Optimal Zielonka-Type Construction of Deterministic Asynchronous Automata

Blaise Genest[1,2], Hugo Gimbert[3], Anca Muscholl[3], Igor Walukiewicz[3]

[1] CNRS, IPAL UMI, joint with I2R-A*STAR-NUS, Singapore
[2] CNRS, IRISA UMR, joint with Université Rennes I, France
[3] LaBRI, CNRS/Université Bordeaux, France

Abstract. Asynchronous automata are parallel compositions of finite-state processes synchronizing over shared variables. A deep theorem due to Zielonka says that every regular trace language can be represented by a deterministic asynchronous automaton. In this paper we improve the construction, in that the size of the obtained asynchronous automaton is polynomial in the size of a given DFA and simply exponential in the number of processes. We show that our construction is optimal within the class of automata produced by Zielonka-type constructions. In particular, we provide the first non trivial lower bound on the size of asynchronous automata.

1 Introduction

Zielonka's asynchronous automata [15] is probably one of the simplest, and yet rich, models of distributed computation. This model has a solid theoretical foundation based on the theory of Mazurkiewicz traces [9,4]. The key property of asynchronous automata, known as Zielonka's theorem, is that every regular trace language can be represented by a deterministic asynchronous automaton [15]. This result is one of the central results on distributed systems and has been applied in many contexts. Its complex proof has been revisited on numerous occasions (see e.g. [2,3,12,13,6] for a selection of such papers). In particular some significant complexity gains have been achieved since the original construction. This paper provides yet another such improvement, and moreover it shows that the presented construction is in some sense optimal.

The asynchronous automata model is basically a parallel composition of finite-state processes synchronizing over shared (state) variables. Zielonka's theorem has many interpretations, here we would like to consider it as a result about distributed synthesis: it gives a method to construct a deterministic asynchronous automaton from a given sequential one and a distribution of the actions over the set of processes. We remark that in this context it is essential that the construction gives a deterministic asynchronous automaton: for a controller it is the behaviour and not language acceptance that is important. The result has applications beyond the asynchronous automata model, for example it can be used to synthesize communicating automata with bounded communication channels

[11,7] or existentially-bounded channels [5]. Despite these achievements, from the point of view of applications, the biggest problem of constructions of asynchronous automata is considered to be their high complexity. The best constructions give either automata of size doubly exponential in the number of processes, or exponential in the size of the sequential automaton.

This paper proposes an improved construction of deterministic asynchronous automata. It offers the first algorithm that gives an automaton of size *polynomial* in the size of the sequential automaton and exponential in the number of processes. We show that this is optimal for Zielonka-type constructions. Namely constructions where each component has complete information about his history. For this we introduce the notion of *locally rejecting* asynchronous automaton and remark that all Zielonka-type constructions produce this kind of automata. To be locally rejecting means that a process should reject as soon as its history tells him that no more accepting extension exists. We believe that a locally rejecting behavior is quite desirable for applications, such as monitoring or control. We show that when transforming a deterministic word automaton to a deterministic locally rejecting automaton, the exponential blow-up in the number of components is unavoidable. Thus, to improve our construction one would need to construct automata that are not locally rejecting. However no general tools for doing this are available at present.

For the upper bound we start from a *deterministic* (I-diamond) word automaton. We think that this is the best point of departure for a study of the complexity of constructing asynchronous automata: considering non deterministic automata would introduce costs related to determinization. The size of the deterministic asynchronous automaton obtained is measured as the sum of the sizes of the local states sets. It means that we do not take global accepting states into account. We think that this is reasonable as it is hardly practical to list these states explicitly. From a deterministic I-diamond automaton $\mathcal{A}$ and a distributed alphabet with process set $\mathcal{P}$, we construct a deterministic asynchronous automaton of size $2^{2\cdot|\mathcal{P}|^4} \cdot |\mathcal{A}|^{|\mathcal{P}|^2}$. We believe that this complexity, although exponential in the number of processes, is interesting in practice. If we want to implement such a device then it will need memory of size logarithmic in $|\mathcal{A}|$ and polynomial in $|\mathcal{P}|$. We also show that computing the next state on-the-fly can be done in time polynomial in both $|\mathcal{A}|$ and $|\mathcal{P}|$.

Related work. Besides the general constructions of Zielonka type, there are a couple of different constructions, however they either apply to subclasses of regular trace languages, or they produce non deterministic automata (or both). The first category includes [10,3], that provide deterministic asynchronous cellular automata from a given trace homomorphism in case that the dependence alphabet is acyclic and chordal, respectively. These constructions are quite simple and only polynomial in the size of the monoid (thus still exponential in the size of a DFA). In the second category we find [16], who gives an inductive construction for non deterministic, deadlock-free asynchronous cellular automata. (A deadlock-free variant of Zielonka's construction was proposed in [14]). The paper [1] proposes a construction of asynchronous automata of size exponential only in the number of processes (and polynomial in $|\mathcal{A}|$) as our construction, but it yields non deterministic asynchronous automata (inappropriate for monitoring or control). Notice that while asynchronous automata can be determinized, there are cases where the blow-up is doubly exponential in the number of processes [8].

2 Preliminaries

We fix a finite set $\mathcal{P}$ of processes and a finite alphabet Σ. Each letter $a \in \Sigma$ is an action associated with the set of processes $\mathrm{dom}(a) \subseteq \mathcal{P}$ involved in its execution. A pair (Σ, dom) is called *distributed alphabet*. A deterministic automaton over the alphabet Σ is a tuple $\mathcal{A} = \langle Q, \Sigma, \Delta, q^0, F\rangle$ with a finite set of states Q, a set of final states F, an initial state q^0 and a transition function $\Delta : Q \times \Sigma \to Q$. As usual we extend Δ to words in Σ^*. We use $\mathcal{L}(\mathcal{A})$ to denote the language accepted by $\mathcal{A}$. The automaton accepts $w \in \Sigma^*$ if $\Delta(q^0, w) \in F$. The size $|\mathcal{A}|$ of $\mathcal{A}$ is the number of its states.

Concurrent systems with shared actions given by a distributed alphabet (Σ, dom), are readily modeled by Mazurkiewicz traces [9]. 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 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*. 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 $[u]$, as 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 $v \sim u$. For two prefixes T_1, T_2 of T, we let $T_1 \cup T_2$ denote the smallest prefix T' of T such that $T_i \leq T'$ for $i = 1, 2$.

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 Figure 1 we give an example for the pomset of a trace T, depicted by its Hasse diagram. A total order $e_1 \cdots e_n$ that is compatible with $\leq$ is called a *linearization* of T.

An automaton $\mathcal{A}$ is called *I-diamond* if for all $(a, b) \in I$, and s a state of $\mathcal{A}$: $\Delta(s, ab) = \Delta(s, ba)$. Note that the I-diamond property implies that the language 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 $u \sim v$. This permits us to write $\Delta(s, T)$ where T is a trace, to denote the state reached by $\mathcal{A}$ from s on some linearization of T. Languages of I-diamond automata is called *regular trace languages.*

Definition 1. *A deterministic asynchronous automaton over the distributed alphabet (Σ, dom) is a tuple $\mathcal{B} = \langle (S_p)_{p\in\mathcal{P}}, (\delta_a)_{a\in\Sigma}, s^0, Acc\rangle$ where:*

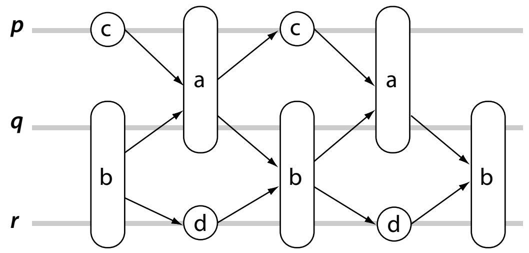

Fig. 1. The pomset associated with the trace $T = [c\,b\,a\,d\,c\,b\,a\,d\,b]$, with $\mathrm{dom}(a) = \{p, q\}$, $\mathrm{dom}(b) = \{q, r\}$, $\mathrm{dom}(c) = \{p\}$, $\mathrm{dom}(d) = \{r\}$.

