Constructing Exponential-size Deterministic Zielonka Automata

Blaise Genest[1,2] and Anca Muscholl[1]

[1] LIAFA, Université Paris 7 et CNRS, 75251 Paris Cedex 05, France
[2] IRISA, Université Rennes I et CNRS, 35042 Rennes Cedex, France

Abstract. The well-known algorithm of Zielonka describes how to transform automatically a sequential automaton into a deterministic asynchronous trace automaton. In this paper, we improve the construction of deterministic asynchronous automata from finite state automaton. Our construction improves the well-known construction in that the size of the asynchronous automaton is simply exponential in both the size of the sequential automaton and the number of processes. In contrast, Zielonka's algorithm gives an asynchronous automaton that is doubly exponential in the number of processes (and simply exponential in the size of the automaton).

1 Introduction

A challenging problem concerning concurrent systems is to design distributed algorithms or, even simpler, distributed finite state devices. The problem is that it is easier to think in a sequential rather than a concurrent way, and easier to model the global behavior of a system. In general it is much harder to synthesize local devices, since they only have a local view of the global behavior. Local control of a single process has to deal with partial information, consisting of local behaviors plus information exchanged with other processes.

In this paper we reconsider the problem of synthesizing deterministic asynchronous (trace) automata. These are basically (deterministic) local automata that exchange information using shared (state) variables. The underlying mathematical theory is the theory of Mazurkiewicz traces [10], which has brought a large number of beautiful results in the theory of automata and logics (see [5] for a survey). The basic idea of trace theory is to model actions in a concurrent system by explicitly providing an independence relation between actions that do not share any resource.

A fundamental and difficult result for Mazurkiewicz traces is Zielonka's theorem [16], which states that (diamond) finite state automata can be effectively transformed into deterministic asynchronous automata. This result is all the more fundamental since it has been used for several other closely related problems, as synthesis of communicating automata, with bounded communication channels [13, 7, 8], or existentially-bounded channels [6] or causal memory [1].

The main drawback of Zielonka's theorem is that it yields an asynchronous automaton of doubly exponential size in the size of the alphabet [16, 4]. The paper [14] gives a more direct proof of Zielonka's theorem, with a complexity which is doubly exponential in the number of processes (instead of the size of the alphabet). Similarly, [7] synthesizes bounded communicating automata, and their construction is of size doubly exponential in the number of processes.

We propose here a simple improvement of Zielonka's algorithm in order to lower the complexity by one exponent. We obtain a deterministic asynchronous automaton of size exponential in the size of the input, that is both in the size of the finite state automaton *and* the number of processes. For applications e.g. in verification it is worth to note that the size of the automaton is exponential only because the memory needed by each process is of polynomial size. The time needed to compute any transition of the automaton is also only polynomial, which can be used in practice whenever we need only to simulate the automaton on-the-fly. Moreover, this construction is tight, since the determinization of sequential automata requires exponential size.

Related work. There were several attempts to simplify Zielonka's construction as described in [16, 12]. In some special cases the construction can be indeed simplified (see e.g . [5] Ch. 8), but the complexity is still exponential (the starting point there is a monoid homomorphism in place of an automaton). Our construction also reduces the complexity of other works [12, 15] using Zielonka's theorem or its variant for communicating processes [1, 13, 7] to produce a distributed automaton with local final states or without deadlock. Very recently, [2] proposed a construction of *non-deterministic* asynchronous automata of size $|\mathcal{A}|^{2^{|\Sigma|}}$ while we produce a *deterministic* one. Their complexity is thus polynomial in $|\mathcal{A}|$ but still doubly exponential in Σ, while our construction is of simply exponential complexity in both $|\mathcal{A}|$ and $|\Sigma|$.

Overview of the paper. We first recall some basics of Mazurkiewicz traces in Section 2. Then we recall the main ingredients of Zielonka's construction in Section 3. In Section 4 we present the new idea of decomposing into zones, and in Section 5 we present the new construction of deterministic asynchronous automata.

2 Preliminaries

We assume that there is a set $\mathcal{P}$ of processes and an alphabet Σ which are fixed. Each letter $a \in \Sigma$ is an action associated with the set of processes $\mathrm{dom}(a) \subseteq \mathcal{P}$ involved in a. A pair (Σ, dom) is called *distributed alphabet*. A (non-deterministic) automaton over the alphabet Σ is a tuple $\mathcal{A} = (V, \Sigma, \rightarrow, v^0, F)$ with a finite set of states V, a set of final states F, an initial state v^0 and a non-deterministic transition function $\rightarrow : V \times \Sigma \rightarrow 2^V$. The size of an automaton is the number of states.

Concurrent systems with shared actions given by a distributed alphabet (Σ, dom), are readily modeled by Mazurkiewicz traces. The idea is that the distribution of the alphabet defines an independence relation among actions $I \subseteq \Sigma \times \Sigma$, by setting $(a, b) \in I$ if and only if $\mathrm{dom}(a) \cap \mathrm{dom}(b) = \emptyset$. We call (Σ, I) an *independence alphabet*. The complementary relation $D = \Sigma \times \Sigma \setminus I$ is called a dependence relation. The independence relation induces a congruence $\sim$ on Σ^* by setting $u \sim v$ if there exist words $u_1, \ldots, u_n \in \Sigma^*$ with $u_1 = u$, $u_n = v$ and such that for every $i < n$ we have $u_i = xaby$, $u_{i+1} = xbay$ for some $x, y \in \Sigma^*$ and $(a, b) \in I$. An $\sim$-equivalence class is simply called a *(Mazurkiewicz) trace* [10]. We denote by $[u]$ the trace associated with the word $u \in \Sigma^*$ (for simplicity we do not refer to I, neither in $\sim$ nor in $[u]$, simply because the independence alphabet is fixed). Trace prefixes and trace factors are defined as usual, with $[p]$ a trace prefix (trace factor, resp.) of $[u]$ if p is a word prefix (word factor, resp.) of some $v \sim u$.

