

A quadratic construction for Zielonka automata with acyclic communication structure*

Siddharth Krishna

Chennai Mathematical Institute, Chennai, India

Anca Muscholl

LaBRI, Université Bordeaux, France

June 10, 2013

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 recognized by a deterministic asynchronous automaton. The construction is rather involved and the most efficient variant produces automata which are exponential in the number of processes and polynomial in the size of the DFA.

In this paper we show a simple, quadratic construction in the case where the synchronization actions are binary and define an acyclic communication graph.

Keywords: Distributed synthesis, asynchronous automata.

1 Introduction

Zielonka's asynchronous automata [20] is probably one of the simplest, and yet powerful, models of distributed computation. An asynchronous automaton is a parallel composition of finite-state processes synchronizing over shared (state) variables. Such an automaton is associated with a fixed distribution $dom : \Sigma \rightarrow (2^{\mathbb{P}} \setminus \{\emptyset\})$ of the alphabet of actions Σ on a (finite) set of processes \mathbb{P} . Informally, action a concerns only those processes that belong to $dom(a)$. This automaton model has a solid theoretical foundation based on the theory of Mazurkiewicz traces [12] (see [4] for a textbook). The key result in this area is that every regular trace language can be recognized by a *deterministic* asynchronous automaton (Zielonka's theorem, [20]).

The above mentioned result is one of the few examples of positive results on distributed synthesis, i.e., of the task of transforming a sequential specification into an equivalent, distributed implementation¹. Distributed synthesis is a major task in the design of correct

*Work partially supported by LIA INFORMEL.

¹Another result in this direction is the theory of regions for Petri nets, [5].

programs, given that concurrency is very challenging and often error-prone. It was actually the primary reason for introducing the branching-time temporal logic CTL².

Zielonka’s theorem has been used as black-box for synthesis of more complex automata models as well, e.g. communicating automata [14, 10, 8]. Its complex proof has been revisited on numerous occasions, see e.g. [2, 3, 15, 16, 9, 7]. The most recent construction [7] starts from a DFA and a distribution $dom : \Sigma \rightarrow (2^{\mathbb{P}} \setminus \{\emptyset\})$ and produces a deterministic asynchronous automaton of size polynomial in the size of the DFA and exponential in the number of processes. This construction is optimal w.r.t. so-called Zielonka-type constructions.

General constructions of Zielonka-type are rather complex since they are based on trace decompositions and time-stamping functions. The general idea is that the information that is computed by an asynchronous automaton in a decentralized way must be assembled together, and this step makes a crucial use of time-stamps (from a finite set). In some particular cases one can do better, since trace decompositions are simpler to handle. The constructions proposed in [13, 17] yield deterministic asynchronous (cellular) automata in the case where the dependence relation $D \subseteq \Sigma \times \Sigma$ induced by $dom : \Sigma \rightarrow (2^{\mathbb{P}} \setminus \{\emptyset\})$ is acyclic. A generalization to the case where D is chordal is presented in [3], see next section for definitions. These constructions are rather simple but still exponential in the size of a given DFA. In this note we present a simple, yet efficient construction for the case where synchronization actions are binary and the distribution $dom : \Sigma \rightarrow (2^{\mathbb{P}} \setminus \{\emptyset\})$ has an *acyclic* communication graph. We show that in this case we can construct a deterministic asynchronous automaton of *quadratic size*. We believe that this construction can have interesting applications, since hierarchical communication occurs naturally, for instance in systems that are layered. Furthermore, even if the communication graph may have cycles, certain distributed algorithms communicate over a pre-computed spanning tree (e.g., for broadcasting), so the communication is *de facto* acyclic.

Related work: A deadlock-free variant of Zielonka’s construction is proposed in [18], and an application to the synthesis of mutex algorithms is provided in [19, 18]. Alternative constructions for non-deterministic asynchronous automata are also known. In [21] an inductive construction for non-deterministic, deadlock-free asynchronous cellular automata is described. In [1] a construction for non-deterministic asynchronous automata of size exponential in the number of processes (and polynomial in the size of the DFA) is described. Note that while asynchronous automata can be determinized, there are cases where the blow-up is necessarily doubly exponential in the number of processes [11].