- S_p *is the finite set of local states of a process* $p \in \mathcal{P}$,
- $\delta_a : \prod_{q \in dom(a)} S_q \to \prod_{q \in dom(a)} S_q$ *is the local transition function associated with an action* $a \in \Sigma$,
- $s^0 \in \prod_{p \in \mathcal{P}} S_p$ *is the global initial state,*
- $Acc \subseteq \prod_{p \in \mathcal{P}} S_p$ *is a set of global accepting states.*

We call $\prod_{p \in \mathcal{P}} S_p$ the set of *global states* (whereas S_p is the set of p-local states). In this paper the size of an asynchronous automaton $\mathcal{B}$ is the total number of *local states* $\sum_{p \in \mathcal{P}} |S_p|$. This definition is very conservative, as one may want to count also Acc or the transition functions (which can be exponential in $|\mathcal{B}|$). We will see that our construction allows to compute both Acc and the transition functions in polynomial time.

With the asynchronous automaton $\mathcal{B}$ one can associate a *global automaton* $\mathcal{A}_\mathcal{B} = \langle Q, \Sigma, \Delta, q^0, \mathrm{Acc} \rangle$ where:

- The set of states is the set of global states $Q = \prod_{p \in \mathcal{P}} S_p$ of $\mathcal{B}$, the initial and the accepting states are as in $\mathcal{B}$.
- The transition function $\Delta : Q \times \Sigma \to Q$ is defined by $\Delta(s, a) = (s'_p)_{p \in \mathcal{P}}$ with $(s'_p)_{p \in \mathrm{dom}(a)} = \delta_a((s_p)_{p \in \mathrm{dom}(a)})$ and $s'_p = s_p$, for every $p \notin \mathrm{dom}(a)$.

Clearly $\mathcal{A}_\mathcal{B}$ is a finite deterministic automaton with the I-diamond property.

Definition 2. *The language of an asynchronous automaton* $\mathcal{B}$ *is the language of the associated global automaton* $\mathcal{A}_\mathcal{B}$.

We conclude this section by introducing some notions that are basic ingredients of the common constructions of asynchronous automata. For a trace T, we denote by $\mathrm{dom}(T) = \bigcup_{e \in T} \mathrm{dom}(\lambda(e))$ the set of processes occurring in T. For a process $p \in \mathcal{P}$, we denote by $\mathrm{pref}_p(T)$ the minimal trace prefix of T containing all events of T on process p. Hence, $\mathrm{pref}_p(T)$ has a unique maximal event that is the last (most recent) event of T on process p. This maximal event is denoted as $\mathrm{last}_p(T)$. Intuitively, $\mathrm{pref}_p(T)$ corresponds to the history of process p after executing T. We extend this notation to a set of processes $P \subseteq \mathcal{P}$ and denote by $\mathrm{pref}_P(T)$ the minimal trace prefix containing all events of T on processes from P. For example, in Figure 1 we have $\mathrm{pref}_p(T) = [cbadcba]$ and $\mathrm{last}_p(T)$ is the second a of the pomset.

3 Zielonka-type constructions: state of the art

All general constructions of deterministic asynchronous automata basically follow the main ideas of the original construction of Zielonka [15]. These constructions start with a regular, I-closed word language, that is given either by a homomorphism to a finite monoid, or by an I-diamond automaton. In most applications we are interested in the second case, where we start with a (possibly non deterministic) automaton. The general constructions yield either asynchronous automata as defined in the previous section, or asynchronous *cellular* automata, that correspond to a concurrent-read-owner-write model.

Theorem 1. *[15] Let $\mathcal{A}$ be an I-diamond automaton over the independence alphabet (Σ, I). A deterministic asynchronous automaton $\mathcal{B}$ can be effectively constructed with $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$.*

We now review the constructions of [2,12,6] and recall their complexities. It is well known that determinization of word automata requires an exponential blow-up, hence the complexity of going from a non deterministic I-diamond automaton $\mathcal{A}$ to a deterministic asynchronous automaton is at least exponential in $|\mathcal{A}|$. In that case, [6] gives an optimal construction in that it is simply exponential. Since determinization has little to do with concurrency, we assume from now on that $\mathcal{A}$ is a deterministic automaton.

– [2] introduces asynchronous mappings and constructs asynchronous cellular automata of size $|\Sigma|^{|\Sigma|^2} \cdot |\mathcal{A}|^{2^{|\Sigma|}}$.
– [12] constructs asynchronous automata of size $|\mathcal{P}|^{|\mathcal{P}^2|} \cdot |\mathcal{A}|^{|\mathcal{A}| \cdot 2^{|\mathcal{P}|}}$.
– [6] introduces zone decompositions and constructs asynchronous automata of size $2^{3|\mathcal{P}^3|} \cdot |\mathcal{A}|^{|\mathcal{A}| \cdot |\mathcal{P}|^2}$.

Comparing our present construction with previous ones, we obtain asynchronous automata of size $2^{2|\mathcal{P}|^4} \cdot |\mathcal{A}|^{|\mathcal{P}|^2}$. In all these constructions, the obtained automata are such that every process knows the state reached by $\mathcal{A}$ on its history. We abstract this property below, and show in the following section that our construction is optimal in this case.

Definition 3. *A deterministic asynchronous automaton $\mathcal{B}$ is called* locally rejecting *if for every process p, there is a set $R_p \subseteq S_p$ such that for every trace T:*

$$\mathit{pref}_p(T) \notin \mathit{pref}(\mathcal{L}(\mathcal{B})) \text{ iff the } p\text{-local state reached by } \mathcal{B} \text{ on } T \text{ is in } R_p.$$

Notice that R_p is a trap: if $\mathcal{B}$ reaches R_p on trace T, then so it does on every extension of $T \leq T'$. Obviously, no accepting global state of $\mathcal{B}$ has a component in R_p. For these reasons we call states of R_p rejecting.

Let us justify the interest in locally rejecting automata by an observation that all general constructions [15,2,3,13,12,6] of deterministic asynchronous automata produce such automata. Let us assume that $\mathcal{A}$ is a (possibly non deterministic) I-diamond automaton, and $\mathcal{B}$ a deterministic asynchronous automaton produced by one of the constructions in [15,13,12,6] (a similar statement applies to the asynchronous cellular automata in [2,3]). Then the local p-state s_p reached by $\mathcal{B}$ after processing the trace T determines the set of states reached by $\mathcal{A}$ on $\mathrm{pref}_p(T)$, for every process p. Thus, if no state in this set can reach a final state of $\mathcal{A}$, then we put s_p in R_p. This makes $\mathcal{B}$ locally rejecting.