A (non-deterministic) automaton $\mathcal{A}$ is called I-diamond if for all $(a, b) \in I$, and all states r, s, t of $\mathcal{A}$ with $r \xrightarrow{a} s$ and $s \xrightarrow{b} t$, there also exists a state s' with $t \xrightarrow{b} s'$ and $s' \xrightarrow{a} t$. Note that the I-diamond property implies that the language $\mathcal{L}(\mathcal{A})$ of $\mathcal{A}$ is *I-closed*: that is, $u \in \mathcal{L}(\mathcal{A})$ if and only if $v \in \mathcal{L}(\mathcal{A})$ for every $u \sim v$.

We use asynchronous automata as distributed models with finite control. Our definition is slightly different from the usual definitions for asynchronous and asynchronous cellular automata [5], see the remark below.

Definition 1 *A deterministic asynchronous automaton over the distributed alphabet (Σ, dom) is a tuple $\mathcal{B} = ((K_p, \delta_p, k_p^0)_{p \in \mathcal{P}}, Acc)$ such that for any $p \in \mathcal{P}$:*

- *K_p is the finite set of local states of process p.*
- *$\delta_p : (\Sigma \times \prod_{q \in \mathcal{P}} K_q) \to K_p$ is the local transition function of process p, satisfying the following conditions for all actions $a \in \Sigma$ and local states $s_q \in K_p$, $q \in \mathcal{P}$:*
  - *for $p \notin dom(a)$, we have $\delta_p(a, s_1, \ldots, s_n) = s_p$.*
  - *for $p \in dom(a)$, the state $\delta_p(a, s_1, \ldots, s_n)$ depends only on $(s_q)_{q \in dom(a)}$, that is $\delta_p(a, s_1, \ldots, s_n) = \delta_p(a, s_1, \ldots, s'_q, \ldots, s_n)$ for $q \notin dom(a)$.*
- *$k_p^0 \in K_p$ is the local initial state of process p.*
- *$Acc \subseteq \prod_{p \in \mathcal{P}} K_p$ is a set of (global) accepting states.*

An asynchronous automaton accepts a regular language with the following global semantics:

Definition 2 *The language of an asynchronous automaton $\mathcal{B} = ((K_p, \delta_p, k_p^0)_{p \in \mathcal{P}}, Acc)$ is defined as $\mathcal{L}(\mathcal{B}) = \mathcal{L}(\mathcal{A})$, where $\mathcal{A} = (K, \delta, k^0, Acc)$ is the following automaton, called the global automaton of $\mathcal{B}$:*

- *The global state space is $K = \prod_{p \in \mathcal{P}} K_p$.*
- *The initial state is $k^0 = (k_p^0)_{p \in \mathcal{P}}$.*
- *The global transition function $\delta : \Sigma \times K \to K$ is defined for all $a \in \Sigma$, $k \in K$ by $\delta(a, k) = (k'_p)_{p \in \mathcal{P}}$ with $k'_p = \delta_p(a, k)$ for all p.*

Remark 1. Our definition of asynchronous automaton differs from the usual one in that we define transitions on processes instead of letters. Moreover, the transitions corresponding to processes are like transitions in a cellular automaton, that is, only the local state associated with the process executing a transition changes.

The definition thus corresponds to a shared-read, owner-write mode (the transition function reads the states of all other processes involved in the current action and writes its local state). However, the difference wrt asynchronous automata is merely syntactical, since in the global behavior of the automaton we synchronize all processes from $\mathrm{dom}(a)$ when executing an action a.

For several purposes it is convenient to represent traces by (labeled) pomsets. Formally, a trace $T = [a_1 \cdots a_n]$ ($a_i \in \Sigma$ for all i) corresponds to a labeled pomset $(E, \lambda, \leq)$ defined as follows: $E = \{e_1, \ldots, e_n\}$ is a set of events (or nodes), one for each position in T. Event e_i is labeled by $\lambda(e_i) = a_i$, for each i. The relation $\leq$ is the least partial order on E with $e_i \leq e_j$ whenever $(a_i, a_j) \in D$ and $i \leq j$. In terms of graphs it is convenient to identify a trace T with its *dependence graph*, by defining an edge from e_i to e_j iff $(a_i, a_j) \in D$ and $i \leq j$. A total order $e_1 \cdots e_n$ that is compatible with $\leq$ is called a *linearization* of T. Since all these formalisms are equivalent, we will refer to graph nodes as events for convenience. Moreover, we will use for convenience set operations on traces, interpreting them on the associated graphs. For instance, assume that T_1, T_2 are both prefixes of some trace T. In other words, each T_i is a downward closed subgraph of T. Then we write $T_1 \cap T_2$ ($T_1 \cup T_2$, resp.) for the least (greatest, resp.) common prefix of T_1, T_2. Also, we write $e_i \in T$ for denoting that e_i is a vertex of (the graph of) T.

For any trace factor T' of T, we denote by $\mathrm{alph}(T') = \bigcup_{e \in T'} \lambda(e)$ the letters occurring in T', resp. by $\mathrm{dom}(T') = \bigcup_{e \in T'} \mathrm{dom}(\lambda(e))$ the processes occurring in T'. For $a \in \Sigma$ we call any event e with $\lambda(e) = a$ an a-event.

We have in Figure 1 a trace T with $(a, b), (a, c) \in D$ and $(a, d) \in I$. Hence, *cbadcbadb* $\sim$ *cbdacbdab* are two representing words of T. Process p can read the state of q when executing action a, but not when executing action c.

