this section, we characterize effectively in Theorem 6.5 those that correspond to the behaviours of non-deterministic DAAs: these are the ones that are *distributed*. Interestingly, for any given distributed regular trace language, the construction presented in Subsection 4.2 yields directly a DAA recognizing it.

Let L be a regular trace language and T be a nonempty subset of Σ. A word u is a *T-independent prefix* of L if there exists a word $uv \in L$ such that $\mathrm{alph}(u) = T$ and $\mathrm{Loc}(T) \cap \mathrm{Loc}(v) = \emptyset$. We denote by $L\|T$ the set of all T-independent prefixes of L. Then $L\|T$ is a regular trace language. Note that, for a DAA $\mathcal{H}$ recognizing L, the set $L\|T$ represents all the words u with $\mathrm{alph}(u) = T$ that can lead all the processes of $\mathrm{Loc}(T)$ to final local states, independently from the behaviours of the other processes.

The next definition characterizes the trace languages that are recognizable by DAAs.

Definition 6.1. *Let L be a regular trace language over (Σ, I). L is* distributed *if one of the two following conditions holds for all nonempty $T \subseteq \Sigma$ with $\mathrm{cc}(T) = (T_1, \ldots, T_n)$:*

C1: $\exists p \notin \mathrm{Loc}(T), \forall v \in L : p \in \mathrm{Loc}(v)$;
C2: $\prod_{i\in[1,n]} L\|T_i \subseteq L$.

Let L be a regular trace language that is not distributed. Then Conditions $C1$ and $C2$ fail for some subset of actions T with $\mathrm{cc}(T) = (T_1, \ldots T_n)$. Suppose that there exists a DAA $\mathcal{H}$ that recognizes L. Since $C2$ fails for T there are words $u_1 \in L\|T_1, \ldots, u_n \in L\|T_n$ such that $u = u_1 \cdots u_n \notin L$. For each $u_i \in L\|T_i$, the processes of $\mathrm{Loc}(T_i)$ can perform u_i and reach final local states. Then, all together, the processes of $\mathrm{Loc}(T)$ can perform the word u and reach final local states, and this because $\mathrm{Loc}(T_i)$ and $\mathrm{Loc}(T_j)$ are disjoint for all $i \neq j$, . Since Condition $C1$ fails for T, the other processes can reach final local states without producing any action. In conclusion, u belongs to $L(\mathcal{H})$ while it doesn't belong to L, which contradicts that $\mathcal{H}$ recognizes L. This leads immediately to the following lemma:

Lemma 6.2. *The language recognized by a DAA is distributed.*

Example 6.3. Consider again the regular trace language $L = [aab] \cup [abb]$ of Example 2.2 that is recognized by no DAA. Now heed of the independent prefixes of L: $L\|\{a\} = \{a, aa\}$, $L\|\{b\} = \{b, bb\}$ and $L\|\{ab\} = L$. We can see that $ab \in L\|\{a\} \cdot L\|\{b\}$ while $ab \notin L$, which means that Condition $C2$ fails for $T = \{a, b\}$. Moreover, it is clear that Condition $C1$ fails for T as well. Then L is not distributed.

On the other hand, the next lemma shows that any distributed regular trace language is the language of some *non-deterministic* high-level DAA. Recall here that Theorem 4.4 states that any located trace language is recognized by a $\mathrm{alph}(r)$-synchronizable DAA.

Lemma 6.4. *Let $\mathcal{A}$ be a trace automaton. For all $r \in \mathcal{R}_f(\mathcal{A})$, we denote by $\mathcal{H}_r$ the $\mathrm{alph}(r)$-synchronizable* DAA *that recognizes the located trace language $L_r(\mathcal{A})$. If $L(\mathcal{A})$ is distributed, then $\oplus_{r\in\mathcal{R}_f(\mathcal{A})}\mathcal{H}_r$ recognizes $L(\mathcal{A})$.*