4 An exponential lower bound

In this section we present our lower bound result. We show that transforming an I-diamond deterministic automaton into a locally rejecting asynchronous automaton may induce an exponential blow-up in the number of processes. For this we define a family of languages Path_n, such that the minimal sequential automaton for Path_n has size $\mathcal{O}(n^2)$ but every locally rejecting automaton recognizing Path_n is of size at least $2^{n/4}$.

Let $\mathcal{P} = \{1, \ldots, n\}$ be the set of processes. The letters of our alphabet are pairs of processes, two letters are dependent if they have a process in common. Formally, the distributed alphabet is $\Sigma = \binom{\mathcal{P}}{2}$ with $\mathrm{dom}(\{p, q\}) = \{p, q\}$.

The language Path_n is the set of traces $[x_1 \cdots x_k]$ such that every two consecutive letters have a process in common: $x_i \cap x_{i+1} \neq \emptyset$ for $i = 1, \ldots, k-1$. Observe that a deterministic sequential automaton recognizing this language simply needs to remember the last letter it has read. So it has less than $|\mathcal{P}|^2$ states.

Theorem 2. *Every locally rejecting asynchronous automaton recognizing $Path_n$ is of size at least $2^{n/4}$.*

Proof. Take a locally rejecting automaton recognizing Path_n. Without loss of generality we suppose that $n = 4k$. To get a contradiction we suppose that process n of this automaton has less than 2^k (local) states.

We define for every integer $0 \leq m < k$ two traces: $a_m = \{4m, 4m+1\}\{4m+1, 4m+2\}\{4m+2, 4m+4\}$ and $b_m = \{4m, 4m+1\}\{4m, 4m+3\}\{4m+3, 4m+4\}$. To get some intuition, the reader may depict traces a_0 and b_0 and see that both a_0 and b_0 form a path from process 0 to process 4, the difference is that trace a_0 goes through process 2 while trace b_0 goes through process 3.

Consider the language L defined by the regular expression $(a_0 + b_0)(a_1 + b_1) \cdots (a_{k-1} + b_{k-1})$. Clearly, language L is included in Path_n and contains $2^k = 2^{n/4}$ different traces. As we have assumed that process n has less than 2^k states, there are two different traces t_1, t_2 from L such that process n is in the same state after t_1 and t_2. For simplicity of presentation we assume that t_1 and t_2 differ on the first factor: t_1 starts with a_0, and t_2 with b_0.

We can remark that processes 0 and n are in the same state after reading $t_1\{0, 3\}$ and t_2. For process 0 it is clear as in both traces it sees the same trace $\{0, 1\}\{0, 3\}$. By our hypothesis, process n is in the same local state after traces t_1 and t_2, therefore also after traces $t_1\{0, 3\}$ and t_2.

Consider now the state s_n reached by n after reading $t_2\{0, n\}$. Since $t_2\{0, n\} \in \mathrm{Path}_n$, we have $s_n \notin R_n$. By the above, the same state s_n is also reached after reading $t_1\{0,3\}\{0,n\}$. Trace t_1 starts with $a_0 = \{0,1\}\{1,2\}\{2,4\}$ and continues with processes whose numbers are greater than 4, so $\{0,3\}$ commutes with all letters of t_1 except $\{0,1\}$. Hence $t_1\{0,3\} \notin \mathrm{pref}(\mathrm{Path}_n)$. Since trace t_1 ends with an action of process n, we have $\mathrm{pref}_n(t_1\{0,3\}\{0,n\}) = t_1\{0,3\}\{0,n\} \notin \mathrm{pref}(\mathrm{Path}_n)$. Since we have assumed that the automaton is locally rejecting, $s_n \in R_n$. A contradiction. $\square$

5 A matching upper bound

Our goal is to modify the construction from [6] in order to make it polynomial with respect to the size of the sequential automaton. We give an overview of the new construction, first describing the objects the asynchronous automaton manipulates. Some details of the mechanics of the automaton will follow (a more detailed presentation can be found in the appendix).

We fix a set of processes $\mathcal{P}$ and a distributed alphabet (Σ, dom). Let $\mathcal{A} = \langle Q, \Sigma, \Delta, q^0, F\rangle$ be a deterministic I-diamond automaton. A candidate for an equivalent asynchronous automaton $\mathcal{B} = \langle (S_p)_{p\in\mathcal{P}}, (\delta_a)_{a\in\Sigma}, s^0, \mathrm{Acc}\rangle$ has a set of states for each process and a local transition function. The goal is to make $\mathcal{B}$ calculate the state reached by $\mathcal{A}$ after reading a linearization of a trace T. Let us examine how $\mathcal{B}$ can accomplish this task. After reading a trace T the local state of a component p of $\mathcal{B}$ depends only on $\mathrm{pref}_p(T)$. Hence, $\mathcal{B}$ can try to calculate the state reached by $\mathcal{A}$ after reading (some linearization of) $\mathrm{pref}_p(T)$. When a next action, say a, is executed, processes in $\mathrm{dom}(a)$ can see each others' states and make the changes accordingly. Intuitively, this means that these processes can now compose their information in order to calculate the state reached by $\mathcal{A}$ on $T' = \mathrm{pref}_{\mathrm{dom}(a)}(T)\, a$. To do so they will need some information about the structure of the trace.

As usual, the tricky part of this process is to reconstruct the common view of $\mathrm{pref}_{\mathrm{dom}(a)}(T)$ from separate views of each process: $\mathrm{pref}_p(T)$ for $p \in \mathrm{dom}(a)$. For the sake of example suppose that $\mathrm{dom}(a) = \{p,q,r\}$, and we know the states reached by $\mathcal{A}$ after reading $\mathrm{pref}_p(T)$, $\mathrm{pref}_q(T)$, and $\mathrm{pref}_r(T)$. We would like to know the state of $\mathcal{A}$ after reading $\mathrm{pref}_{\{p,q,r\}}(T)$. This would be possible if we could compute contributions of $\mathrm{pref}_q(T) \setminus \mathrm{pref}_p(T)$ and $\mathrm{pref}_r(T) \setminus \mathrm{pref}_{\{p,q\}}(T)$. The automaton $\mathcal{B}$ should be able to do this by looking at s_p, s_q, and s_r, only. This remark points out the challenge of the construction: find the type information that allows to deduce the behaviour of $\mathcal{A}$, and that at the same time is updatable by an asynchronous automaton, see e.g. [13]. As in all general Zielonka constructions, we use the fact that e.g. $\mathrm{pref}_r(T) \cap \mathrm{pref}_{\{p,q\}}(T)$ is uniquely determined by $\mathrm{last}(\mathrm{pref}_r(T)) \cap \mathrm{last}(\mathrm{pref}_{\{p,q\}}(T))$ ([15], see also [13]). By $\mathrm{last}(U)$ we denote the set of events $\{\mathrm{last}_p(U) \mid p \in \mathcal{P}\}$.

5.1 General structure

Before introducing formal definitions it may be worth to say what is the general structure of the states of the automaton $\mathcal{B}$. Every local state will be a triple $(ts, ZO, \overline{\Delta})$, where

- ts will be a time stamping information as in all general constructions of asynchronous automata;
- ZO will be a zone order, a bounded size partial order on a partition of the trace;
- $\overline{\Delta}$ will be state information, recording the behavior of $\mathcal{A}$ on the partition given by ZO.

Roughly, we will use time stamping to compute zone orders, and zone orders to compute state information. The latter provides all the necessary information about the behaviour of $\mathcal{A}$ on (a linearization of) the trace.