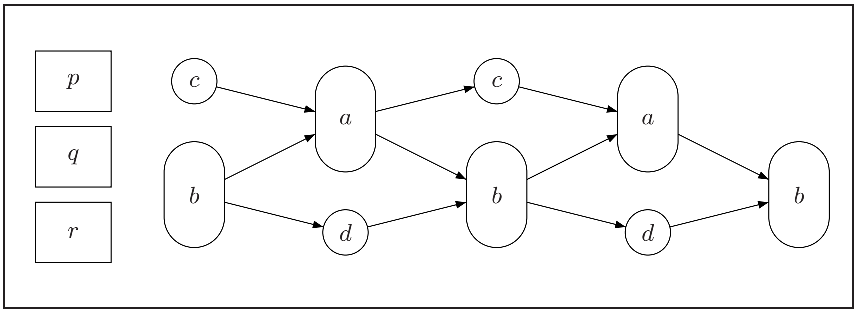

Fig. 1. The pomset associated with the trace $T = [c\,b\,a\,d\,c\,b\,a\,d\,b]$, with $\mathrm{dom}(a) = \{p, q\}$, $\mathrm{dom}(b) = \{q, r\}$, $\mathrm{dom}(c) = \{p\}$, $\mathrm{dom}(d) = \{r\}$.

3 Zielonka's Theorem

We recall the main ingredients in Zielonka's construction of deterministic asynchronous automata. Our presentation is based on [5, 14, 4]. Let T be a trace and $p \in \mathcal{P}$ a process, then we denote by $\mathrm{pref}_p(T)$ the minimal trace prefix of T which contains all events of T on process p. Hence, $\mathrm{pref}_p(T)$ has a unique maximal event which is the last event of T on process p. That is, $\mathrm{pref}_p(T)$ corresponds to the history of process p after executing T, that we also refer to as *p-view*. For instance, in Figure 1 we have $\mathrm{pref}_p(T) = [cbadcba]$. For a set of processes $P \subseteq \mathcal{P}$, let $\mathrm{pref}_P(T) = \cup_{p\in P}\mathrm{pref}_p(T)$ be the P-view of T.

Zielonka's construction starts with a regular, I-closed language, that is presented either through a monoid homomorphism or an automaton satisfying the I-diamond property, [4]. In most applications we are interested in the second case, where we start with a (non-deterministic) I-diamond automaton.

Theorem 1 *[16] Let $\mathcal{A}$ be an I-diamond automaton over the independence alphabet (Σ, I). An equivalent deterministic asynchronous automaton $\mathcal{B}$ with $2^{O(|\mathcal{A}|^2(2^{|\Sigma|}))}$ states can be obtained applying the construction from [4]. An equivalent deterministic asynchronous automaton with $2^{O(|\mathcal{A}|^2(2^{|\mathcal{P}|}))}$ states can be obtained applying [14].*

3.1 General idea and timestamping

We first describe informally how Zielonka's construction works. Let $\mathcal{A} = (V, \Sigma, \to, v^0, F)$ be an I-diamond automaton. When an action $a \in \Sigma$ is executed after T by the processes in $\mathrm{dom}(a)$, each process of $\mathrm{dom}(a)$ reads the states of the other processes in $\mathrm{dom}(a)$ and changes its own state accordingly. At this step, each process $p \in \mathrm{dom}(a)$ computes the events of $\mathrm{dom}(a)$ that were not in its p-view, that is $\mathrm{pref}_p(Ta) \setminus \mathrm{pref}_p(T)$, or equivalently, $\bigcup_{q\in\mathrm{dom}(a)} \mathrm{pref}_q(T) \setminus \mathrm{pref}_p(T)$ plus the last a. First, events are labeled by timestamps in order to recover which events are in $\mathrm{pref}_q(T) \cap \mathrm{pref}_p(T)$ and which events are in $\mathrm{pref}_q(T) \setminus \mathrm{pref}_p(T)$, by simply comparing the timestamps. For instance, in Figure 1, when the last b is executed, then process q reads the state of process r and vice-versa. They find that the second b is the only maximal event in their common past, q discovers that action d was executed in $\mathrm{pref}_r(T) \setminus \mathrm{pref}_q(T)$, and r that ca was executed in $\mathrm{pref}_q(T) \setminus \mathrm{pref}_r(T)$.

The problem is that the number of events that have to be stored is arbitrarily large. Zielonka's construction explains that only a bounded set S_1 of events needs to be timestamped. Finally a finite representation of the behavior of the given sequential automaton $\mathcal{A}$ on $\mathrm{pref}_q(T) \setminus \mathrm{pref}_P(T)$ is needed. To this purpose, transition relations Δ_X are used for $X \in \Sigma^\star$, where $\Delta_X(R) = \{s \in V \mid \exists r \in R, r \xrightarrow{X} s\}$ for any subset R of states of $\mathcal{A}$. That is, $\Delta_X(R)$ is the set of states of $\mathcal{A}$ reached after reading the word X from any state in R. Zielonka's construction explains how process q remembers relations Δ_X only for an exponential number of sequences X. A global state of the asynchronous automaton is then accepting if and only if the Δ relation associated with it satisfies $\Delta_T(v^0) \cap F \neq \emptyset$.

In order to explain the construction in more detail, we define now the sets of events S_1, S_2 used by Zielonka's timestamping and the set of factors X for which we remember the function Δ_X. We will not provide the definition of the timestamping, which is the usual one (the reader is referred to [4, 5, 14] instead). First, the set S_1 simply consists of the last event on process p, for every $p \in \mathcal{P}$:

Definition 3 *Let $T = (E, \lambda, \leq)$ be a trace. The primary information of T is $S_1(T) = \{e \in E \mid \exists p \in dom(e), \forall f \in E, p \in dom(f) \implies f \leq e\}$.*

The following crucial property of S_1 can be found in [14] (Lemma 1, page 9) and can be quickly obtained from [5] (Proposition 8.3.5, page 259).

