Implementing Realistic Asynchronous Automata

Sundararaman Akshay, Ionut Dinca, Blaise Genest, Alin Stefanescu

Implementing Realistic Asynchronous Automata

S. Akshay[1], Ionut Dinca[2], Blaise Genest[3], and Alin Stefanescu[2]

1 Indian Institute of Technology, Bombay, India
2 University of Pitesti, Romania
3 CNRS, IRISA, Rennes, France

Abstract

Zielonka's theorem, established 25 years ago, states that any regular language closed under commutation is the language of an *asynchronous automaton* (a tuple of automata, one per process, exchanging information when performing common actions). Since then, constructing asynchronous automata has been simplified and improved [6, 20, 7, 12, 8, 4, 2, 21, 22].

We first survey these constructions and conclude that the synthesized systems are not *realistic* in the following sense: existing constructions are either plagued by deadends, non deterministic guesses, or the acceptance condition or choice of actions are not distributed. We tackle this problem by giving (effectively testable) necessary and sufficient conditions which ensure that deadends can be avoided, acceptance condition and choices of action can be distributed, and determinism can be maintained. Finally, we implement our constructions, giving promising results when compared with the few other existing prototypes synthesizing asynchronous automata.

1998 ACM Subject Classification F.1.1 Models of Computation, F.4.3 Formal Languages

Keywords and phrases Asynchronous automata, Zielonka construction, Implementability

Digital Object Identifier 10.4230/LIPIcs.xxx.yyy.p

1 Introduction

Designing distributed systems is notoriously difficult and prone to bugs. Verification algorithms are very useful to detect and report bugs, but the discovered issues must be solved by the designer. An alternative is to use automatic implementation tools, which directly *synthesize* an implementation that is guaranteed to be correct by construction. As the complexity of automatic implementation is quite high in the general case of *open* distributed systems (distributed games) [9], we focus on closed systems in this paper.

Here, the specification is given as a regular language L over an alphabet Σ where every action (i.e., letter in Σ) is associated with the set of processes managing that action. Such

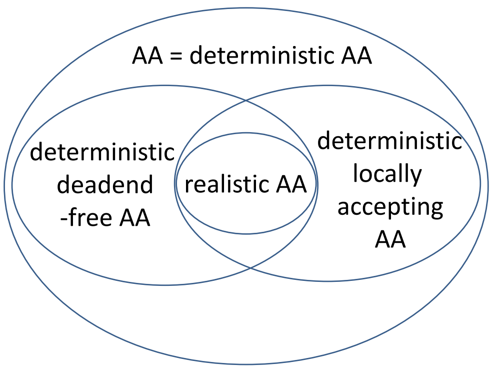

Figure 1 Expressivity of different types of asynchronous automata (AA)

Conference title on which this volume is based on.
Editors: Anil Seth, Nisheeth Vishnoi - Bill Editors; pp. 1–16
Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

a specification allows to reason globally about the requirements, instead of having to deal carefully with partial views of each process in a distributed manner (which is one of the error-prone tasks). The problem is then to automatically implement a (truly) distributed control that will globally have the same behavior as the given specification language L. Of course, not all languages can be implemented with such a distributed control. For instance, if ab is the only word in the specification language, with a an action local to a process and b local to another process, then it cannot be implemented in a truly distributed manner. Indeed, any distributed implementation will also feature the word ba, since a process is unable to know when another process performs an (independent) action.

Zielonka's theorem, established 25 years ago [23], states that this is sufficient: every language closed by this commutation relation can be implemented in the form of an *asynchronous automaton*, that is, a network of automata where the control is *mostly* distributed, and two processes can exchange information whenever they perform a common action. Initially, this was merely an expressiveness result and was believed to be rather impractical due to its prohibitive complexity. During subsequent years, this construction has been simplified and improved in several works [6, 20, 7, 12, 8]. Also, different constructions [4, 2, 21] and heuristics [22] have been proposed to handle the complexity blow-up.

However, none of these constructions gives a general *realistic* distributed implementation: either the constructions are plagued by deadends [23, 6, 20, 7, 12, 8, 2, 4], non-deterministic guesses [24, 5, 2], or the acceptance condition or choice of actions are not distributed [23, 6, 20, 7, 12, 8]. Further, while the initial state is trivially distributed in Zielonka's construction (since it is unique, due to determinism), this is not the case in [24, 5]. One cannot always obtain an implementation satisfying all these conditions: we schematically depict in Figure 1 the relations between corresponding subclasses, proved in Proposition 6.

Thus, our main goal is to characterize the class of regular languages that can be implemented by a *realistic asynchronous automaton*, i.e., one which is deterministic, deadend-free, and has distributed final states and choice of actions. This notion strictly subsumes the class of deadend-free synchronized product of automata [17]. Our central result provides semantical and syntactical characterizations of languages of realistic asynchronous automata, together with algorithms to check these characterizations. Thus, given a global regular specification passing these algorithmic tests, we build a realistic asynchronous automaton which distributedly implements the specification. Finally, we implement our procedure, based on the latest, state of the art variant of Zielonka's construction [8]. On a variety of distributed programs, we show that this gives realistic distributed implementations of a size which is reasonable compared to existing implementations.

Asynchronous automata model shared-memory systems directly. However, even for message passing systems, Zielonka's theorems continue to remain interesting: [15] and [10] build bounded message passing automata using Zielonka's construction (see [3] for a survey). We are confident that combining the techniques in [15] with our results would lead to the automatic implementation of realistic bounded message passing automata.

The paper is structured as follows. In section 2, we define (realistic) asynchronous automata, and restate the different implementation theorems. In section 3, we come up with semantical and syntactical characterizations of realistic asynchronous automata. In section 4, we exhibit algorithms to test the characterizations and analyze their complexity. In section 5, we experiment and compare the automatic distributed implementation of different specifications.