Timestamping: The goal of the *time stamping function* [15] is to determine for a set of processes P and a process q the set $\mathrm{last}(\mathrm{pref}_P(T)) \cap \mathrm{last}(\mathrm{pref}_q(T))$. This set uniquely determines the intersection of $\mathrm{pref}_P(T)$ and $\mathrm{pref}_q(T)$ (for details see e.g. [13]). Computing such intersections is essential when composing information about $\mathrm{pref}_p(T)$ for every $p \in \mathrm{dom}(a)$ to information about $\mathrm{pref}_{\mathrm{dom}(a)}(T)$. The main point is that there exists a deterministic asynchronous automaton that can accomplish this task (for a formal description see the appendix). Each of its local states can be described with $O(|\mathcal{P}|^2 \log(|\mathcal{P}|)$ bits.

For instance, if a new b is executed after $T = [cbadcbad]$ in Figure 2, processes q, r determine that the intersection of their last-sets consists of the second b. Indeed, $\mathrm{last}(\mathrm{pref}_q(T))$ is made of the second a (for $\mathrm{last}_p = \mathrm{last}_q$) and the second b (for last_r). Also, $\mathrm{last}(\mathrm{pref}_r(T))$ is made of the second d (for last_r), the second b (for last_q) and the first a (for last_p).

Zone orders: Recall that one of our objectives is to calculate, for every $p \in \mathcal{P}$, the state reached by $\mathcal{A}$ on $\mathrm{pref}_p(\mathcal{P})$. As the discussion on page 7 pointed out, for this we may need to recover the transition function of $\mathcal{A}$ associated with $\mathrm{pref}_q(T) \setminus \mathrm{pref}_P(T)$ for a process q and a set of processes P. Hence we need to store information about the behaviour of $\mathcal{A}$ on some relevant factors of T that are not prefixes. Zones are such relevant factors. They are defined in such a way that there is a bound on the number of zones in a trace. The other crucial property of zones is that for every extension T' of T and $P \subseteq \mathcal{P}, q \in \mathcal{P}$, if a zone of T intersects $\mathrm{pref}_q(T') \setminus \mathrm{pref}_P(T')$ then it is entirely in this set. A zone order will be an abstract representation of the decomposition of a trace into zones.

Definition 4. *[6] Let $T = \langle E, \leq, \lambda \rangle$ be a trace. For an event $e \in E$ we define the set of events $L(e) = \{f \in \mathrm{last}(T) \mid e \leq f\}$. We say that two events e, e' are equivalent (denoted as $e \equiv e'$) if $L(e) = L(e')$. The equivalence classes of $\equiv$ are called* zones. *The set of processes that are active in Z is denoted by $dom(Z)$.*

There is a useful partial order on zones that we define now. Let Z, Z' be two zones of some trace T. We write $Z \lessdot Z'$ if $Z \neq Z'$ and $e < e'$ for some events $e \in Z$, $e' \in Z'$. It is easy to see that $Z \lessdot Z'$ implies that $L(Z') \subsetneq L(Z)$. Thanks to this property we can define the *order on zones*, denoted $Z \leq Z'$, as the smallest partial order containing the $\lessdot$ relation.

Lemma 1. *A trace is partitioned in at most $|\mathcal{P}|^2$ zones.*

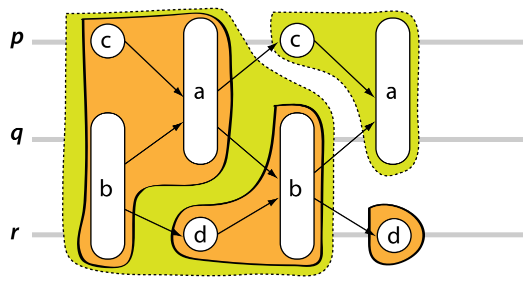

Fig. 2. The three zones of $\text{pref}_r(T)$ are marked with solid lines. The two zones of $\text{pref}_{\{p,q\}}(T)$ are represented by dotted lines.

In Appendix A we show a trace with $\Omega(|\mathcal{P}|^2)$ zones. Figure 2 depicts the trace $T = [cbadcbad]$. Recall that $\text{last}(\text{pref}_r(T))$ consists of the first a, the second b and the second d. There are three zones in $\text{pref}_r(T)$: Z_1 contains the first a, b and c, Z_2 the first d and the second b, and Z_3 the second d. We have $Z_1 < Z_2 < Z_3$.

Definition 5. *A* zone order *is a labeled partial order $ZO = \langle V, \leq, \xi\colon V \to 2^{\mathcal{P}} \rangle$, where every element is labeled by a set of processes. We require that every two elements whose labels have non empty intersection are comparable: $\xi(v) \cap \xi(v') \neq \emptyset \;\Rightarrow\; (v \leq v' \vee v' \leq v)$. We say that such a zone order is the* zone order of a trace T*, if there is a bijection μ from V to zones of T preserving the order and satisfying $\xi(v) = dom(\mu(v))$.*

Lemma 2. *The zone order of a trace can be stored in $|\mathcal{P}|^2(|\mathcal{P}|^2 + |\mathcal{P}|)$ space. So there are at most $2^{O(|\mathcal{P}|^4)}$ zone orders.*

State information: We describe now the state information for each zone of the trace. Let $ZO = \langle V, \leq, \xi : V \to 2^{\mathcal{P}} \rangle$ be the zone order of some trace T, via a bijection μ. For an element $v \in V$ we denote by T_v the factor of T consisting of zones up to $\mu(v)$: that is, the factor covering $\mu(v')$ for all $v' \leq v$. Observe that T_v is a prefix of T.

Definition 6. *We say that a function $\overline{\Delta} : V \to Q$ is* state information *for the zone order ZO of a trace T if for every v we have $\overline{\Delta}(v) = \Delta(q^0, T_v)$, namely the state of $\mathcal{A}$ reached on a linearization of T_v.*

Observe that a zone order for a trace of the form $\text{pref}_p(T)$ has one maximal element v_p: it corresponds to the last action of p. If $\overline{\Delta}$ is the state information for this zone then the state reached by $\mathcal{A}$ on reading a linearization of $\text{pref}_p(T)$ is $\overline{\Delta}(v_p)$.

5.2 The construction of the asynchronous automaton

Let us come back to the description of the asynchronous automaton $\mathcal{B}$. For every $p \in \mathcal{P}$, a local state of S_p will have the form $(ts_p, ZO_p, \overline{\Delta}_p)$. The automaton will be defined in such a way that after reading a trace T the state s_p reached at the component p will satisfy:

- ts_p is the time stamping information [13];
- ZO_p is the zone order of $\mathrm{pref}_p(T)$;
- $\overline{\Delta}_p$ is the state information for ZO_p.

By [13] we know that $\mathcal{B}$ can update the ts_p component. The proposition below says that $\mathcal{B}$ can update the ZO_p and $\overline{\Delta}_p$ components (for the proof see Appendix C).

Proposition 1. *Let T be a trace and $a \in \Sigma$ an action. Suppose that for every $p \in dom(a)$ we have the time stamping information ts_p and the zone order with state information $(ZO_p, \overline{\Delta}_p)$ of $pref_p(T)$. We can then calculate the zone order and the state information of $pref_p(Ta)$, for every $p \in dom(a)$.*

We also need to define the set of local rejecting states R_p and the global accepting states Acc of $\mathcal{B}$. Observe that by Proposition 1, from the local state s_p we can calculate $\Delta(q^0, \mathrm{pref}_p(T))$, namely the state of $\mathcal{A}$ reached after reading a linearization of $\mathrm{pref}_p(T)$. This state is exactly the state associated to the unique maximal element of the zone order in s_p. Hence, $\mathcal{B}$ can be made locally rejecting by letting $s_p \in R_p$ if $\Delta(q^0, \mathrm{pref}_p(T))$ is a deadend state of $\mathcal{A}$.