Lemma 1. *Let T be a trace and $P, Q \subseteq \mathcal{P}$ two subsets of processes. Then $\max(\mathit{pref}_P(T) \cap \mathit{pref}_Q(T)) \subseteq S_1(\mathit{pref}_P(T)) \cap S_1(\mathit{pref}_Q(T))$.*

That is, the maximal events of the intersection of the views of the sets $P, Q \subseteq \mathcal{P}$ belong to the primary information of each of these two views.. However, knowing the events in $S_1(\mathrm{pref}_P(T))$ and $S_1(\mathrm{pref}_Q(T))$ is not enough for computing $\max(\mathrm{pref}_P(T) \cap \mathrm{pref}_Q(T))$.

Notice that during the execution of an asynchronous automaton, every event is created as an event of S_1 and can eventually be removed from S_1. Note also that $\{e\} \subseteq S_1(Te) \subseteq S_1(T) \cup \{e\}$ for every event $e \in E$.

After the trace $T' = [cbadcbad]$ is executed, $S_1(\mathrm{pref}_r(T')) = S_1([cbadbd])$ contains the first a (for process p), the second b (for q) and the second d (for r), as shown in Figure 2. The common past of the views of q, r has the second b as unique maximal event. If an event b is then executed (as in Figure 1), we have that $S_1(T'b)$ contains the second a (for p) and the third b (for q and r).

The timestamping function TS in Zielonka's construction [16, 5, 14] labels every event in $S_1(\mathrm{pref}_p(T))$. It allows to compute $S_1(\mathrm{pref}_p(T)) \cap S_1(\mathrm{pref}_q(T))$ for every $p, q \in \mathcal{P}$. For this, the timestamping labels events wrt. the so-called secondary information, that contains the last event on process q before the last event on p for every $p, q \in \mathcal{P}$ (these two events can be equal).

Definition 4 *Let $T = (E, \lambda, \leq)$. The secondary information of T is the set $S_2(T) = \{e \in E \mid \exists g \in S_1(T), q \in dom(e), \forall f \leq g, q \in dom(f) \implies f \leq e\}$.*

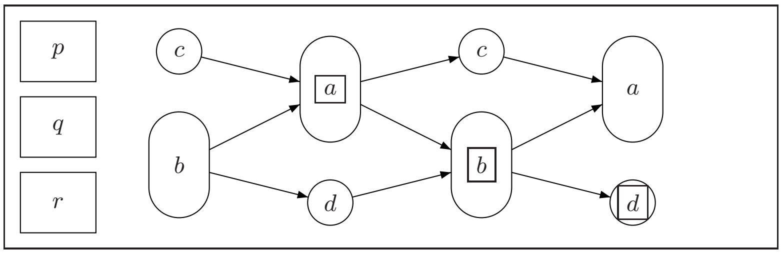

Fig. 2. Trace T', where the distinguished events are those in $S_1(\mathrm{pref}_r(T'))$.

In particular, $S_1(T) \subseteq S_2(T)$. The primary information S_1 is of size $|\mathcal{P}|$, the secondary information has at most $|\mathcal{P}|^2$ events, and the timestamping is of size $2|\mathcal{P}|^3 \log(|\mathcal{P}|)$ [14]. In Figure 2, the secondary information $S_2(\mathrm{pref}_r(T'))$ contains the first two b, the first a and the second d. The first c is not in $S_2(\mathrm{pref}_r(T'))$.

Let $T = (E, \lambda, \leq)$ be a trace. For every subset $W \subseteq S_1(\mathrm{pref}_p(T))$, we define the suffix $X_W(T) = \{e \in E \mid \forall s \in W : e \not\leq s\}$ of T. In particular, $X_\emptyset(T) = T$. For instance, in Figure 2 with $\mathrm{pref}_r(T) = [cbadbd]$, if we fix W to be the first a, then $X_W(\mathrm{pref}_r(T))$ consists of the first and second d and of the second b. For every set $P \subseteq \mathcal{P}$ and every subset $W \subseteq S_1(\mathrm{pref}_P(T))$ of primary events of the P-view, we remember the transition relation Δ_W which associates with every state r of $\mathcal{A}$ the set of states $\Delta_W(r) = \{s \in V \mid r \xrightarrow{X_W} s\}$ that can be reached from r by the trace $X_W(T)$. Process p thus remembers an exponential number (wrt. the number of processes $|\mathcal{P}|$) of relations in $V \times V$. Note that we can compute on-the-fly $\Delta_W(R) = \bigcup_{r \in R} \Delta_W(r)$ for any subset R of states of $\mathcal{A}$.

3.2 The asynchronous automaton

We describe now how a deterministic asynchronous automaton updates the state information wrt. the I-diamond automaton, as done e.g. in [16, 5, 14]. A transition of process p in the asynchronous automaton $\mathcal{B}$ consists of reading the states of other processes, and adding a new event e. For instance, if $\lambda(e) = a$ and $\mathrm{dom}(a) = \{p, q, r\}$, then we can decompose the a-transition of process p as first reading the local state of process q and updating its local state, then doing it again with process r, and finally adding the event e.

The state reached on a trace $\mathrm{pref}_P(T)$, $P \subseteq \mathcal{P}$, is a tuple $(S_1, \mathrm{TS}, (\Delta_W)_{W \subseteq S_1})$, containing the primary information S_1 of the P-view $\mathrm{pref}_P(T)$, the timestamping TS, and the transition relations $(\Delta_W)_{W \subseteq S_1}$.

We do not describe here the update of S_1 and TS, since we use the same timestamping algorithm as in [16, 5, 4, 14].