2 Basic definitions

A *distributed action alphabet* on a finite set \mathbb{P} of processes is a pair (Σ, dom) , where Σ is a finite set of *actions* and $dom : \Sigma \rightarrow (2^{\mathbb{P}} \setminus \{\emptyset\})$ is a *location function*. The location $dom(a)$ of action $a \in \Sigma$ comprises all processes that need to synchronize in order to perform this action.

We use the standard definition of a (sequential) deterministic finite automaton (DFA for short) over Σ as a tuple $\mathcal{A} = \langle S, \Sigma, \Delta, s^0, F \rangle$ consisting of a finite set of states S , a transition function $\Delta : S \times \Sigma \rightarrow S$, an initial state $s^0 \in S$ and a set of final states $F \subseteq S$. As usual we extend Δ to words in Σ^* . The language accepted by \mathcal{A} is denoted $L(\mathcal{A}) \subseteq \Sigma^*$, and the size $|\mathcal{A}|$ of \mathcal{A} is defined to be the number of states.

²Quoting from [6]: “We present a method of constructing concurrent programs in which the synchronization skeleton of the program is automatically synthesized from a (branching time) temporal logic specification.”

A (deterministic) *asynchronous automaton* $\mathcal{B} = \langle \{S_p\}_{p \in \mathbb{P}}, s_{in}, \{\delta_a\}_{a \in \Sigma}, F \rangle$ over (Σ, dom) is given by

- for every process p a finite set S_p of (local) states,
- the initial state $s_{in} \in \prod_{p \in \mathbb{P}} S_p$,
- for every action $a \in \Sigma$ a partial transition function $\delta_a : \prod_{p \in dom(a)} S_p \dashrightarrow \prod_{p \in dom(a)} S_p$ on tuples of states of processes in $dom(a)$,
- a set of global final states $F \subseteq \prod_{p \in \mathbb{P}} S_p$.

For a set $X \subseteq \mathbb{P}$ of processes and a (global) state $(s_p)_{p \in \mathbb{P}}$ we abbreviate the tuple $(s_p)_{p \in X}$ by s_X .

An asynchronous automaton can be seen as a sequential automaton with state set $S = \prod_{p \in \mathbb{P}} S_p$, transitions $s \xrightarrow{a} s'$ if $(s_{dom(a)}, s'_{dom(a)}) \in \delta_a$ and $s_{\mathbb{P} \setminus dom(a)} = s'_{\mathbb{P} \setminus dom(a)}$, and set of final states F . The language $L(\mathcal{B})$ is defined to be the language of the associated sequential automaton. The size of \mathcal{B} is defined here as $\sum_{p \in \mathbb{P}} |S_p|$. As we will consider only actions that are at most binary, the transition functions will be polynomial in the size defined above.

Example 2.1 Let $\mathbb{P} = \{p, q, r\}$ with $\Sigma = \{a, b, c, d\}$ and $dom(a) = \{p, q\}$, $dom(b) = \{q, r\}$, $dom(c) = \{p\}$, $dom(d) = \{r\}$. We consider an asynchronous automaton with $S_p = S_q = S_r = \{0, 1\}$ and $\delta_c(0) = 1$, $\delta_b(0, 0) = (1, 1)$, $\delta_a(1, 1) = (0, 0)$ and $\delta_d(1) = 0$ (in all other cases the transitions are undefined). Starting and ending in state $(0, 0, 0)$, the automaton accepts the set of all words with $\{a, b\}$ -projection from $(ba)^*$, $\{a, c\}$ -projection from $(ca)^*$ and $\{b, d\}$ -projection from $(bd)^*$.