To define accepting tuples of states of $\mathcal{B}$ we use the following proposition:

Proposition 2. *Let T be a trace. Given for every $p \in \mathcal{P}$ the time stamping ts_p, and the zone order ZO_p with state information $\overline{\Delta}_p$ of $pref_p(T)$, we can calculate $\Delta(q^0, T)$, the state reached by $\mathcal{A}$ on a linearization of T.*

In the light of Proposition 2, a tuple of states of $\mathcal{B}$ is accepting if the state $\Delta(q^0, T)$ of $\mathcal{A}$ is accepting. The two propositions give us:

Theorem 3. *Let $\mathcal{A}$ be a deterministic I-diamond automaton over the distributed alphabet (Σ, dom). We can construct an equivalent deterministic locally rejecting asynchronous automaton $\mathcal{B}$ with at most $2^{2|\mathcal{P}|^4} \cdot |\mathcal{A}|^{|\mathcal{P}|^2}$ states.*

We now describe informally the main ingredients of the proof of Proposition 1 (Proposition 2 goes along similar lines). The zone order ZO of $\mathrm{pref}_{P\cup\{q\}}(T)$ is built in two steps from ZO_P and ZO_q: first we construct a so-called pre-zone order ZO' by adding to ZO_P the zones from $\mathrm{pref}_q(T) \setminus \mathrm{pref}_P(T)$ [6]. Then we quotient ZO' in order to obtain ZO. The quotient operation amounts to merge zones. The difficulty compared to [6] is posed by the update of the state information. Since the state information for the pre-zone ZO' is inconsistent due to the merge, the crucial step is to compute this information on downward closed sets of zones:

Lemma 3. *Let $ZO = \langle V, \leq, \xi \rangle$, $\overline{\Delta}$ be the zone order and state information for a trace T (via the bijection μ). For every downward closed $B \subseteq V$ we can compute the state reached by $\mathcal{A}$ on a linearization of $T_B = \bigcup\{T_v \mid v \in B\}$, using only ZO and $\overline{\Delta}$.*

The proof for the lemma above is based on a nice observation about I-diamond automata $\mathcal{A}$, [2]. It says that for every three traces T_0, T_1, T_2 with $\mathrm{dom}(T_1) \cap \mathrm{dom}(T_2) = \emptyset$, the state reached by $\mathcal{A}$ on a linearization of $T_0T_1T_2$ can be computed from $\mathrm{dom}(T_1)$ and the states reached on (linearizations of) T_0, T_0T_1, T_0T_2, respectively.

We now sketch the proof of the lemma. We first choose some linearization $v_1, \ldots, v_n$ of B, and let $B_i = \{v_1, \ldots, v_i\}$. Let us write $B_{i,k}$ $(i \leq k)$ for the set $B_i \cup \{v_j \mid v_j \leq v_k, j > i\}$. We show now how to compute inductively $\Delta(q^0, T_{B_{i,k}})$.

Suppose we know already $\Delta(q^0, T_{B_{i-1,k}})$, for all $k \geq i-1$. In particular, note that the states q^{i-1}, q^i reached on $\mu(v_1 \cdots v_{i-1})$ and $\mu(v_1 \cdots v_i)$, respectively, are known (cases $k = i-1$ and $k = i$).

We compute now $\Delta(q^0, T_{B_{i,k}})$, for $k > i$. Two cases arise. If $v_i \not< v_k$ then we apply the observation of [2] to $q^{i-1}, q^i, \Delta(q^0, T_{B_{i-1,k}}), \xi(v_i)$, which yields $\Delta(q^0, T_{B_{i,k}})$. If $v_i < v_k$, then $B_{i-1,k} = B_{i,k}$ and the state $\Delta(q^0, T_{B_{i,k}})$ is already known. At the end of this polynomial time procedure, we have computed $\Delta(q^0, T_B) = \Delta(q^0, T_{B_{n,n}})$.

Remark 1. The automaton $\mathcal{B}$ of Theorem 3 can be constructed on-the-fly, i.e. given the action $a \in \Sigma$ and the local states s_p of $\mathcal{B}$, $p \in \mathrm{dom}(a)$, one can compute the successor states $\delta_a((s_p)_{p \in \mathrm{dom}(a)})$. The question is now how much time we need for this computation. The update of the time stamping and that of zone orders takes time polynomial in $|\mathcal{P}|$. The update of state information can be done in time polynomial in $|\mathcal{P}|$ and linear in the number of transitions of $|\mathcal{A}|$. So overall, we can compute transitions on-the-fly in polynomial time. Similarly, we can decide whether a global state is accepting in polynomial time.

6 Conclusion

In this paper we presented an improved construction of asynchronous automata. Starting from a zone construction of [6], we have shown how to keep just one state per zone instead of a transition table. This allows to obtain the first construction that is polynomial in the size of the sequential automaton and exponential only in the number of processes.

It is tempting to conjecture that our construction is optimal. Unfortunately, it is very difficult to provide lower bounds on sizes of asynchronous automata. We gave a matching lower bound for the subclass of locally rejecting automata. It is worth to recall that all general constructions in the literature produce automata of this kind. Moreover the concept of locally rejecting automaton is interesting on its own from the point of view of applications.

We conjecture that the translation from deterministic word automata to asynchronous automata must be exponential in the number of processes; where the size means the total number of local states.

References

1. N. Baudru. Distributed asynchronous automata. In *CONCUR'09*, number 5710 in LNCS, pages 115–130. Springer, 2009.
2. R. Cori, Y. Métivier, and W. Zielonka. Asynchronous mappings and asynchronous cellular automata. *Information and Computation*, 106:159–202, 1993.
3. V. Diekert and A. Muscholl. Construction of asynchronous automata. In V. Diekert and G. Rozenberg, editors, *Book of Traces*, pages 249–267. World Scientific, Singapore, 1995.
4. V. Diekert and G. Rozenberg, editors. *The Book of Traces.* World Scientific, Singapore, 1995.
5. B. Genest, D. Kuske, and A. Muscholl. A Kleene theorem and model checking algorithms for existentially bounded communicating automata. *Inf. Comput.*, 204(6):920–956, 2006.
6. B. Genest and A. Muscholl. Constructing exponential-size deterministic zielonka automata. In *ICALP'06*, number 4052 in LNCS, pages 565–576. Springer, 2006.
7. J. G. Henriksen, M. Mukund, K. N. Kumar, M. Sohoni, and P. S. Thiagarajan. A theory of regular MSC languages. *Inf. Comput.*, 202(1):1–38, 2005.
8. N. Klarlund, M. Mukund, and M. Sohoni. Determinizing asynchronous automata. In *ICALP'94*, number 820 in LNCS, pages 130–141. Springer, 1994.
9. A. Mazurkiewicz. Concurrent program schemes and their interpretations. DAIMI Rep. PB 78, Aarhus University, Aarhus, 1977.
10. Y. Métivier. An algorithm for computing asynchronous automata in the case of acyclic non-commutation graph. In *ICALP'87*, number 267 in LNCS, pages 226–236. Springer, 1987.
11. M. Mukund, K. N. Kumar, and M. Sohoni. Synthesizing distributed finite-state systems from MSCs. In *CONCUR'00*, number 1877 in LNCS, pages 521–535. Springer, 2000.
12. M. Mukund and M. Sohoni. Gossiping, asynchronous automata and Zielonka's theorem. Report TCS-94-2, School of Mathematics, SPIC Science Foundation, Madras, India, 1994.
13. M. Mukund and M. A. Sohoni. Keeping track of the latest gossip in a distributed system. *Distributed Computing*, 10(3):137–148, 1997.
14. A. Stefanescu. *Automatic synthesis of distributed transition systems.* PhD thesis, Universität Stuttgart, 2006.
15. W. Zielonka. Notes on finite asynchronous automata. *RAIRO–Theoretical Informatics and Applications*, 21:99–135, 1987.
16. W. Zielonka. Safe executions of recognizable trace languages by asynchronous automata. In *Symposium on Logical Foundations of Computer Science, Logic at Botik '89, Pereslavl-Zalessky (USSR)*, number 363 in LNCS, pages 278–289. Springer, 1989.