Conference title on which this volume is based on.
Editors: Anil Seth, Nisheeth Vishnoi - Bill Editors; pp. 2–16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

2 Realistic Asynchronous Automata

Let $\mathcal{P}$ be a fixed set of processes. A distributed alphabet (Σ, dom) is a finite set Σ of actions together with the domain function $\mathrm{dom} : \Sigma \to 2^{\mathcal{P}} \setminus \emptyset$, which associates to each action a the set $\mathrm{dom}(a)$ of processes executing a. For any $p \in \mathcal{P}$, we also denote $\Sigma_p = \{a \in \Sigma \mid p \in \mathrm{dom}(a)\}$. We say that actions a and b are *independent*, denoted $(a, b) \in I$, iff $\mathrm{dom}(a) \cap \mathrm{dom}(b) = \emptyset$. This gives rise to an equivalence relation on words: first, for all words $v, w \in \Sigma^*$ and actions $(a, b) \in I$, we define $vabw \equiv_1 vbaw$. Then, the transitive reflexive closure of $\equiv_1$, denoted $\equiv$, is an equivalence relation. The equivalence class containing v, denoted $[v]$, is called a (Mazurkiewicz) *trace* [7]. Given a word $w \in \Sigma^*$ and a process $p \in \mathcal{P}$, the *p-view* of w, denoted $\mathrm{view}_p(w)$, is the shortest trace $[v]$ such that: there exists v' with $w \equiv vv'$, and each action $a \in \Sigma_p$ occurs as many times in v as in w. Finally, for a language $L \subseteq \Sigma^*$, $\mathrm{pref}(L)$ will denote its set of prefixes and ϵ will denote the empty string.

An *asynchronous automaton* is a tuple $((S_p)_{p \in \mathcal{P}}, (\Delta_a)_{a \in \Sigma}, In, Fin)$, where for all $p \in \mathcal{P}$, S_p is the set of *local states of process* p, and for all $a \in \Sigma$, $\Delta_a \subseteq \prod_{p \in \mathrm{dom}(a)} S_p \times \prod_{p \in \mathrm{dom}(a)} S_p$ defines the (partial) *transition relation*. Note that while we define the transition relation on letters for ease of presentation, it is equivalent to a corresponding definition on processes. Any $s = (s_p)_{p \in \mathcal{P}} \in \prod_{p \in \mathcal{P}} S_p$ is called a *global state* and $In, Fin \subseteq (S_p)_{p \in \mathcal{P}}$ denote the set of global initial and final states, respectively.

The *semantics* of an asynchronous automaton $AA = ((S_p)_{p \in \mathcal{P}}, (\Delta_a)_{a \in \Sigma}, In, Fin)$ is given by the (sequential) automaton $S(AA) = (C, \to, In, Fin)$ over Σ, where $C = \prod_{p \in \mathcal{P}} S_p$ is the set of global states, and the global transition relation is given by $\to$: $C \to C$ with $(s_p)_{p \in \mathcal{P}} \xrightarrow{a} (s'_p)_{p \in \mathcal{P}}$ iff $(s'_p)_{p \in \mathrm{dom}(a)} \in \Delta_a((s_p)_{p \in \mathrm{dom}(a)})$ and $s'_p = s_p$ for all $p \notin \mathrm{dom}(a)$. As usual, we extend $\to$ to words by fixing for ϵ the empty word: for all $s, s' \in C$, $s \xrightarrow{\epsilon} s'$ iff $s' = s$ and $s \xrightarrow{aw} s'$ iff there exists $s'' \in C$ with $s \xrightarrow{a} s''$ and $s'' \xrightarrow{w} s'$. In case $\to$ is deterministic (which is the case for deterministic asynchronous automata), we will denote $\delta_w(s)$ for the unique state $s' \in C$ (if it exists) such that $s \xrightarrow{w} s'$. The language $\mathcal{L}(AA)$ accepted by AA is by definition $\mathcal{L}(S(AA))$, the language accepted by $S(AA)$.

An automaton $A = (C, \to, In, Fin)$ is *diamond* [7] if for all $s, s', t \in C$ and all $(a, b) \in I$, if $s \xrightarrow{a} s' \xrightarrow{b} t$, then there exists t' with $s \xrightarrow{b} t' \xrightarrow{a} t$. For any given asynchronous automaton AA, $S(AA)$ is *diamond* [7], which implies that $\mathcal{L}(AA)$ is *closed by commutation*: for all $v \in \mathcal{L}(AA)$ and $w \equiv v$, we also have $w \in \mathcal{L}(AA)$.

An asynchronous automaton, as defined above, cannot always be implemented in a distributed manner, without adding further restrictions. For instance, the set of final states is currently given globally. To obtain purely distributed implementations, we now introduce several restrictions on asynchronous automata.

▶ **Definition 1** (determinism). An asynchronous automaton $AA = ((S_p)_{p \in \mathcal{P}}, (\Delta_a)_{a \in \Sigma}, In, Fin)$ is *deterministic*, if $|In| = 1$ and $|\Delta_a(s)| \leq 1$ for all $a \in \Sigma$ and $s \in \prod_{p \in \mathrm{dom}(a)} S_p$.

Non-determinism allows a process to guess what another process is doing concurrently. Note that every asynchronous automaton can be transformed into a deterministic asynchronous automaton, albeit with an unavoidable blow-up in the number of states [14].

▶ **Definition 2** (deadend-freeness). A global state s is called a *deadend*, if there does not exist a word $w \in \Sigma^*$ and global state $s' \in Fin$ with $s \xrightarrow{w} s'$. An asynchronous automaton is *deadend-free* iff no global state reachable from an initial state is a deadend: for all $v \in \Sigma^*, s_0 \in In$, and all s with $s_0 \xrightarrow{v} s$, the state s is not a deadend.