The location mapping dom defines in a natural way an independence relation I : two actions $a, b \in \Sigma$ are independent (written as $(a, b) \in I$) if they involve different processes, that is, if $dom(a) \cap dom(b) = \emptyset$. Notice that the order of execution of two independent actions $(a, b) \in I$ in an asynchronous automaton is irrelevant, they can be executed as a, b , or b, a - or even concurrently. More generally, we can consider the congruence \sim_I on Σ^* generated by I , and observe that whenever $u \sim_I v$ the global state reached from the initial state on u and v , respectively, is the same. Hence, $u \in L(\mathcal{B})$ if and only if $v \in L(\mathcal{B})$. To simplify notation we write \sim instead of \sim_I when I is clear from the context. The complementary relation $D = (\Sigma \times \Sigma) \setminus I$ is called the *dependence relation*.

An equivalence class $T = [w]$ of \sim is called a Mazurkiewicz *trace* [12, 4]. As an example, Figure 1 shows the trace $[bcadbcbadb]$. The word $bcadbcbadb$ is a *linearization* of the trace represented in Figure 1. We have $(x, y) \in I$ iff $\{x, y\}$ is one of $\{a, d\}, \{b, c\}, \{c, d\}$. On that trace, the automaton in Example 2.1 reaches state s with $s_p = 0$ and $s_q = s_r = 1$. As we have observed, the language of an asynchronous automaton is a sum of such equivalence classes, in other words it is *trace-closed*.

For two traces $T_1 = [u], T_2 = [v]$, we define the *concatenation* $T_1 T_2$ as the trace $[uv]$. A trace T_1 is a *prefix* of a trace T if there exists another trace T_2 such that $T = T_1 T_2$. We write $view_p(T)$ to denote the minimal trace prefix of T that contains all occurrences of actions a with $p \in dom(a)$. More generally, $view_X(T)$ for a set of processes $X \subseteq \mathbb{P}$ is the minimal trace prefix of T that contains all occurrences of actions a with $dom(a) \cap X \neq \emptyset$. We also use the notation $dom(T)$ to denote all the processes that take part in actions in T . In Figure 1 we have $view_p([bcadbcbadb]) = [bcadbca]$ and $dom(bcadbcbadb) = \mathbb{P}$.

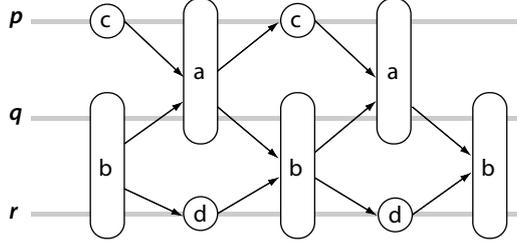


Figure 1: A trace with $\Sigma = \{a, b, c, d\}$, $\text{dom}(a) = \{p, q\}$, $\text{dom}(b) = \{q, r\}$, $\text{dom}(c) = \{p\}$, $\text{dom}(d) = \{r\}$.

A sequential automaton \mathcal{A} is called *I-diamond* if for all $(a, b) \in I$ and s state of \mathcal{A} we have $\Delta(s, ab) = \Delta(s, ba)$. For such an automaton we can define $\Delta(s, T)$ as the state $\Delta(s, u)$, where u is any word such that $T = [u]$. So the *I-diamond* property implies that a language is trace-closed. The sequential automaton obtained from an asynchronous automaton has the *I-diamond* property. There is a further, more interesting property of *I-diamond* automata, that we will repeatedly use in our construction:

Lemma 2.2 [2] *Given a deterministic I-diamond automaton $\mathcal{A} = \langle S, \Sigma, \Delta, s^0, F \rangle$, there is a function $\text{Diam} : S^3 \times 2^{\mathbb{P}} \rightarrow S$ such that for every three states $s_0, s_1, s_2 \in S$ and a set of processes $X \subseteq \mathbb{P}$, the state $s = \text{Diam}(s_0, s_1, s_2, X)$ satisfies the following property:*

For all traces T_0, T_1, T_2 with $\text{dom}(T_1) \subseteq X$ and $\text{dom}(T_2) \subseteq \mathbb{P} \setminus X$, if $\Delta(s^0, T_0) = s_0$, $\Delta(s^0, T_0 T_1) = s_1$ and $\Delta(s^0, T_0 T_2) = s_2$, then $\Delta(s^0, T_0 T_1 T_2) = s$.