A A trace with many zones

Here is an example showing that there can be $\mathcal{O}(|\mathcal{P}|^2)$ zones in a trace. As set of processes we take $\mathcal{P} = \{0_x, 1_x, \dots, n_x\} \cup \{0_y, 1_y, \dots, n_y\}$; that is two copies of the initial segment of natural numbers. The alphabet Σ is $\{(i_x, j_y) \mid i, j = 0, \dots, n\}$. So a letter is a pair of processes, and naturally, the domain of such a letter are those two processes.

We consider a trace T where there is one event for every letter from Σ. For simplicity of notation we will identify the events with letters. The order between events is determined by the rule

- $(i_x, j_y) \leq (k_x, l_y)$ if $i \geq k$ and $j \geq l$; notice the inversion of orders.

The trace is depicted in Figure 3 where the subscripts x and y are omitted for readability. It is easy to verify that every event (i_x, j_y) is a zone by itself with $L(i_x, j_y) = \{0_x, \dots, i_x\} \cup \{0_y, \dots, j_y\}$.

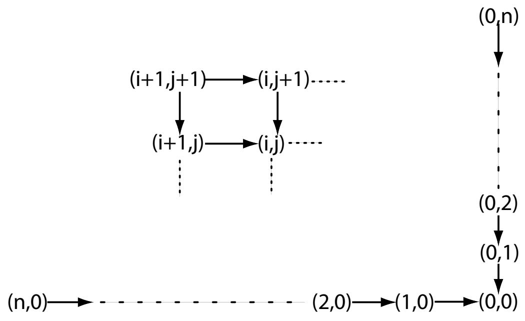

Fig. 3. A trace with a big zone graph

B Time stamping

Theorem 4. *[13] There exist a deterministic asynchronous automaton $\mathcal{A}_{TS} = \langle (S_p)_{p\in\mathcal{P}}, (\Delta_a)_{a\in\Sigma}, s^0 \rangle$ such that for every trace T and state $s = \Delta(s^0, T)$ reached by $\mathcal{A}_{TS}$ after reading T:*

> *for every $P \subseteq \mathcal{P}$ and $q, r, r' \in \mathcal{P}$, the set of local states $\{s_p \mid p \in P \cup \{q\}\}$ allows to determine if $\text{last}_r(\mathit{pref}_P(T)) = \text{last}_{r'}(\mathit{pref}_q(T))$.*

Moreover, such $\mathcal{A}_{TS}$ can be effectively computed and its local states can be described using $O(|\mathcal{P}|^2 \log(|\mathcal{P}|))$ bits.

C Updating state and zone information

In this section we show how an asynchronous automaton updates the zone order and the state information (Proposition 1). We will get also the proof of Proposition 2 as a side result.

Suppose that we extend a trace T by an action $a \in \Sigma$. We first need to construct the zone order and the state information of $\mathrm{pref}_{\mathrm{dom}(a)}(T)a$, given the zone orders and state information of each of $\mathrm{pref}_p(T)$, $p \in \mathrm{dom}(a)$. The following proposition implies that this is possible.

Proposition 3. *Let $P \subseteq \mathcal{P}, q \in \mathcal{P}$ and let T be a trace. Assume that we are given the zone orders with state information $(ZO_P, \overline{\Delta}_P), (ZO_q, \overline{\Delta}_q)$ for $pref_P(T)$ and $pref_q(T)$, respectively. Suppose that we also know the local states $(s_p)_{p\in P}, s_q$ reached by the time stamping automaton $\mathcal{A}_{TS}$ on $pref_P(T)$ and $pref_q(T)$, respectively. Then we can compute the zone order and the state information $(ZO, \overline{\Delta})$ for $pref_{P\cup\{q\}}(T)$.*

The proof of Proposition 3 will occupy most of this section. We start with the following crucial property of zones, that shows that the zone update can be performed without splitting zones:

Proposition 4 ([6], Props. 1, 3). *Let T be a trace, $P \subseteq \mathcal{P}$ and $q \in \mathcal{P}$. For every zone Z of $pref_P(T)$, we have either $Z \subseteq pref_P(T) \cap pref_q(T)$ or $Z \subseteq pref_P(T) \setminus pref_q(T)$. Moreover, if Z is a zone of $pref_P(T)$ or $pref_q(T)$ then Z is a factor of some zone of $pref_{P\cup\{q\}}(T)$.*

We need also to determine which zones of $\mathrm{pref}_q(T)$ are within $\mathrm{pref}_P(T) \cap \mathrm{pref}_q(T)$. This task uses the time stamping automaton $\mathcal{A}_{\mathrm{TS}}$ (cf. Theorem 4):

Lemma 4. *Let Z be a zone of $pref_q(T)$. Then $Z \subseteq pref_P(T) \cap pref_q(T)$ if and only if $Z \leq Z'$ for some zone Z' of $pref_q(T)$ such that $dom(Z')$ contains processes $r, r' \in \mathcal{P}$ with $\mathrm{last}_r(pref_P(T)) = \mathrm{last}_{r'}(pref_q(T))$.*

Proof. We use the following basic fact (see e.g. [13]):

$$\max(\mathrm{pref}_P(T) \cap \mathrm{pref}_q(T)) \subseteq \mathrm{last}(\mathrm{pref}_P(T)) \cap \mathrm{last}(\mathrm{pref}_q(T))$$

Thus, $Z \subseteq \mathrm{pref}_P(T) \cap \mathrm{pref}_q(T)$ if and only if $Z \leq Z'$ for some zone Z' that contains an event e in $\max(\mathrm{pref}_P(T) \cap \mathrm{pref}_q(T))$. For such an event e, there exist r, r' with $e = \mathrm{last}_r(\mathrm{pref}_P(T)) = \mathrm{last}_{r'}(\mathrm{pref}_q(T))$. □

We now come back to the proof of Proposition 3. Given the zone orders $ZO_P = \langle V_P, \leq_P, \xi_P\rangle$ and $ZO_q = \langle V_q, \leq_q, \xi_q\rangle$, we build the required $ZO = \langle V, \leq, \xi\rangle$ in two steps: (i) we construct an intermediate zone order ZO' by adding a part of ZO_q to ZO_P, then (ii) we need to quotient ZO' in order to obtain ZO.