Assume that the current state reached on $\mathrm{pref}_{\mathrm{dom}(a)}(T)$ is $(S_1, \mathrm{TS}, (\Delta_W)_{W \subseteq S_1})$ and we add the event e with $\lambda(e) = a$. The new state will be $(S'_1, \mathrm{TS}', (\Delta'_W)_{W \subseteq S'_1})$. For every $W \subseteq S'_1$ we have two cases, depending on whether or not $e \in W$:

- Either $W \subseteq S_1$, and then $\Delta'_W = \xrightarrow{a} \circ \Delta_W$.
- Else we have $e \in W$, hence $X_W = \emptyset$ and $\Delta_W = \mathrm{Id}$.

We now look at the local state modifications. Assume that $q \in \mathrm{dom}(a)$ and that the local state of process q, say $(S^q_1, \mathrm{TS}^q, (\Delta^q_W)_{W \in S^q_1})$, is added to the state $(S_1, \mathrm{TS}, (\Delta_W)_{W \subseteq S_1})$ reached on $\mathrm{pref}_P(T)$, $P \subseteq \mathrm{dom}(a)$, $q \notin P$. The state reached on $\mathrm{pref}_{P \cup \{q\}}(T)$, say $(S'_1, \mathrm{TS}', (\Delta'_W)_{W \subseteq S'_1})$ is computed as follows. For every $W' \subseteq S'_1$ we set:

- Let $W = W' \cap S_1$ and $W^q = (W' \cup S_1) \cap S^q_1$.
- Set $\Delta'_{W'} = \Delta^q_{W^q} \circ \Delta_W$. Thus, the $X_{W'}$-suffix of $\mathrm{pref}_{P \cup \{q\}}(T)$ is the union of two suffixes (over disjoint sets of processes), namely the X_W-suffix consisting of events of the P-view that are not below some event in W'; and the X_{W^q}-suffix consisting of events from $\mathrm{pref}_q(T) \setminus \mathrm{pref}_P(T)$ that are not below some event in W'.

A global state$(S_1, \mathrm{TS}, (\Delta_W)_{W\subseteq S_1})$ is accepting if and only if $\Delta_\emptyset(v^0) \cap F \neq \emptyset$.

Let us comment on the complexity of the construction. At each event $e \in E$, updating the primary and secondary information takes polynomial time. Updating the relations Δ may take an exponential time in the number of processes $|\mathcal{P}|$. Hence, Zielonka's construction gives a transition function which needs exponential time (and space) to compute the next state. Overall there are doubly exponentially many different memory configurations, which is why the automaton is doubly exponential.

The exponential memory comes only from the straightforward use of Δ. In the rest of the paper we explain how we can achieve a better complexity by using new transition relations Δ, still keeping the primary information and the timestamping.

4 Zone Decomposition

The idea of this paper is to define transition relations Δ on *zones* of the p-views $\mathrm{pref}_p(T)$. When the transition relation on some factor of T is needed, we compose the transition relations of the zones included in the factor. Zones are defined as equivalence classes of the following relation:

Definition 5 *Let $T = (E, \leq, \lambda)$ be a trace. For an event $e \in E$ we define the set of events $S(e) = \{f \in S_1(T) \mid e \leq f\}$. We say that two events e, f are equivalent (denoted as $e \equiv f$) if and only if $S(e) = S(f)$. The equivalence classes of $\equiv$ are called zones.*

Let Z be a zone and define $S(Z) = S(e)$ for some event $e \in Z$. Let also Z, Z' be two zones of some trace T. We write $Z < Z'$ if $Z \neq Z'$ and if $e < e'$ for some events $e \in Z$, $e' \in Z'$. By $\mathrm{dom}(Z)$ we denote the set of processes occurring in the zone Z, i.e., $\mathrm{dom}(Z) = \cup_{e\in Z}\mathrm{dom}(e)$. The following lemma is easy to show:

Lemma 2. *1. A zone of T is a factor of T and contains at most one event from the secondary information $S_1(T)$.*
2. The set of zones partitions the set of events of T.
3. The relation $<$ on zones is acyclic. It induces the least partial order such that $S(Z) \supsetneq S(Z')$ and $dom(Z) \cap dom(Z') \neq \emptyset$ implies $Z < Z'$.

Proof of 3). Assume that $Z_1 \leq Z_2 \leq \cdots \leq Z_k = Z_1$, say with $e_i \in Z_i$ and $f_j \in Z_{j+1}$ $(1 \leq i, j < k)$ such that $e_i < f_i$ for all i. Hence, $S(Z_1) \supseteq S(Z_2) \supseteq \cdots \supseteq S(Z_k) = S(Z_1)$, thus $S(Z_i) = S(Z_j)$ and $Z_i = Z_j$ for all i, j.

Figure 3 depicts the same trace $T = [cbadcbad]$ as Figure 2. Recall that $S_1(\mathrm{pref}_r(T))$ consists of the first a, the second b and the second d. There are three zones in $\mathrm{pref}_r(T)$: Z_1 is the first a, b and c, Z_2 is the first d and the second b, and Z_3 is the second d (see also Figure 3). We have $Z_1 < Z_2 < Z_3$. Also, $S(Z_2)$ consists of the second b and the second d.

Zones enjoy some crucial properties, that are stated in the following.