Proof. In this proof, we denote $\oplus_{r\in\mathcal{R}_f(\mathcal{A})}\mathcal{H}_r$ by $\mathcal{H}$. It is not hard to prove that $L(\mathcal{A})$ is included in $L(\mathcal{H})$. So we prove only the backward inclusion. Let $u \in L(\mathcal{H})$, $T = \mathrm{alph}(u)$ and $\mathrm{cc}(T) = (T_1, \dots, T_n)$. By definition of $\mathrm{cc}(T)$, $\mathrm{Loc}(T_i) \cap \mathrm{Loc}(T_j) = \emptyset$ for all $1 \leq i < j \leq n$. Since $\mathrm{alph}(u) = T$, u is equivalent (w.r.t. $\sim$) to some word $u_1 \cdots u_n$ where $\mathrm{alph}(u_i) = T_i$ for each $i \in [1, n]$. Moreover $u_1 \cdots u_n$ belongs to $L(\mathcal{H})$ as well, because the latter is a trace language. So, there is an execution s of $\mathcal{H}$ that yields $u_1 \cdots u_n$ and leads all processes of $\mathcal{P}$ from initial states to final states.

Now consider any process $p \notin \mathrm{Loc}(T)$. Then, along s, p takes part in mute transitions only. By construction of $\mathcal{H}$ and because each DAA $\mathcal{H}_r$ is $\mathrm{alph}(r)$-synchronizable, the only way for p to do only mute transitions is to start in an initial local state of some DAA component $\mathcal{H}_{r_p}$ with $p \notin \mathrm{Loc}(\mathrm{alph}(r_p))$. Moreover the definition of $\mathcal{R}_f(\mathcal{A})$ gives that $L(\mathcal{H}_{r_p}) = L_{r_p}(\mathcal{A}) \neq \emptyset$. Then there exists some $v_p \in L_{r_p}(\mathcal{A})$ such that $p \notin \mathrm{Loc}(v_p)$. Since similar facts hold for all processes that are not in $\mathrm{Loc}(T)$ and $L_{r_p}(\mathcal{A}) \subseteq L(\mathcal{A})$, Condition $C1$ of Def. 6.1 fails for T. However we have supposed that $L(\mathcal{A})$ is distributed. Then Condition $C2$ holds for T, that is $\prod_{i\in[1,n]} L(\mathcal{A})\|T_i \subseteq L(\mathcal{A})$.

Let $i \in [1, n]$. We know that $\mathrm{alph}(u_i) = T_i$, (T_i, D) is connected (by def. of $\mathrm{cc}(T)$) and each DAA component $\mathcal{H}_r$ of $\mathcal{H}$ is $\mathrm{alph}(r)$-synchronizable. Then, along s, all processes of $\mathrm{Loc}(T_i)$ (those that take part in u_i) are forced to travel all together some DAA component $\mathcal{H}_{r_i}$ such that T_i appears in $\mathrm{cc}(\mathrm{alph}(r_i))$. Furthermore, $L(\mathcal{H}_{r_i}) = L_{r_i}(\mathcal{A})$ and the definition of $\mathcal{R}_f(\mathcal{A})$ gives that $L_{r_i}(\mathcal{A}) \neq \emptyset$. So, there is some word v_i such that $u_i v_i \in L_{r_i}(\mathcal{A})$ and $\mathrm{Loc}(v_i) \cap \mathrm{Loc}(T_i) = \emptyset$. In other words, $u_i \in L_{r_i}(\mathcal{A})\|T_i$. By Prop. 4.3, $L(\mathcal{A}) = \bigcup_{r\in\mathcal{R}_f(\mathcal{A})} L_r(\mathcal{A})$. It follows that $u_i \in L(\mathcal{A})\|T_i$.

To conclude, we have just shown that $\prod_{i\in[1,n]} L(\mathcal{A})\|T_i \subseteq L(\mathcal{A})$ and $u_i \in L(\mathcal{A})\|T_i$ for all $i \in [1, n]$. In consequence, $u_1 \cdots u_n$ belongs to $L(\mathcal{A})$. Since $L(\mathcal{A})$ is a trace language, u belongs to $L(\mathcal{A})$ as well. ■

We come to our main result that follows immediately from the two previous lemmas. Again, for each roadmap r in $\mathcal{R}_f(\mathcal{A})$, $\mathcal{H}_r$ refers to the $\mathrm{alph}(r)$-synchronizable DAA that recognizes the located trace language $L_r(\mathcal{A})$.