The first step is to add to ZO_P the zones of ZO_q included in $\mathrm{pref}_q(T) \setminus \mathrm{pref}_P(T)$. Let $W' \subseteq V_q$ denote the set of zones of ZO_q satisfying Lemma 4, and let $W = V_q \setminus W'$. The zone order $ZO' = \langle V', \leq', \chi'\rangle$ is defined by:

- $V' = V_P \cup W$,
- $\leq'$ is the least partial order containing the orders of ZO_P and ZO_q, and such that $\xi_P(v) \cap \xi_q(w) \neq \emptyset$ implies $v \leq' w$, for all $v \in V_P, w \in W$,
- $\xi'(v) = \xi_P(v)$ for $v \in V_P$, and $\xi'(v) = \xi_q(v)$ for $v \in W$.

By assumption we have a bijection μ_P between ZO_P and the zones of $\mathrm{pref}_P(T)$; and a bijection μ_q between ZO_q and the zones of $\mathrm{pref}_q(T)$. We can define a bijection μ' from ZO' to factors of T, by simply using μ_P for $v' \in V_P$ and μ_q for $v' \in W$.

It should be of no surprise that ZO' is in general *not* the zone order of $\mathrm{pref}_{P\cup\{q\}}(T)$. This order has some weaker property though, that we summarize in the following definition.

Definition 7. *A zone order $ZO = \langle V, \leq, \xi\colon V \to 2^{\mathcal{P}}\rangle$ is an* pre-zone order of a trace T *if there is a bijection μ from V to factors of T such that*

- *μ preserves order and domains: $\xi(v) = dom(\mu(v))$;*
- *$(\mu(v))_{v\in V}$ is a partition of T into factors, and every one of them is a factor of a zone of T.*

Lemma 5. *ZO' is an pre-zone order of $\mathit{pref}_{P\cup\{q\}}(T)$, via the mapping μ'.*

Proof. By definition it is clear that μ' respects the domains and that its values are factors in T, as they are zones in $\mathrm{pref}_P(T)$ or $\mathrm{pref}_q(T)$. By Proposition 4 we know that such zones are factors of zones in $\mathrm{pref}_{P\cup\{q\}}(T)$. It remains to show that μ' preserves order. For this suppose that $v_1' \leq' v_2'$ in ZO'. If the two elements come from V_P then it is clear, as $\mu'(v_1') = \mu_P(v_1') \leq_P \mu_P(v_2') = \mu'(v_2')$. Similarly if the two elements come from W. The last possible case is when $v_1' \in V_p$ and $v_2' \in W$. We suppose that $\xi_P(v_1') \cap \xi_q(v_2') \neq \emptyset$; the general case being an obvious extension. By preservation of domains we get $\mathrm{dom}(\mu'(v_1')) \cap \mathrm{dom}(\mu'(v_2')) \neq \emptyset$. Let r be a process belonging to this intersection and let $e_1 \in \mu'(v_1')$ and $e_2 \in \mu'(v_2')$ be events involving r. These two events are ordered. Since by definition of W, e_2 does not belong to $\mathrm{pref}_P(T)$, we have $e_1 \leq e_2$. Hence $\mu'(v_1') \leq' \mu'(v_2')$. □

Observe that this lemma implies that every zone of $\mathrm{pref}_{P\cup\{q\}}(T)$ is a union of factors of ZO'. In a second step we show how to glue together these factors to obtain the zone order of $\mathrm{pref}_{P\cup\{q\}}(T)$. We define the following equivalence relation $\equiv$ on $V' \times V'$: let $v_1 \equiv v_2$ if $\bigcup\{\xi'(w) \mid v_1 \leq' w\} = \bigcup\{\xi'(w) \mid v_2 \leq' w\}$. Then let $ZO = ZO'/_{\equiv}$ be the quotient partial order $\langle V, \leq, \xi\rangle$ with

- $V = V'/_{\equiv}$,
- $\xi([v]_{\equiv}) = \bigcup_{v'\equiv v} \xi'(v')$,
- $\leq$ is the smallest reflexive and transitive relation satisfying: $[v]_{\equiv} \leq [w]_{\equiv}$ if $v' \leq' w'$ for some $v \equiv v'$, $w \equiv w'$.

Lemma 6. *If ZO' is a pre-zone order of a trace T then the quotient $ZO = ZO'/_{\equiv}$ is the zone order of T.*

Proof. Let $ZO' = \langle V', \leq', \xi' \rangle$ be as in the assumption of the lemma; in particular let μ' be the map from V' to factors of T showing that ZO' is an pre-zone order. It is not difficult to check for $v \in V'$ that

$$\bigcup\{\xi'(w) \mid v \leq' w\} = \bigcup\{\mathrm{dom}(\mu'(w)) \mid v \leq' w\} = \mathrm{dom}(L(\mu'(v)))$$

where L refers to the function of Definition 4 w.r.t the trace T. Notice that $L \circ \mu'$ is well-defined, since each $\mu'(v)$ is included in a zone of T. Thus, $v_1 \equiv v_2$ iff $\mathrm{dom}(L(\mu'(v_1))) = \mathrm{dom}(L(\mu'(v_2)))$ iff $L(\mu'(v_1)) = L(\mu'(v_2))$ iff $\mu'(v_1)$, $\mu'(v_2)$ are included in the same zone of T. $\square$

We show in the remaining how to obtain the state information for ZO. First we introduce some notation. For $ZO_P = \langle V_P, \leq_P, \xi_P \rangle$ and $v \in V_P$ we write T^P_v for the prefix of T consisting of all factors $\mu_P(w)$, with $w \leq_P v$. More generally, for a downward closed subset B of ZO_P we denote by T^P_B the prefix of T consisting of all zones $\mu_P(v)$ for $v \in B$. Similarly, we define T^q_v using ZO_q and μ_q. Finally, T_v is defined using ZO and μ. The notations T^q_B and T_B are also evident.

Calculating the state information for ZO means to calculate for every $v \in V$ the state $\Delta(q^0, T_v)$ reached by $\mathcal{A}$ on a linearization of T_v. The following lemma says that this amounts to compute the state information for downward closed subsets of V_P and V_q, respectively. Recall first that $V' = V_P \cup W$ where $W \subseteq V_q$ is the set of zones of $\mathrm{pref}_q(T)$ included in $\mathrm{pref}_q(T) \setminus \mathrm{pref}_P(T)$. We denote by $[W]_\equiv$ the set $\{[w]_\equiv \mid w \in W\}$.

Lemma 7. *For every $v \in V \setminus [W]_\equiv$, there is a downward closed set B of ZO_P such that $T_v = T^P_B$. For every $v \in [W]_\equiv$ there is a downward closed set B of ZO_q such that $T_v = T^q_B$.*

Proof. By definition, T_v is a union of zones of $\mathrm{pref}_{P\cup\{q\}}(T)$. By Proposition 4, T_v is either included in $\mathrm{pref}_P(T)$ or $\mathrm{pref}_q(T)$. Suppose it is included in $\mathrm{pref}_P(T)$, the other case being similar. The same Proposition 4 says that T_v is union of zones of $\mathrm{pref}_p(T)$. Let $B = \{w \mid \mu_P(w) \subseteq T_v\}$. It is straightforward to check that B is downward closed in ZO_P. $\square$

Lemma 7 suggests that it is interesting to calculate $\Delta(q^0, T^P_B)$ for a downward closed subset B of $\mathrm{pref}_P(T)$; and $\Delta(q^0, T^q_B)$ as well. Lemma 9 below shows that this is indeed possible. It is based on the following useful observation about I-diamond automata.