Proposition 1 *Let T be a trace, $P, Q \subseteq \mathcal{P}$ sets of processes, and Z a zone of $pref_P(T)$. Then either $Z \subseteq pref_P(T) \cap pref_Q(T)$, or $Z \cap (pref_P(T) \cap pref_Q(T)) = \emptyset$.*

Proof. Assume by contradiction that Z is a zone that violates the statement of the proposition. Consider the factor $Z_1 = Z \cap \mathrm{pref}_P(T) \cap \mathrm{pref}_Q(T)$. By assumption we can write the trace factor Z as $Z \sim Z_1 Z_2$ with Z_1, Z_2 both non-empty. Let $e \in Z_1$ and $f \in Z_2$. Since $e \in \mathrm{pref}_P(T) \cap \mathrm{pref}_Q(T)$, there exists a maximal event g of $\mathrm{pref}_P(T) \cap \mathrm{pref}_Q(T)$ such that $e \leq g$. By Lemma 1, $g \in S_1(\mathrm{pref}_P(T))$. Since $e, f \in Z$, we have $e \equiv f$ and thus $f \leq g$. This implies $f \in \mathrm{pref}_P(T) \cap \mathrm{pref}_Q(T)$, a contradiction. $\square$

Proposition 2 *Let T be a trace. There are at most $|\mathcal{P}|^2 + |\mathcal{P}|$ zones in T.*

Proof. Assume by contradiction that there are more than $|\mathcal{P}|(|S_1(T)| + 1)$ zones. Hence we can find a process p involved in at least $k = |S_1(T)| + 2$ zones. Since these zones have intersecting domains, they are strictly ordered, let us say $Z_1 < \cdots < Z_k$. Hence $S(Z_1) \supsetneq \cdots \supsetneq S(Z_k)$. This implies $|S(Z_1)| \geq k - 1 = |S_1(T)| + 1$. This is a contradiction since there are at most $|S_1(T)|$ events in $S(Z_i)$ for all i. $\square$

The last property is crucial when the zone partition is updated.

Proposition 3 *Let T be a trace, $P \subseteq \mathcal{P}$ a set of processes, $q \notin P$ a process and e, f two events of $pref_P(T)$. Let us write S for the function of Def. 5 on $pref_P(T)$ and S' for the function on $pref_{P \cup q}(T)$. If $S(e) = S(f)$, then $S'(e) = S'(f)$.*

Proof. Assume that $S(e) = S(f)$. Let $g \in S'(e)$. The first case is where $g \in S_1(pref_P(T))$, hence $g \in S(e) = S(f)$ and $f \leq g$ by definition of $S(f)$.

Else, $g \notin S_1(pref_P(T))$, hence $g \in S_1(\mathrm{pref}_q(T))$ and $g \notin \mathrm{pref}_P(T)$. Let h be a maximal event in $\mathrm{pref}_P(T)$ with $e \leq h \leq g$. There exists h' in $\mathrm{pref}_q(T)$ with $h \lessdot h' \leq g$. That is $\mathrm{dom}(h) \cap \mathrm{dom}(h') \neq \emptyset$, and let r be a process in this intersection. By maximality of h, we have $h' \notin \mathrm{pref}_P(T)$. That is, h is the last event on process r of $\mathrm{pref}_P(T)$. By definition of S, we have $h \in S(e) = S(f)$, hence $f \leq h \leq g$. We conclude by symmetry between e and f. $\square$

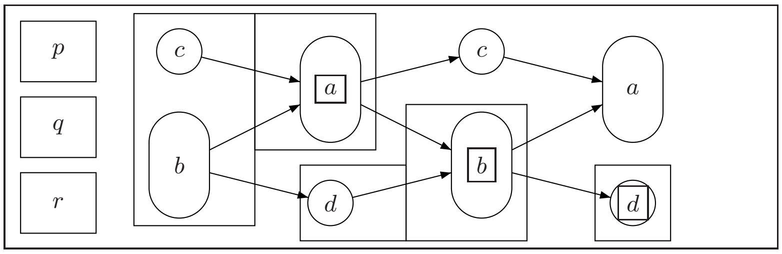

Fig. 3. The three zones of $\mathrm{pref}_r(T)$. The distinguished nodes are those in $S_1(\mathrm{pref}_r(T))$.

For each zone Z, the function Δ_Z requires space $\leq |\mathcal{A}|^2$, and there are at most $|\mathcal{P}|^2 + |\mathcal{P}|$ zones. The transition function constructed in the next section gives:

Theorem 2 *Let $\mathcal{A}$ be a (non-deterministic) I-diamond automaton over the independence alphabet (Σ, I). We can construct an equivalent deterministic asynchronous automaton $\mathcal{B}$ with less than $2^{|\mathcal{A}|^2 \times (|\mathcal{P}|^2 + |\mathcal{P}|) + 2|\mathcal{P}|^4}$ states. Each process has a memory of size $O(|\mathcal{A}|^2 \times |\mathcal{P}|^2 + |\mathcal{P}|^4)$, and computes its next state in time $O(|\mathcal{A}|^2 \times |\mathcal{P}|^2 + |\mathcal{P}|^4)$.*

5 The new construction

0ur construction follows Zielonka's construction, up to the Δ relations which are remembered only for the zones of $\text{pref}_p(T)$, $p \in \mathcal{P}$. Of course, we have to adapt the transition function of the asynchronous automaton accordingly.

Let $\mathcal{A} = (V, \Sigma, \to, v^0, F)$ be a non-deterministic I-diamond automaton. We explain in the following how to build a deterministic asynchronous automaton $\mathcal{B}$ with the same language. The local p-state of $\mathcal{B}$ reached on a prefix $\text{pref}_p(T)$ is the tuple $(S_1, \text{TS}, \langle \text{dom}(Z_i), S(Z_i), \Delta(Z_i) \rangle_{i=0,\dots m})$, where:

- $\{Z_1 \dots, Z_m\}$ is the set of zones of $\text{pref}_p(T)$.
- S_1 is the primary information of $\text{pref}_p(T)$.
- The timestamping TS associates every event of S_1 with its timestamp.
- For a zone Z, $\text{dom}(Z)$ denotes the set of processes occurring in Z.
- For a zone Z, $S(Z) \subseteq \mathcal{P}$ corresponds to Definition 5. That is, $q \in S(Z)$ means that the last event on q in $\text{pref}_p(T)$ is above an event of Z.
- The transition relation Δ_Z gives for each state v of $\mathcal{A}$ the set of states $\Delta_Z(v)$ that can be reached in $\mathcal{A}$ from v by reading some linearization of Z (remember that since $\mathcal{A}$ is I-diamond, this corresponds to the set of states reached by any linearization of Z).