Deadend-freeness prevents a process from performing actions that will not be observable in terms of the language. For instance, consider two processes p, q and actions a, b, c such that

Conference title on which this volume is based on.
Editors: Anil Seth, Nisheeth Vishnoi - Bill Editors; pp. 3–16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

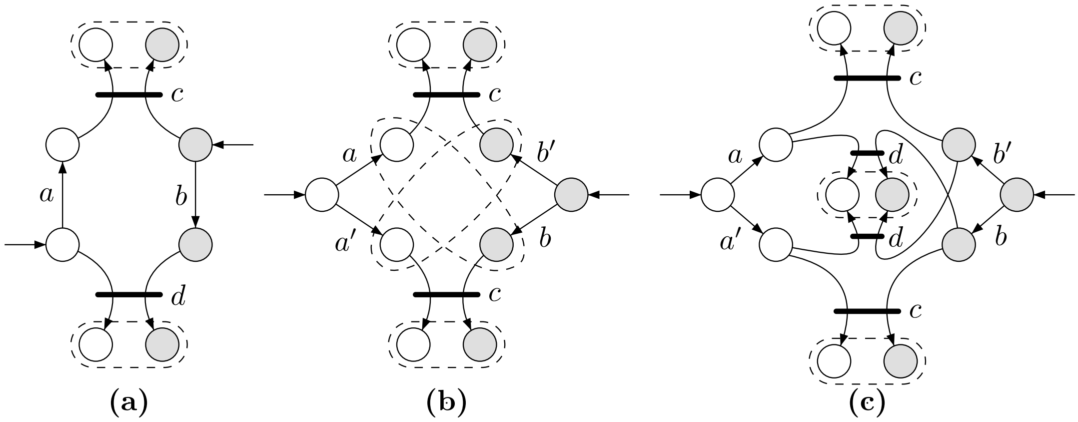

Figure 2 Examples of “unrealistic” asynchronous automata accepting respectively L_1, L_2, L_3. States of process p (resp. q) are unshaded (resp. shaded). Dashed lines mark global final states.

$\mathrm{dom}(a) = p$, $\mathrm{dom}(b) = q$ and $\mathrm{dom}(c) = \mathrm{dom}(d) = \{p, q\}$. Then, the language $L_1 = \{ac, bd\}$ cannot be implemented deterministically and without deadends. Indeed, both a and b are allowed from the initial state (which is unique, if the implementation is deterministic), and thus any realistic implementation would also allow ab (and ba), as $\mathrm{dom}(a) \cap \mathrm{dom}(b) = \{p\} \cap \{q\} = \emptyset$. However, an asynchronous automaton with deadends can implement this language as shown in Figure 2(a): the state reached after reading the trace $[ab]$ is a deadend.

▶ **Definition 3** (local acceptance). An asynchronous automaton $((S_p)_{p\in\mathcal{P}}, (\Delta_a)_{a\in\Sigma}, \mathit{In}, \mathit{Fin})$ is said to be *locally accepting* or *have local final states*, if $\mathit{Fin} = \prod_{p\in\mathcal{P}} \mathit{Fin}_p$ for some $\mathit{Fin}_p \subseteq S_p$ for all $p \in \mathcal{P}$.

Local final states ensure that processes can stop locally, and there is no supervisor which looks at all processes at the same time to choose to stop them. Note that the asynchronous automaton in Figure 2(a) has local final states. Now, the language $L_2 = \{ab, ba, a'b', b'a', a'bc, ba'c, ab'c, b'ac\}$ with $\mathrm{dom}(a) = \mathrm{dom}(a') = p$, $\mathrm{dom}(b) = \mathrm{dom}(b') = q$, and $\mathrm{dom}(c) = \{p, q\}$ cannot be accepted by a deterministic asynchronous automaton having local final states as local states reached on p after a, a' and local states reached on q after b, b' can all be final, depending what the other process did. However, there is a deadend-free deterministic asynchronous automaton with global final states accepting this language, as shown in Figure 2(b). Here, the global final states reached after reading $[ab]$ and $[a'b']$ cannot be expressed as a product of local final states (without also accepting $[ab'], [a'b]$).

▶ **Definition 4** (locally enabled). An asynchronous automaton $((S_p)_{p\in\mathcal{P}}, \Delta, \mathit{In}, \mathit{Fin})$ is called *locally enabled*, if for all reachable global states $s = (s_p)_{p\in\mathcal{P}}$, $s' = (s'_p)_{p\in\mathcal{P}}$, and $s'' = (s''_p)_{p\in\mathcal{P}}$, if there exist $a \in \Sigma$ and global states t, t' with $s''_p \in \{s_p, s'_p\}$ for all $p \in \mathrm{dom}(a)$ and $s \xrightarrow{a} t$ and $s' \xrightarrow{a} t'$, then there exists a global state t'' with $s'' \xrightarrow{a} t''$.

Local enabledness prevents the processes from taking into account the state of other processes to decide whether they should propose an action or not. In terms of distributed control, process based controllers [9] have this property, while action based controllers [11] do not. The asynchronous automata in Figure 2(a,b) are locally enabled. In a distributed implementation, non-local enabledness is not realistic. For instance, consider the language $L_3 = \{abd, bad, a'bc, ba'c, ab'c, b'ac, a'b'd, b'a'd\}$, with $\mathrm{dom}(a) = \mathrm{dom}(a') = p$, $\mathrm{dom}(b) = \mathrm{dom}(b') = q$ and $\mathrm{dom}(c) = \mathrm{dom}(d) = \{p, q\}$. Intuitively, processes p, q should synchronize

Conference title on which this volume is based on.
Editors: Anil Seth, Nisheeth Vishnoi - Bill Editors; pp. 4–16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

with d if they did both a, b or a', b', and with c if they did a, b' or a', b. This language cannot be realized by a *deadend-free and locally enabled* asynchronous automaton. However, the deadend-free and locally accepting asynchronous automaton shown in Figure 2(c) accepts L_3, but it is not locally-enabled.

Ideally, we would like a realistic distributed implementation to satisfy *all* the properties of determinism, deadend-freeness, local acceptance and local enabledness and not just *some* of them. Thus, by combining all the above desired properties of a distributed implementation we arrive at our proposal for a realistic asynchronous automaton.

▶ **Definition 5.** An asynchronous automaton AA is said to be *realistic*, if AA is deterministic, deadend-free, has local final states, and is locally enabled.

With this definition, language L_3 above cannot be accepted by a realistic asynchronous automaton (because of local enabledness). Using languages L_1, L_2, L_3, we conclude:

▶ **Proposition** 6. The inclusions schematically represented in Figure 1, between the expressive powers of the above introduced restrictions of asynchronous automata, are strict. Further, the classes of deterministic deadend-free and deterministic locally accepting asynchronous automata have incomparable expressive power.