Theorem 6.5. *Let L be a regular trace language. The next statements are equivalent:*

1. *L is recognized by some* DAA *;*
2. *L is distributed;*
3. *$L = L(\oplus_{r\in\mathcal{R}_f(\mathcal{A})}\mathcal{H}_r)$ for all trace automata $\mathcal{A}$ such that $L(\mathcal{A}) = L$.*
4. *$L = L(\oplus_{r\in\mathcal{R}_f(\mathcal{A})}\mathcal{H}_r)$ for some trace automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L$.*

As corollary we can check effectively whether a given regular trace language is distributed. For instance it suffices to check whether $L(\oplus_{r\in\mathcal{R}_f(\mathcal{A})}\mathcal{H}_r) = L(\mathcal{A})$ for the minimal trace automaton $\mathcal{A}$ such that $L(\mathcal{A}) = L$.

Discussion

In this paper, we have paid attention to the particular case of non-deterministic distributed asynchronous automata. We have shown that they correspond to the regular trace languages that are distributed. Interestingly, we can verify whether or not a language is distributed. However we do not know the complexity of this problem, yet.

Another question is to determine how to verify whether a regular trace language is recognized by a *deterministic* DAA. Indeed, the class of regular trace languages recognized by deterministic DAAs is different from the one recognized by non-deterministic DAAs. For instance, consider the regular trace language $L = [c^\star a] \cup [c^\star b]$ over $\{a, b, c\}$ where $\mathrm{Loc}(a) = \{i\}$, $\mathrm{Loc}(b) = \{j\}$ and $\mathrm{Loc}(c) = \{i, j\}$. It is not hard to see that L is distributed. Then Theorem 6.5 ensures that L is recognized by some DAA. However, no deterministic DAA recognizes L.

Acknowledgment. The author would like to thank Rémi Morin for fruitful discussions and comments. We are grateful to the referees for their constructive criticism.

References

1. Baudru N. and Morin R.: *Safe Implementability of Regular Message Sequence Charts Specifications*. Proc. of the ACIS 4th Int. Conf. SNDP (2003) 210–217
2. Baudru N. and Morin R.: *Unfolding Synthesis of Asynchronous Automata*. CSR, LNCS **3967** (2006) 46–57
3. Cori R., Métivier Y. and Zielonka W.: *Asynchronous mappings and asynchronous cellular automata*. Inform. and Comput. **106** (1993) 159–202
4. Diekert V. and Rozenberg G.: *The Book of Traces*. (World Scientific, 1995)
5. Diekert V. and Muscholl A.: *Construction of asynchronous automata*. (Ch. 5 of [4], 1995)
6. Genest B. and Muscholl A.: *Constructing Exponential-size Deterministic Zielonka Automata*. ICALP, LNCS **4052** (2006) 565–576
7. Genest B., Kuske D., and Muscholl A.: *A Kleene theorem and model checking algorithms for existentially bounded communicating automata*. I&C **204** (2006) 920–956
8. Henriksen J.G., Mukund M., Narayan Kumar K., Sohoni M. and Thiagarajan P.S.: *A Theory of Regular MSC Languages*. I&C **202** (2005) 1–38
9. Klarlund N., Mukund M. and Sohoni M.: *Determinizing Asynchronous Automata*. ICALP, LNCS **820** (1994) 130–141
10. McNaughton R., Yamada H.: *Regular Expressions and State Graphs for Automata*. J. Symbolic Logic **32** (1967) 390–391
11. Métivier Y.: *An algorithm for computing asynchronous automata in the case of acyclic non-commutation graph*. ICALP, LNCS **267** (1987) 226–236
12. Morin R.: *Concurrent Automata vs. Asynchronous Systems*. MFCS, LNCS **3618** (2005) 686–698
13. Mukund M. and Sohoni M.: *Gossiping, Asynchronous Automata and Zielonka's Theorem*. Report TCS-94-2, SPIC Science Foundation (Madras, India, 1994)
14. Muscholl A.: *On the complementation of Büchi asynchronous cellular automata*. ICALP, LNCS **820** (1994) 142–153
15. Pighizzini G.: *Synthesis of Nondeterministic Asynchronous Automata*. Algebra, Logic and Applications, vol. **5** (1993) 109–126
16. Thiagarajan P.S.: *Regular Event Structures and Finite Petri Nets: A Conjecture*. Formal and Natural Computing, LNCS **2300** (2002) 244–256
17. Zielonka W.: *Notes on finite asynchronous automata*. RAIRO, Theoretical Informatics and Applications **21** (Gauthiers-Villars, 1987) 99–135