We define now the local transition function δ_p of the asynchronous automaton $\mathcal{B}$. We do not recall how to update the primary information S_1 and the timestamping TS, though the update of S-values includes the update of the secondary information S_2. Recall that the order on zones $Z_i < Z_j$ can be computed from the knowledge of $S(Z_i)$ and of $\text{dom}(Z_i)$, for all zones Z_i (see Lemma 2).

Assume that the action a with $p \in \text{dom}(a)$ is added to the current p-state $(S_1, \text{TS}, \langle \text{dom}(Z_i), S(Z_i), \Delta(Z_i) \rangle_{i=0,\dots,m})$.

The new p-state is $(S_1', \text{TS}', \langle \text{dom}(Z_i'), S(Z_i'), \Delta(Z_i') \rangle_{i=0,\dots,m+1})$, with:

1. Let $\text{dom}(Z_{m+1}') = \text{dom}(a)$, $\Delta_{Z_{m+1}'} = \xrightarrow{a}$, and $S(Z_{m+1}') = \text{dom}(a)$ (actually $S(Z_{m+1}')$ consists of the unique maximal a event).
2. Let $\langle \text{dom}(Z_j'), S(Z_j'), \Delta_{Z_j'} \rangle_{j=1,\dots,m} = \langle \text{dom}(Z_j), S(Z_j), \Delta_{Z_j} \rangle_{j=1,\dots,m}$.
3. For all Z_i', let $S(Z_i') \leftarrow S(Z_i') \cup \{\text{dom}(a)\}$.
4. If $S(Z_i') = S(Z_j')$ with $Z_j \not< Z_i$ then we merge Z_i', Z_j' and let $\Delta_{Z_i'} = \Delta_{Z_j'} \circ \Delta_{Z_i'}$ and $\text{dom}(Z_i') = \text{dom}(Z_j') \cup \text{dom}(Z_i')$ and we delete Z_j'.

That is, a new zone Z'_{m+1} is created representing the new event a (line 1) and the other zones are copied (line 2). We then update the S-values in line 3 and merge zones with equal S-value in line 4.

Assume that the process q is in local state $(S^q_1, \mathrm{TS}^q, \langle \mathrm{dom}(Z^q_i), S(Z^q_i), \Delta_{Z^q_i}\rangle_{i=1,\dots,n})$ with history $\mathrm{pref}_q(T)$. Moreover, process p in current state $(S_1, \mathrm{TS}, \langle \mathrm{dom}(Z_i), S(Z_i), \Delta_{Z_i}\rangle_{i=1,\dots,m}\rangle$ and history $\mathrm{pref}_P(T)$ reads the state of q (as usual, $p, q \in \mathrm{dom}(a)$ where a is the new action). The updated state of p is $(S'_1, \mathrm{TS}', \langle \mathrm{dom}(Z'_i), S(Z'_i), \Delta_{Z'_i}\rangle_{i=1,\dots,k})$, with $\mathrm{pref}_{P\cup q(}T)$ as history, where:

1. Let $J = \{i \mid 1 \le i \le n, \mathrm{TS}^q(S(Z^q_i)) \cap \mathrm{TS}(S_1) = \emptyset\}$ and $k = m + |J|$,
2. Let $\langle \mathrm{dom}(Z'_j), S(Z'_j), \Delta_{Z'_j}\rangle_{j=1,\dots,m} = (\mathrm{dom}(Z_j), S(Z_j), \Delta_{Z_j}\rangle_{j=1,\dots,m}$,
3. $\langle \mathrm{dom}(Z'_{m+j}), S(Z'_{m+j}), \Delta_{Z'_{m+j}}\rangle_{j=1,\dots,|J|} = \langle \mathrm{dom}(Z^q_i), S(Z^q_i), \Delta_{Z^q_i}\rangle_{i\in J}$,
4. The partial order $<'$ on the new zones is given by the transitive closure of the relation $< \cup <^q \cup \{(Z'_i, Z'_j) \mid i \le m < j, \mathrm{dom}(Z'_i) \cap \mathrm{dom}(Z'_j) \neq \emptyset\}$,
5. For all $Z'_i <' Z'_j$, $S(Z'_i) \leftarrow S'(Z'_i) \cup S'(Z'_j)$,
6. If $S(Z'_i) = S(Z'_j)$ and $Z_j \not< Z_i$, then we merge Z'_i and Z'_j and set $\mathrm{dom}(Z'_i) = \mathrm{dom}(Z'_j) \cup \mathrm{dom}(Z'_i)$ and $\Delta_{Z'_i} = \Delta_{Z'_j} \circ \Delta_{Z'_i}$, and we delete Z'_j.

The update operations consists in copying the zones that form a partition of $\mathrm{pref}_P(T)$ (line 2) and adding in line 3 the zones $(Z^q_i)_{i\in J}$ that partition $\mathrm{pref}_q(T) \setminus \mathrm{pref}_P(T)$. We then update the S-values in line 5. The last line merges zones with equal S-value. We say that a global state with local p-component $S^p_1, \mathrm{TS}^p, \langle \mathrm{dom}(Z^p_i), S(Z^p_i), \Delta_{Z^p_i}\rangle_{i=1,\dots,n_p})$ is accepting if we have $\Delta_{Z_n} \circ \dots \circ \Delta_{Z_1}(v^0) \cap F \neq \emptyset$ for $Z_1 \cdots Z_n$ a linearization of $(Z_i, <)_{i=1,\dots,n}$.