Lemma 8. *[2] Let $\mathcal{A} = \langle Q, \Sigma, \Delta, q^0, F \rangle$ be a deterministic I-diamond automaton. There is a function $Diam : Q^3 \times 2^{\mathcal{P}} \to Q$ such that for every three states q_0, q_1, q_2 of $\mathcal{A}$ and a set of processes X, the state $q = Diam(q_0, q_1, q_2, X)$ satisfies the property:*

> *For all traces T_0, T_1, T_2 with $dom(T_1) \subseteq X$ and $dom(T_2) \subseteq \mathcal{P} \setminus X$, if $\Delta(q^0, T_0) = q_0$, $\Delta(q^0, T_0T_1) = q_1$, and $\Delta(q^0, T_0T_2) = q_2$, then $\Delta(q^0, T_0T_1T_2) = q$.*

Proof. Fix some q_0, q_1, q_2, and X. Take arbitrary two words w_1, w_1' such that $\mathrm{dom}(w_1), \mathrm{dom}(w_1') \subseteq X$, $\Delta(q_0, w_1) = \Delta(q_0, w_1') = q_1$; and two words w_2, w_2' such that $\mathrm{dom}(w_2), \mathrm{dom}(w_2') \subseteq \mathcal{P} \setminus X$, $\Delta(q_0, w_2) = \Delta(q_0, w_2') = q_2$. We get

$$\Delta(q_0, w_1 w_2) = \Delta(\Delta(q_0, w_1), w_2) = \Delta(\Delta(q_0, w_1'), w_2) = \Delta(q_0, w_1' w_2)\,.$$

Since w_2 and w_1' commute and $\mathcal{A}$ is I-diamond we have $\Delta(q_0, w_1' w_2) = \Delta(q_0, w_2 w_1')$. Repeating the reasoning similarly to the above we get $\Delta(q_0, w_2 w_1') = \Delta(q_0, w_2' w_1') = \Delta(q_0, w_1' w_2')$. □

Lemma 9. *Let $(ZO, \overline{\Delta})$ be the zone order and state information for a trace T. For every downward closed $B \subseteq V$ we can compute $\Delta(q^0, T_B)$ using only ZO and $\overline{\Delta}$.*

Proof. Let μ be the mapping from V to factors of T showing that ZO is an pre-zone order. The proof is by induction on the number of elements in B. If there is only one then we are done, as $T_B = T_v$ for the unique $v \in B$.

For the induction step take a maximal element v of B. Let $B_1 = \{v' \mid v' \leq v\}$ be the set of elements below v; and let $B' = B \setminus B_1$. Clearly B_1 is downward closed, and so are $B_0 = B_1 \setminus \{v\}$ and $B_2 = B_0 \cup B'$. By induction assumption we can calculate $q_i = \Delta(q^0, T_{B_i})$ for $i = 0, 1, 2$. Let T_2 be the union of factors $\mu(v')$ for $v' \in B'$. We get $T_{B_1} = T_{B_0}\mu(v)$ and $T_{B_2} = T_{B_0}T_2$. Recall that the definition of a zone order requires that if $\xi(v) \cap \xi(v') \neq \emptyset$ then v and v' are comparable in the order. Hence $\xi(v) \cap \xi(B') = \emptyset$ as all the elements of B' are incomparable with v. This means that $\mathrm{dom}(\mu(v)) \cap \mathrm{dom}(T_2) = \emptyset$ because $\mathrm{dom}(\mu(v)) = \xi(v)$ and $\mathrm{dom}(T_2) = \xi(B')$. We are in a position to apply Lemma 8 obtaining $\Delta(q^0, T_B) = \mathrm{Diam}(q_0, q_1, q_2, \xi(v))$. □

We are ready to complete the proof of Proposition 3. For every $v \in ZO$ we need to calculate $\overline{\Delta}(v) = \Delta(q^0, T_v)$. By Lemma 7 we know that $T_v = T^P_B$ or $T_v = T^q_B$ for some downward closed subset of T^P or T^q respectively. In the first case $\Delta(v) = \Delta(q^0, T^P_B)$ and this is computable by Lemma 9. Similarly in the second case.

Remark 2. The update algorithm given by Lemma 9 is exponential in the size of the factor order. In the following we describe an alternative, quadratic algorithm.

Assume we are given the factor order $ZO = \langle V, \leq, \xi : V \to 2^{\mathcal{P}} \rangle$ of trace T (via the bijection μ) and the state information $\overline{\Delta} : V \to Q$ such that $\overline{\Delta}(v) = \Delta(q^0, T_v)$ for every $v \in V$. We choose some linearization $v_1, \ldots, v_n$ of B, and let $B_i = \{v_1, \ldots, v_i\}$. Let us write $B_{i,k}$ $(i \leq k)$ for the set $\{v_1, \ldots, v_i\} \cup \{v_j \mid v_j \leq v_k, j > i\}$. We show now how to compute inductively $\Delta(q^0, T_{B_{i,k}})$. The desired state is $\Delta(q^0, T_{B_{n,n}})$.

Suppose we know already $\Delta(q^0, T_{B_{i-1,k}})$, for all $k \geq i-1$. In particular, note that the states q^{i-1}, q^i reached on $\mu(v_1 \cdots v_{i-1})$ and $\mu(v_1 \cdots v_i)$, respectively, are known (cases $k = i-1$ and $k = i$).

We compute now $\Delta(q^0, T_{B_{i,k}})$, for $k > i$. Two cases arise. If $v_i \not< v_k$ then we apply Lemma 8 to $q^{i-1}, q^i, \Delta(q^0, T_{B_{i-1,k}}), \mathrm{dom}(v_i)$, which yields $\Delta(q^0, T_{B_{i,k}})$. If $v_i < v_k$, then $B_{i-1,k} = B_{i,k}$ and the state $\Delta(q^0, T_{B_{i,k}})$ is already known.

The proof of Proposition 2 is obtained by a repetitive use of Proposition 3. For the proof of Proposition 1 we additionally need the following.

Proposition 5. *Let $a \in \Sigma$ and $T = pref_{dom(a)}(T)$ be a trace. Given the zone order ZO and state information $\overline{\Delta}$ of T, we can compute the zone order ZO' and the state information $\overline{\Delta}'$ of $pref_{dom(a)}Ta$.*

Proof. We first add to ZO a new, maximal node w_a with $\xi(w_a) = \mathrm{dom}(a)$. It is not difficult to verify that we obtain a pre-zone order of $\mathrm{pref}_{\mathrm{dom}(a)}Ta$. By Lemma 6, the quotient of this order is the desired ZO'. By Lemma 9 we can calculate $\overline{\Delta}(B)$ for every downward closed subset B of ZO. Notice that $\bigcup\{[w]_\equiv \mid [w]_\equiv \leq' [v]_\equiv\}$ is downward closed in ZO. So we can compute $\overline{\Delta}'([v]_\equiv)$ for every $v \in V$.

For $\{w_a\}$ we first compute $\overline{\Delta}(V)$ by another application of Lemma 9. Then the state information for $\{w_a\}$ is $\Delta(\overline{\Delta}(V), a)$, namely the state reached by $\mathcal{A}$ on Ta. $\square$

The proof of Proposition 1 follows from Propositions 3 and 5.