Proof. First, the fact that asynchronous automata and deterministic asynchronous automata have the same expressive power is proved in Theorem 8(1).

Second, $L_1 = \{ac, bd\}$, with $\mathrm{dom}(a) = p$, $\mathrm{dom}(b) = q$ and $\mathrm{dom}(c) = \mathrm{dom}(d) = \{p, q\}$, is accepted by a deterministic locally accepting asynchronous automaton (see Figure 2) but not by any deterministic deadend-free asynchronous automata. Indeed, by contradiction, if $L(AA) = L_1$ with AA deterministic and deadend-free, then denoting by (s_0^p, s_0^q) the initial state of AA, we have $\Delta_a(s_0^p) = s$ and $\Delta_b(s_0^q) = t$ for some local states s of p and t of q. Thus (s, t) is a global state reachable by ab. As AA is deadend-free, there exists w with $(s, t) \xrightarrow{w} f$ with f accepting. It means that $abw \in L_1$, a contradiction.

Third, L_2 is accepted by a deterministic deadend-free asynchronous automaton (see Figure 2) but not by any deterministic locally accepting asynchronous automata. Indeed, by contradiction, if $L(AA) = L_2$ with AA deterministic and locally accepting, then denoting by (s_0^p, s_0^q) the initial state of AA, we have $\Delta_a(s_0^p) = s$, $\Delta_{a'}(s_0^p) = s'$, $\Delta_b(s_0^q) = t$ and $\Delta_{b'}(s_0^q) = t'$. Now, $ab, a'b' \in L_2$ hence (s, t) and (s', t') are final. It means that s, s', t, t' are local final states as AA is locally accepting, and thus (s, t') and (s', t) are final. Thus $ab' \in L_2$, a contradiction.

Last, the expressive power of realistic asynchronous automata is strictly included into the expressive power of deterministic deadend-free and locally accepting asynchronous automata. Indeed, L_3 is accepted by a deterministic deadend-free and locally accepting asynchronous automata (see Figure 2). Now, assume by contradiction that there exists a deterministic deadend-free locally enabeled asynchronous automaton AA accepting L_3. Denoting by (s_0^p, s_0^q) the initial state of AA, we have $\Delta_a(s_0^p) = s$, $\Delta_{a'}(s_0^p) = s'$, $\Delta_b(s_0^q) = t$ and $\Delta_{b'}(s_0^q) = t'$. Consider the global states (s, t);(s', t');(s, t'). We have $(s, t) \xrightarrow{d} (s_2, t_2)$ and $(s', t') \xrightarrow{d} (s_2', t_2')$ for some s_2, s_2', t_2, t_2'. Hence there exists a global state (s_2'', t_2'') such that $(s, t') \xrightarrow{d} (s_2'', t_2'')$. As AA is deadend-free, there exists a global final state r and a word w such that $(s_2'', t_2'') \xrightarrow{w} r$. Thus $ab'dw \in L_3$, a contradiction. ☐

We remark here that the notion of realistic automata as defined above strictly subsumes the notion of (deadend-free) synchronized product of automata [17]. Such an automaton is given by a tuple of automata $A = (A_p)_{p \in \mathcal{P}}$, one for each process p on alphabet $\Sigma_p = \Sigma \cap \mathrm{dom}^{-1}(p)$, such that $u \in L(A)$ iff $\pi_p(u) \in L(A_p)$ for all $p \in \mathcal{P}$, where $\pi_p(u)$ is the projection of u on Σ_p, that is u where actions not in Σ_p have been deleted.

Conference title on which this volume is based on.
Editors: Anil Seth, Nisheeth Vishnoi - Bill Editors; pp. 5–16
LIPICS Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

▶ **Proposition 7.** Let $A = (A_p)_{p\in\mathcal{P}}$ be a (possibly non-deterministic) deadend-free synchronized product of automata. Then there exists a realistic asynchronous automaton B with $L(B) = L(A)$. However, the converse does not hold.

Proof. First, one can determinize A without changing the language and keeping the deadend-freeness. This follows since each local automaton can be determinized keeping the same language, and because of the characterization $u \in L(A)$ iff $\pi_p(u) \in L(A_p)$. The resulting automaton can be seen as a deterministic deadend-free asynchronous automaton B. By definition of the synchronized product of automata, B is also locally accepting and locally enabled, that is, B is a realistic asynchronous automaton, and we have $L(B) = L(A)$.

To see that the converse does not hold, consider the following language: $L_5 = \{acb, a'cb'\}$ with $\mathrm{dom}(a) = \mathrm{dom}(a') = p, \mathrm{dom}(b) = \mathrm{dom}(b') = q, \mathrm{dom}(c) = \{p, q\}$. This language is easily realizable by a realistic asynchronous automaton. To see that it cannot be realized by the synchronized product of automata, let us argue by contradiction: if L_4 was the language of a synchronized product of automata $A = (A_p)_{p\in\mathcal{P}}$, then we would have $ac, a'c \in L(A_p)$ and $cb, cb' \in L(A_q)$, and hence we would have $acb' \in L(A)$, a contradiction. ☐

2.1 Survey of the different constructions

In the past 25 years, several attempts have been made to construct asynchronous automata from regular (commutation-closed) specifications which preserve *some* (but not all) of these above mentioned properties. We summarize them below.