For the proof of the lemma note that if T'_0, T'_1, T'_2 satisfy the same conditions as T_0, T_1, T_2 , then $\Delta(s^0, T'_0 T'_1 T'_2) = \Delta(s^0, T_0 T_1 T'_2) = \Delta(s^0, T_0 T_2 T'_1) = \Delta(s^0, T'_0 T'_2 T_1) = \Delta(s^0, T_0 T_2 T_1) = \Delta(T_0 T_1 T_2)$.

Finally, the restriction we have on the structure of the given distributed alphabet is defined on the following object.

Definition 2.3 *A distributed alphabet (Σ, dom) with unary and binary actions defines an undirected graph \mathcal{CG} with node set \mathbb{P} and edges $\{p, q\}$ if there exists $a \in \Sigma$ with $\text{dom}(a) = \{p, q\}$, $p \neq q$. Such a graph is called a communication graph.*

In this paper, we consider the case when the distributed alphabet defines an *acyclic* communication graph \mathcal{CG} . Without loss of generality, we will assume \mathcal{CG} to be a *rooted* tree. For a process q , let $\text{parent}(q)$ be the parent of q in \mathcal{CG} (for the root r , $\text{parent}(r)$ is undefined). We also define $\overleftarrow{\text{view}}_q(T)$ to be the smallest trace prefix of T containing all occurrences of actions a such that $\text{dom}(a) = \{q, \text{parent}(q)\}$, if q is different from the root. For the root r , let $\overleftarrow{\text{view}}_r(T)$ be the empty trace for all traces T . For instance, if p is the root process in Figure 1 then $\overleftarrow{\text{view}}_p([bcadcbdb]) = [bca]$, whereas $\overleftarrow{\text{view}}_q([bcadcbdb]) = [bcadb]$.

For $q \in \mathbb{P}$ we denote by $X_q \subseteq \mathbb{P}$ the set of processes in the subtree of q in \mathcal{CG} .

3 The construction

We are given a DFA \mathcal{A} accepting a trace-closed language $L \subseteq \Sigma^*$, over a distributed alphabet with only unary and binary actions such that the communication graph \mathcal{CG} is acyclic. We want to construct a deterministic asynchronous automaton that recognizes L .

Let $\mathcal{A} = \langle S, \Sigma, \Delta, s^0, F \rangle$ be the minimal DFA of L . It can be easily checked that \mathcal{A} has the I -diamond property.

We construct the asynchronous automaton $\mathcal{B} = \langle \{S_p\}_{p \in \mathbb{P}}, s_{in}, \{\delta_a\}_{a \in \Sigma} \rangle$ for L as follows:

- For every process $p \in \mathbb{P}$, let $S \times S$ be the set of local states. We write in the following $\langle s, s' \rangle_p$ to denote a local state of process p .
- The start state of process p is $(s_{in})_p = \langle s^0, s^0 \rangle_p$.
- The transition function δ_a for each action $a \in \Sigma$ is defined as follows:
 - if $\text{dom}(a) = \{p\}$ then $\delta_a(\langle s, s' \rangle_p) = \langle s, \Delta(s', a) \rangle_p$.
 - if $\text{dom}(a) = \{p, q\}$ and p is the parent of q in \mathcal{CG} , then

$$\delta_a(\langle s_1, s'_1 \rangle_p, \langle s_2, s'_2 \rangle_q) = (\langle s_1, s' \rangle_p, \langle s', s'_2 \rangle_q),$$

where $s = \text{Diam}(s_2, s'_2, s'_1, X_q)$ and $s' = \Delta(s, a)$.

Note that we only use the values of $\text{Diam}(*, *, *, X)$ for linearly many $X \subseteq \mathbb{P}$, making the precomputation polynomial in the size of the input.