Example: Consider the same trace $T = [cbadcbad]$ as in Figure 3. Then $\mathrm{pref}_q(T) = [cbadcba]$ has two zones (see also Figure 4): zone Z^q_1 consisting of the first c, a, d, and the first two b, and zone Z^q_2 consisting of the second c and a. Assume that the letter b is now executed, which means that process r can read the state of process q. For instance, we have $S(Z^q_2)) = \{p, q\}$ and with Figure 3, $S(Z_2) = \{q, r\}$, that is Z_2 is not before the last event on p. Also, $S(Z_1) \setminus S(Z_2) = \{p\}$.

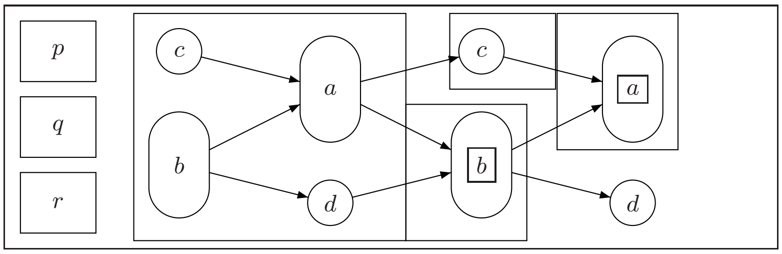

Fig. 4. Zones of $\mathrm{pref}_q(T)$. The distinguished nodes are the primary events of $\mathrm{pref}_q(T)$.

First, with the timestamping, process r computes the maximal event of $\text{pref}_q(T) \cap \text{pref}_r(T)$, which is the second b. Thus $J = \{2\}$, and the new set of zones is $\{Z'_1, Z'_2, Z'_3, Z'_4\}$ with $Z'^4 = Z^q_2$. Process r then computes $Z'_1 < Z'_2 < Z'_4$ and that Z'_3 and Z'_4 are incomparable. Thus $Z'_2 < Z'_4$, then p is added to $S'(Z'_2)$. It means that $S'(Z'_1) = S'(Z'_2)$, and hence it merges Z'_2 with Z'_1. The new set of zones is then $\{Z'_1, Z'_3, Z'_4\}$. Then the letter b is added as a new zone Z''_5, and we get $S''(Z''_1) = \{p, q, r\}, S''(Z''_3) = \{q, r\}, S''(Z''_4) = \{p, q, r\}, S''(Z''_5) = \{q, r\}$, thus we merge zones and we keep two zones $Z''_1 = Z''_1 \cup Z''_4$ and $Z''_3 = Z''_3 \cup Z''_5$.

Acknowledgement We would like to thank Dietrich Kuske for simplifying the definition of zones.

References

1. B. Adsul, M. Mukund, K. Narayan Kumar and V. Narayanan. Causal closure for MSC languages. Proc. of *FSTTCS'05*, LNCS 3821, pp. 335-347, 2005.
2. N. Baudru and R. Morin. Unfolding Synthesis of Asynchronous Automata. International Computer Science Symposium in Russia, CSR 2006. Available at http://www.cmi.univ-mrs.fr/~morin/papers/CSR.pdf.
3. D. Brand and P. Zafiropulo. On communicating finite-state machines. In *J. of the ACM*, 30(2):323-342, 1983.
4. R. Cori, Y. Métivier and W. Zielonka. Asynchronous Mappings and Asynchronous Cellular Automata. In *Inf. and Comput.*, 106(2):159-202, 1993.
5. V. Diekert and G. Rozenberg, editors. *The Book of Traces*. Chapter 8 by V. Diekert and A. Muscholl. World Scientific, Singapore, 1995.
6. B. Genest, D. Kuske and A. Muscholl. A Kleene Theorem and Model Checking for a Class of Communicating Automata. Proc. of *DLT'04*, LNCS 3340, pp. 30-48, 2004. Journal version to appear in *Inf. and Comput.*, 2006.
7. J. G. Henriksen, M. Mukund, K. Narayan Kumar, M. Sohoni and P. S. Thiagarajan. A Theory of Regular MSC Languages. In *Inf. and Comput.* 202(1):1-38, 2005.
8. D. Kuske. Regular sets of infinite message sequence charts. In *Inf. and Comput.*, 187(1):80-109, 2003.
9. M. Lohrey and A. Muscholl. Bounded MSC communication. In *Inf. and Comput.*, (189):135–263, 2004.
10. A. Mazurkiewicz. Concurrent program schemes and their interpretation. Technical report, DAIMI Report PB-78, Aarhus University, 1977.
11. P. Madhusudan and B. Meenakshi. Beyond Message Sequence Graphs. Proc. of *FSTTCS'01*, LNCS 2245, pp. 256-267, 2001.
12. M. Mukund. From global specification to local implementations. In *Synthesis and Control of Discrete Event Systems*, Kluwer, pp. 19-34, 2002.
13. M. Mukund, K. Narayan Kumar and M. Sohoni. Synthesizing Distributed Finite-State Systems from MSCs. *TCS* 290(1):221-239 (2003). Extended abstract in *CONCUR'00*, LNCS 1877 , pp. 521-535, 2000.
14. M. Mukund and M. Sohoni. Keeping Track of the Latest Gossip in a Distributed System. In *Distr. Computing* 10(3):137-148, 1997.
15. A. Stefanescu, J. Esparza, and A. Muscholl. Synthesis of distributed algorithms using asynchronous automata. Proc. of *CONCUR'03*, LNCS 2761, pp. 27-41, 2003.
16. W. Zielonka. Note on finite asynchronous automata. In *R.A.I.R.O. - Informatique Théorique et Applications*, 21:99-135, 1987.