▶ **Theorem 8.** *Let L be a regular language closed by commutation. Then, there exists an asynchronous automaton AA over (Σ, dom) with $\mathcal{L}(AA) = L$ such that either:*

1. *AA is deterministic [23, 6, 7, 20, 12, 8], or*
2. *AA is deadend-free [24] (see also [5] for a proof for message-passing systems), or*
3. *AA has local initial and final states [2].*

We provide here the worst case space complexities (the number of local states) to obtain a deterministic or non deterministic asynchronous automaton (Det AA, Non Det AA), given a deterministic or non deterministic diamond automaton A over a set of processes $\mathcal{P}$:

complexity	Det AA	Non Det AA
Det A	$\lvert A\rvert^{O(\lvert\mathcal{P}\rvert^2)} \cdot 2^{2\lvert\mathcal{P}\rvert^4}$ [8]	–
Non Det A	$2^{O(\lvert A\rvert\cdot\lvert\mathcal{P}\rvert^2+\lvert\mathcal{P}\rvert^4)}$ [12]	$\lvert A\rvert^{O(\lvert\mathcal{P}\rvert^2)}$ [2]

The complexities stated to obtain a deterministic asynchronous automaton from [12, 8] are optimal, while optimality is not proven for obtaining a non deterministic asynchronous automaton (using [2] for instance). Note that [2] uses a construction not based on Zielonka's. Determinizing an asynchronous automaton is possible, but the blow-up is doubly exponential [14]: constructing a deterministic asynchronous automaton directly is preferable. Notice that the complexity is much better in case where the architecture is constrained to be a tree, see [18].

3 Obtaining Realistic Asynchronous Automata

We now turn to the question of characterizing regular languages L for which there exists a *realistic* asynchronous automaton AA such that $\mathcal{L}(AA) = L$. We will give necessary and sufficient semantical conditions on L to have a realistic distributed implementation AA accepting L. Further, we will provide syntactical conditions on automata to be equivalent

Conference title on which this volume is based on.
Editors: Anil Seth, Nisheeth Vishnoi - Bill Editors; pp. 6–16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

to realistic distributed implementations, and prove that a diamond automaton with such conditions is always constructible. Our proofs are constructive, in that they provide realistic asynchronous automata. We also offer characterizations for subsets of realistic properties.

Our main proof can use any of the variants of the Zielonka construction from [23, 6, 7, 20, 12, 8] as a "black box", without having to reprove them. Moreover, the changes we make to the implementation obtained from the Zielonka construction do not add states.

3.1 A Theoretical Characterization of Realistic AA

Before stating the main theoretical result of the paper, we first define the syntactical and semantical restrictions which will enable realistic asynchronous automata. Recall that we defined the notion of $\mathrm{view}_p(u)$ in Section 2, which stands for all actions of u that p has seen directly or indirectly (through a common action). For instance, let $\mathrm{dom}(a) = p, \mathrm{dom}(b) = q$ and $\mathrm{dom}(c) = \{p, q\}$. Then $\mathrm{view}_p(abcb) \equiv abc$ since c is "seen" by p ($p \in \mathrm{dom}(c)$) and b is "before" c, b and c are not independent as $\mathrm{dom}(b) \cap \mathrm{dom}(c) = \{q\} \neq \emptyset$.

▶ **Definition 9** (Semantical conditions)**.** For language L, we define the following conditions:

(LC1) *forward diamond*: Whenever $w \in \Sigma^*, (a, b) \in I$ and $wa, wb \in \mathrm{pref}(L)$, we have $wab \in \mathrm{pref}(L)$.

(LC2) *causally closed*: Whenever $w \in \Sigma^*$, if for all $p \in \mathcal{P}$ there exists $v_p \in L$ with $\mathrm{view}_p(v_p) = \mathrm{view}_p(w)$, then $w \in L$.

(LC3) *locally closed*: Whenever $w \in \mathrm{pref}(L)$, if for all actions c and all $p \in \mathrm{dom}(c)$, there exists $v_p c \in \mathrm{pref}(L)$ with $\mathrm{view}_p(v_p) = \mathrm{view}_p(w)$, then $wc \in \mathrm{pref}(L)$.

The first two *language conditions* (LC1,LC2) have been defined before (in the different setting of *Message Sequence Charts* for (LC2) [1]), and their names are standard in the Mazurkiewicz trace community. However, they have only been considered separately; and the third notion (LC3) is new.

▶ **Definition 10** (Syntactical conditions)**.** For a sequential diamond deterministic automaton $A = (C, \to, In, Fin)$, we define the following conditions:

$(AC1)$ *forward diamond*: Whenever $s, s', t' \in C$, $(a, b) \in I$ with $s \xrightarrow{a} s'$ and $s \xrightarrow{b} t'$, there exists a state t with $s' \xrightarrow{b} t$ and $t' \xrightarrow{a} t$.

$(AC2)$ Whenever $s \in C$, if for all $p \in \mathrm{dom}(a)$ there exist $r^p, t^p \in C$ and words $w^p, (w')^p \in (\Sigma \setminus \Sigma_p)^*$, such that $r^p \xrightarrow{w^p} s$, $r^p \xrightarrow{(w')^p} t^p$ and $t^p \in Fin$, then $s \in Fin$.

$(AC3)$ Whenever $s \in C$ and $a \in \Sigma$, if for all $p \in \mathrm{dom}(a)$ there exist $r^p, t^p, x^p \in C$ and words $w^p, (w')^p \in (\Sigma \setminus \Sigma_p)^*$, such that $r^p \xrightarrow{w^p} s$ and $r^p \xrightarrow{(w')^p} t^p \xrightarrow{a} x^p$, then there exists $t' \in C$ with $s \xrightarrow{a} t'$.

The first *automaton condition* (AC1) has been defined earlier, while the two others are new. Our main theorem below shows that these local syntactical conditions have a global semantical implication.

To illustrate (LC3) and (AC3), consider the language L_3 in Section 2 (Figure 2(c)). We observe that L_3 does not meet (LC3) as $w = ab \in \mathrm{pref}(L_3)$, $\mathrm{dom}(c) = \{p, q\}$, $ab'c \in \mathrm{pref}(L_3)$ with $\mathrm{view}_p(ab') = [a] = \mathrm{view}_p(w)$ and $ad'bc \in \mathrm{pref}(L_3)$ with $\mathrm{view}_q(a'b) = [b] = \mathrm{view}_q(w)$ but $wc \notin \mathrm{pref}(L_3)$. Further, if A_3 is a deterministic automaton with $L(A_3) = L_3$, denoting by s_w the state reached after w, we consider state s_{ab} and action c. Then, letting $r^p = s_a, w^p = b', t^p = s_{ab}$ and $r^q = s_b, w^q = a', t^q = s_{a'b}$ it follows that A_3 does not satisfy (AC3).