The basic idea is that each process p simulates the automaton \mathcal{A} on the part of the input T it has seen ($\text{view}_p(T)$), and stores this information in the second co-ordinate of its state. But to combine this information from each process and its parent and get the state reached by \mathcal{A} on the entire trace T , we need to store some extra information. This is the state reached by \mathcal{A} on the part of the trace until process p last synchronized with its parent ($\overleftarrow{\text{view}}_p(T)$). We then use Lemma 2.2 to compute the state reached by \mathcal{A} on all of T (see Propositions 4.1 and 4.2 below).

The accepting condition requires additional notation, and is defined in the next section. With this construction, we obtain the following theorem, the main result of this paper:

Theorem 3.1 *Let (Σ, dom) be a distributed alphabet with unary and binary actions, whose communication graph is acyclic. Then every regular, trace-closed language L over Σ can be recognized by a deterministic asynchronous automaton. The number of states is quadratic in the number of states of the minimum DFA for L .*

4 Correctness of the construction

We prove the equivalence of \mathcal{A} and \mathcal{B} with the help of the following proposition:

Proposition 4.1 *After the execution of a trace T by the automaton \mathcal{B} , each process q is in a state $\langle s_q, s'_q \rangle$ such that $s_q = \Delta(s^0, \overleftarrow{\text{view}}_q(T))$ and $s'_q = \Delta(s^0, \text{view}_q(T))$.*

Proof. We show this by induction on the length of the trace T , and notice that the base case is trivially true. For the inductive step, assume the statement holds for a trace T and let $T' = Ta$. Let $\langle s_q, s'_q \rangle$ be the state of process $q \in \mathbb{P}$ after execution of T by \mathcal{B} . We have the following cases:

Then we simply define $\langle s_p, s'_p \rangle_{p \in \mathbb{P}}$ to be a final state of \mathcal{B} if and only if $\mathbf{state}(r) \in F$, where r is the root process.

```

input : A process  $p$ 
output: the state  $\Delta(s^0, view_{X_p}(T))$ 
1  $s \leftarrow s'_p$ ;
2 foreach child  $q$  of  $p$  do
3    $s \leftarrow \text{Diam}(s_q, \mathbf{state}(q), s, X_q)$ ;
4 end
5 return  $s$ 

```

Function $\mathbf{state}(p)$

We next prove the correctness of the function $\mathbf{state}(p)$:

Proposition 4.2 *The function $\mathbf{state}(p)$ returns the state $\Delta(s^0, view_{X_p}(T))$.*

Proof. Let q_1, \dots, q_k be the children of p . We prove by induction on i , that $s = \Delta(s^0, view_{\{p\} \cup X_{q_1} \cup \dots \cup X_{q_i}}(T))$ after the i th round of the **foreach** loop. Before the first iteration, we have $i = 0$ and the result follows simply from Proposition 4.1. For the induction step, factorize

$$view_{\{p\} \cup X_{q_1} \cup \dots \cup X_{q_i}}(T) = T_0 T_1 T_2$$

where

$$T_0 = \overleftarrow{view}_{q_i}(T), \quad T_0 T_1 = view_{X_{q_i}}(T), \quad T_0 T_2 = view_{\{p\} \cup X_{q_1} \cup \dots \cup X_{q_{i-1}}}(T)$$

such that $dom(T_1) \cap dom(T_2) = \emptyset$. We have (by Proposition 4.1) that

$$\Delta(s^0, T_0) = \Delta(s^0, \overleftarrow{view}_{q_i}(T)) = s_{q_i}.$$

Moreover, recursively assuming correctness of $\mathbf{state}(q_i)$ for the children, we have

$$\Delta(s^0, T_0 T_1) = \Delta(s^0, view_{X_{q_i}}(T)) = \mathbf{state}(q_i).$$

By the induction hypothesis, the value of s before the i th round is

$$\Delta(s^0, T_0 T_2) = \Delta(s^0, view_{\{p\} \cup X_{q_1} \cup \dots \cup X_{q_{i-1}}}(T)).$$

Thus, using Lemma 2.2 we know that after the i th round

$$s = \text{Diam}(s_{q_i}, \mathbf{state}(q_i), s, X_{q_i}) = \Delta(s^0, T_0 T_1 T_2) = \Delta(s^0, view_{\{p\} \cup X_{q_1} \cup \dots \cup X_{q_i}}(T)).$$

Since $view_{X_p}(T) = view_{\{p\} \cup X_{q_1} \cup \dots \cup X_{q_k}}(T)$, we get the required result. □

Finally, since $X_r = \mathbb{P}$, we have $view_{X_r}(T) = T$ and $\mathbf{state}(r) = \Delta(s^0, T)$, thus $L(\mathcal{A}) = L(\mathcal{B})$.