▶ **Theorem 11.** *Let L be a regular language. Then, the following are equivalent:*

Conference title on which this volume is based on.
Editors: Anil Seth, Nisheeth Vishnoi - Bill Editors; pp. 7–16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

1. *There exists a (sequential, finite) deterministic diamond automaton $A = (C, \to, \{s_0\}, Fin)$ satisfying (AC1,AC2,AC3), with $\mathcal{L}(A) = L$, and such that every state is reachable from s_0 and every state can reach Fin.*
2. *L is closed under commutation and satisfies (LC1,LC2,LC3).*
3. *There exists a realistic asynchronous automaton AA with $\mathcal{L}(AA) = L$.*

The construction of a realistic asynchronous automaton first builds a deterministic asynchronous automaton by applying the algorithm from [8]. Then, a realistic asynchronous automaton is obtained by following the transformation described in the next section, which does not add any state or transition to [8] (though it may result in the removal of some states). For complexity issues, we expect that L is given by a deterministic diamond automaton A satisfying (AC1,AC2,AC3). Indeed, checking that A fulfills (AC1,AC2,AC3) is doable in polynomial time (see section 4).

3.2 Proof of Theorem 11

Theorem 11 is shown by proving ($1 \implies 2$), then ($2 \implies 3$), and last ($3 \implies 1$).

3.3 From (AC1,AC2,AC3) to (LC1,LC2,LC3): Theorem 11($1 \implies 2$)

We start the proof of Theorem 11 by proving that for any reachable and co-reachable diamond deterministic automaton A satisfying (AC1,AC2,AC3), then $\mathcal{L}(A)$ is closed by commutation and satisfies (LC1,LC2,LC3).

Firstly, as L is diamond, we get that $\mathcal{L}(A)$ is closed by commutation. Then:

($LC1$): Let $w \in \Sigma^*, (a, b) \in I$ such that $wa, wb \in \text{pref}(\mathcal{L}(A))$. By (AC1) we have that $In \xrightarrow{wab}$ is defined, and because every state of A is co-reachable, we have that $wab \in \text{pref}(L)$.

($LC2$): Let $w \in \Sigma^*$ such that for all $p \in \mathcal{P}$, there exists $v_p \in \mathcal{L}(A)$ with $\text{view}_p(v_p) = \text{view}_p(w)$. As A satisfies (AC1), $In \xrightarrow{w} s$ is defined. We let $v_p = \text{view}_p(v_p)v'_p$ and $w = \text{view}_p(w)w_p$ for all $p \in \mathcal{P}$. We define $In \xrightarrow{\text{view}_p(v_p)=\text{view}_p(w)} r^p$ and $r^p \xrightarrow{v'_p} t^p$. As $v_p \in L$ and A deterministic, we have that t^p is final. Also, we have $r^p \xrightarrow{w_p} s$. By (AC2), we thus have $s \in F$, that is, $w \in \mathcal{L}(A)$. (Note that, here and below, for a global state s we use s_p to denote the p-local component and hence we use s^p to denote the global state s indexed by process p. However for a word v, v_p always denotes indexing by process p and hence we do not need to change this notation.)

($LC3$): Let $w \in \text{pref}(L)$ and $a \in \Sigma$ such that for all $p \in \text{dom}(a)$, there exists $v_p a \in \text{pref}(\mathcal{L}(A))$ with $\text{view}_p(v_p) = \text{view}_p(w)$. We define s such that $In \xrightarrow{w} s$. We let $v_p = \text{view}_p(v_p)v'_p$ and $w = \text{view}_p(w)w_p$ for all $p \in \mathcal{P}$. We define $In \xrightarrow{\text{view}_p(v_p)=\text{view}_p(w)} r^p$ and $r^p \xrightarrow{v'_p} t^p \xrightarrow{a} x^p$. We have $r^p \xrightarrow{w_p} s$. By (AC3) and as the automaton is co reachable, we thus have $wa \in \text{pref}(\mathcal{L}(A))$.

3.4 From (LC1,LC2,LC3) to realistic AA: Theorem 11 ($2 \implies 3$)

Our basic strategy to construct a realistic AA is to use Theorem 8 (part 1.) to construct a *deterministic AA* from a given language L and then refine this AA to obtain a *realistic AA* which accepts the same language. For this, we will use as our template the recent construction from [8], and hence we begin by stating the relevant result and a definition that we need from this paper.

Conference title on which this volume is based on.
Editors: Anil Seth, Nisheeth Vishnoi - Bill Editors; pp. 8–16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

▶ **Definition 12** ([8])**.** A deterministic asynchronous automaton $AA = ((S_p)_{p\in\mathcal{P}}, \Delta, \{s_0\}, Fin)$ is called *locally rejecting* if for every process p, there is a set of states $R_p \subseteq S_p$ such that for each word w: $\mathrm{view}_p(w) \notin \mathrm{pref}(\mathcal{L}(AA))$ iff the p-local state reached by AA on w is in R_p.

Notice that if AA reaches R_p on a word w, then it does so on every extension of w, i.e., every word w' such that w is a prefix of w'. Obviously, no reachable global final state of AA has a (projected) component in R_p, which justifies why the states in R_p are called rejecting. Any Zielonka construction gives a naturally locally rejecting asynchronous automaton. In particular:

▶ **Theorem 13** ([8])**.** *Let A be a deterministic diamond automaton over the alphabet (Σ, dom). We can construct a deterministic locally rejecting asynchronous automaton AA with at most $|A|^{|\mathcal{P}|^2} \cdot 2^{2|\mathcal{P}|^4}$ states such that $\mathcal{L}(A) = \mathcal{L}(AA)$.*

Now we can prove our result as follows. Given a regular language L closed by commutation under (Σ, dom), we first build its minimal deterministic automaton A. It is then easy to check that A has the diamond property [7]. Now, we apply Theorem 13 to obtain a deterministic asynchronous automaton AA such that $\mathcal{L}(AA) = \mathcal{L}(A) = L$. Of course, AA may still have deadends (or global final states or not be locally enabled). Henceforth, for $s \xrightarrow{w} t$ with $t = (t_p)_{p\in\mathcal{P}}$, we will denote the (local) state t_p by $\delta^p_w(s)$. Notice that as the asynchronous automaton is deterministic, $\delta^p_w(s)$ is unique (if it exists) for each p, w, s.