5 Conclusion

We showed a simple and efficient construction for deterministic asynchronous automata in the case where shared actions are binary and induce an acyclic synchronization relation between processes. This restriction seems very natural, since many systems use process hierarchy (or spanning trees) for communication. As shown by [7] it is very unlikely that the general case can be simplified, due to the exponential lower bound in the number of processes. Whether other natural restrictions on the communication graph can lead to similar simple constructions as presented here, remains open.

Acknowledgment: We thank the referees for their constructive and detailed suggestions for improving the presentation. The second author thanks M. Bojańczyk for the question leading to this result.

References

- [1] N. Baudru. Distributed Asynchronous Automata. In *CONCUR'09*, LNCS 5710, 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 Diekert and Rozenberg [4], pages 249–267.
- [4] V. Diekert and G. Rozenberg, editors. *The Book of Traces*. World Scientific, Singapore, 1995.
- [5] A. Ehrenfeucht and G. Rozenberg. Partial (set) 2-structures: Parts i and ii. *Acta Informatica*, 27(4):315–368, 1989.
- [6] E. A. Emerson and E. M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Sci. Comput. Program.*, 2(3):241–266, 1982.
- [7] B. Genest, H. Gimbert, A. Muscholl, and I. Walukiewicz. Optimal Zielonka-type construction of deterministic asynchronous automata. In *Proceedings ICALP'10*, volume 6199 of *LNCS*. Springer, 2010.
- [8] 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.
- [9] B. Genest and A. Muscholl. Constructing Exponential-Size Deterministic Zielonka Automata. In *ICALP'06*, LNCS 4052, pages 565–576. Springer, 2006.
- [10] 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.
- [11] N. Klarlund, M. Mukund, and M. Sohoni. Determinizing Asynchronous Automata. In *ICALP'94*, LNCS 820, pages 130–141. Springer, 1994.
- [12] A. Mazurkiewicz. Concurrent program schemes and their interpretations. DAIMI Rep. PB 78, Aarhus University, Aarhus, 1977.

- [13] Y. Métivier. An algorithm for computing asynchronous automata in the case of acyclic non-commutation graph. In *ICALP'87*, LNCS 267, pages 226–236. Springer, 1987.
- [14] M. Mukund, K. N. Kumar, and M. Sohoni. Synthesizing distributed finite-state systems from MSCs. In *CONCUR'00*, LNCS 1877, pages 521–535. Springer, 2000.
- [15] 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.
- [16] M. Mukund and M. A. Sohoni. Keeping Track of the Latest Gossip in a Distributed System. *Distributed Computing*, 10(3):137–148, 1997.
- [17] D. Perrin. Partial commutations. In *Proc. ICALP*, number 372 in LNCS, pages 637–651. Springer, 1989.
- [18] A. Stefanescu. *Automatic synthesis of distributed transition systems*. PhD thesis, Universität Stuttgart, 2006.
- [19] A. Stefanescu, J. Esparza, and A. Muscholl. Synthesis of distributed algorithms using asynchronous automata. In *CONCUR*, number 2761 in LNCS, pages 27–41, 2003.
- [20] W. Zielonka. Notes on finite asynchronous automata. *RAIRO—Theoretical Informatics and Applications*, 21:99–135, 1987.
- [21] W. Zielonka. Safe executions of recognizable trace languages by asynchronous automata. In A. R. Meyer et al., editors, *Symposium on Logical Foundations of Computer Science (Logic at Botik'89)*, number 363 in Lecture Notes in Computer Science, pages 278–289. Springer Verlag, 1989.