First, we show that deadends can be avoided by using the locally rejecting property of AA. We remove all states of R_p from $AA = ((S_p)_{p\in\mathcal{P}}, (\Delta_a)_{a\in\Sigma}, \{s_0\}, Fin)$. That is, we define the asynchronous automaton $AA' = ((S'_p)_{p\in\mathcal{P}}, (\Delta'_a)_{a\in\Sigma}, \{s_0\}, Fin')$ with $S'_p = S_p \setminus R_p$ for all $p \in \mathcal{P}$, and $\Delta'_a = \Delta_a \cap \prod_{p\in\mathrm{dom}(a)} S'_p \times \prod_{p\in\mathrm{dom}(a)} S'_p$ for all $a \in \Sigma$, $Fin' = Fin \setminus R$, where $R = \{(s_p)_{p\in\mathcal{P}} \in \prod_{p\in\mathcal{P}} S_p \mid \exists q, s_q \in R_q\}$. We assume for convenience that $s_0 \notin R$ (else $L = \emptyset$ is trivial to deal with).

▶ **Lemma 14.** *AA' is deadend-free and $\mathcal{L}(AA') = \mathcal{L}(AA) = L$.*

Proof. Recall that AA' is the asynchronous automaton $AA' = ((S'_p)_{p\in\mathcal{P}}, (\Delta'_a)_{a\in\Sigma}, \{s_0\}, Fin')$ with $S'_p = S_p \setminus R_p$ for all $p \in \mathcal{P}$, and $\Delta'_a = \Delta_a \cap \prod_{p\in\mathrm{dom}(a)} S'_p \times \prod_{p\in\mathrm{dom}(a)} S'_p$ for all $a \in \Sigma$, and $Fin' = Fin \setminus R$ where $R = \{(s_p)_{p\in\mathcal{P}} \in (S_p)_{p\in\mathcal{P}} \mid \exists q, s_q \in R_q\}$.

We first show that $\mathcal{L}(AA') = \mathcal{L}(AA)$. By construction, if $w \in \mathcal{L}(AA')$, then $w \in \mathcal{L}(AA)$. For the converse, take $w \in \mathcal{L}(AA)$. Assume that $w \notin \mathcal{L}(AA')$. This means that for some prefix va of w, for some process q, $\delta^q_{va}(s_0) \in R_q$ (but $\delta^q_v(s_0) \in S'_q$). By definition of R_q, $\mathrm{view}_q(va) \notin \mathrm{pref}(\mathcal{L}(AA))$. This is a contradiction with $\mathrm{view}_q(va)$ is a prefix of $w \in \mathcal{L}(AA) = L$. Thus, we have $\mathcal{L}(AA') = \mathcal{L}(AA) = L$.

Now, let us prove that AA' has no deadend. Let $s = (s_p)_{p\in\mathcal{P}}$ be any global state reachable from the initial state s_0: that is, there exists w such that for all p, $\delta'^p_w(s_0) = s_p$. In particular, $\delta'^p_w(s_0)$ is defined for all p. It follows by definition of rejecting states, that $\mathrm{view}_p(w) \in \mathrm{pref}(\mathcal{L}(AA))$, for all process p. Now we can prove that $w \in \mathrm{pref}(\mathcal{L}(AA))$ using (LC1). Let us first sketch the case when $|\mathcal{P}| = 2$, denoting $\mathcal{P} = \{p, q\}$. We have $[w] = [uaxbb'y]$ with $[uax] = \mathrm{view}_p(w) \in \mathrm{pref}(\mathcal{L}(AA))$, $[ubb'y] = \mathrm{view}_q(w) \in \mathrm{pref}(\mathcal{L}(AA))$ and the domain of any action of ax and the domain of any action of $bb'y$ is disjoint. Now, ua and ub are in $\mathrm{pref}(\mathcal{L}(AA))$, and $(a, b) \in I$. So by (LC1), we get $uba \in \mathrm{pref}(\mathcal{L}(AA))$. But we also have $ubb' \in \mathrm{pref}(\mathcal{L}(AA))$ with $(a, b') \in I$, hence by (LC1) we get $ubb'a \in \mathrm{pref}(\mathcal{L}(AA))$. Now, a first induction on the length of $bb'y$ gives $ubb'ya \in \mathrm{pref}(\mathcal{L}(AA))$. And a second induction on the length of ax gives $ubb'yax \in \mathrm{pref}(\mathcal{L}(AA))$, that is $w \in \mathrm{pref}(\mathcal{L}(AA))$ (as $\mathrm{pref}(\mathcal{L}(AA))$ is closed by commutation). This completes the proof for the case of $|\mathcal{P}| = 2$. And the result follows by a third induction on the number of processes $|\mathcal{P}|$.

Conference title on which this volume is based on.
Editors: Anil Seth, Nisheeth Vishnoi - Bill Editors; pp. 9–16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Hence we have shown that for any reachable global state s, there exists $w \in \mathrm{pref}(\mathcal{L}(AA))$ with $s = \delta'_w(s_0)$. Thus there exists v such that $wv \in \mathcal{L}(AA) = \mathcal{L}(AA')$, and $\delta'_v(s)$ is final as AA' is deterministic. That is, no reachable global state s is a deadend. ☐

Now, AA' may still not be realistic due to final states that are global. To obtain local final states, we define $Fin_p = \{\delta^p_w(s_0) \in S_p \mid w \in L\}$ for all $p \in \mathcal{P}$ and let $Fin'' = \prod_{p\in\mathcal{P}} Fin_p$. Thus, we obtain a new asynchronous automaton $AA'' = ((S'_p)_{p\in\mathcal{P}}, (\Delta'_a)_{a\in\Sigma}, \{s_0\}, Fin'')$, differing from AA' only in its final states.

▶ **Lemma 15.** *AA'' is a realistic asynchronous automaton such that $\mathcal{L}(AA'') = \mathcal{L}(AA') = L$.*

Proof. By definition, AA'' is locally accepting, and it is deterministic since AA' and AA were deterministic. Also as $Fin' \subseteq Fin''$, setting the final states to be Fin'' does not add a deadend. Next, we show that $\mathcal{L}(AA'') = \mathcal{L}(AA') = L$. Take a word $w \in \mathcal{L}(AA'')$. Hence $\delta^p_w(s_0) \in Fin_p$ for all p. By definition of Fin_p, for all p there exists $v_p \in \mathcal{L}(AA') = L$ with $\delta^p_w(s_0) = \delta^p_{v_p}(s_0)$. We want to use (LC2) to conclude, but so far, there is no reason that $\mathrm{view}_p(v_p) = \mathrm{view}_p(w)$ for any p. We will thus build $v'_p \in L$ such that $\mathrm{view}_p(v'_p) = \mathrm{view}_p(w)$ for every p. Let $p \in \mathcal{P}$. It suffices to decompose $[v_p] = \mathrm{view}_p(v_p)\,[y_p]$. We then set $v'_p = \mathrm{view}_p(w)\,y_p$ and so $\mathrm{view}_p(v'_p) = \mathrm{view}_p(w)$ for all p. To obtain that $v'_p \in L$, we use a property of the Zielonka's construction from a deterministic automaton A: for all words w, w' such that $\delta^p_w(s_0) = \delta^p_{w'}(s_0)$, the state of A reached from the initial state after reading $\mathrm{view}_p(w)$ is the same as the state reached after reading $\mathrm{view}_p(w')$ (in other words, the p-state maintains the information about the state of A reached by the p-view of the executed trace). Now, let s be the state of the minimal deterministic automaton A for L reached after reading $\mathrm{view}_p(v'_p) = \mathrm{view}_p(w)$. This is also the state reached after reading $\mathrm{view}_p(v_p)$ because $\delta^p_w(s_0) = \delta^p_{v_p}(s_0)$ and by the property above. Reading y_p from s thus leads to a final state of A, as $\mathrm{view}_p(v_p)y_p \in L$ and the automaton is deterministic. Thus $v'_p = \mathrm{view}_p(w_p)y_p \in L$ too. Applying (LC2), we get $w \in L = \mathcal{L}(AA')$, and thus $L = \mathcal{L}(AA') = \mathcal{L}(AA'')$.

It remains to prove that AA'' is locally enabled: Let w^1, w^2, w^3 be words with $(s^i_p)_{p\in\mathcal{P}}$ the state reached on w^i in AA'' (the state is unique by determinism of AA''). Let a be an action such that $s^3_p \in \{s^1_p, s^2_p\}$ for all $p \in \mathrm{dom}(a)$. Assume that $(s^1_p)_{p\in\mathcal{P}} \xrightarrow{a} (t^1_p)_{p\in\mathcal{P}}$ and that $(s^2_p)_{p\in\mathcal{P}} \xrightarrow{a} (t^2_p)_{p\in\mathcal{P}}$ for some $(t^1_p)_{p\in\mathcal{P}}$ and $(t^2_p)_{p\in\mathcal{P}}$. As AA'' is deadend-free, it means that w^1a and w^2a are in $\mathrm{pref}(L)$. We want to show that $w^3a \in \mathrm{pref}(L)$, which will imply that there exists $(t^3_p)_{p\in\mathcal{P}}$ with $(s^3_p)_{p\in\mathcal{P}} \xrightarrow{a} (t^3_p)_{p\in\mathcal{P}}$, that is local enabledness of AA''.

We now use, as above, the property that if w, w' reach in AA the same local state s_p on process p, then $\mathrm{pref}_p(w), \mathrm{pref}_p(w')$ reach the same state s in A. We decompose $\mathrm{dom}(a)$ into two sets: P_1, P_2 such that $p \in P_1$ iff $s^3_p = s^1_p$. We have that for all $p \in P_1$, both $\mathrm{pref}_p(w^3)$ and $\mathrm{pref}_p(w^1)$ reach s^1_p on process p. Hence $\mathrm{pref}_p(w^3)$ and $\mathrm{pref}_p(w^1)$ reach in A the same state s^1. Writing $w^1 = \mathrm{pref}_p(w^1)v^p$, we have that for $w^p = \mathrm{pref}_p(w^3)v^p$, $w^pa \in \mathrm{pref}(L)$, with $\mathrm{view}_p(w^p) = \mathrm{pref}_p(w^3) = \mathrm{view}_p(w^3)$. The same reasoning for $p \in P_2$ gives the existence of some w^p with $\mathrm{view}_p(w^p) = \mathrm{view}_p(w^3)$ and $w^pa \in \mathrm{pref}(L)$. It suffices to apply the definition of (LC_3) to deduce that $w^3a \in \mathrm{pref}(L)$, which implies that AA'' is locally enabled. ☐

Notice that Fin_p can be computed in time $O(|A|)$. Indeed, we reuse the property that a local p states $\delta^p_w(s_0)$ keeps the state s of A reached from the initial state after reading $\mathrm{view}_p(w)$. To know whether there exists $w' \in L$ such that $\delta^p_w(s_0) = \delta^p_{w'}(s_0)$, it suffices to perform one graph search(e.g. DFS) using edges of A except the ones using letters in Σ_p, and see whether a final state of A is reachable from s by this search. This takes time $O(|A|)$.

Conference title on which this volume is based on.
Editors: Anil Seth, Nisheeth Vishnoi - Bill Editors; pp. 10–16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

3.5 From realistic AA to (AC1,AC2,AC3): Theorem 11 ($3 \implies 1$)

Given a realistic asynchronous automaton $AA = ((S_p)_{p\in\mathcal{P}}, \Delta, In, Fin)$, we consider its global (sequential) automaton $S(AA) = (C, \to, In, Fin)$. We can then prove that (the reachable part of) $S(AA)$ satisfies $(AC1, AC2, AC3)$. As AA is *deadend-free*, every reachable state s of $S(AA)$ is also co-reachable, that is, there exists w such that $s \xrightarrow{w} s'$ with $s' \in Fin$. First, by construction, as AA is deterministic, $S(AA)$ is deterministic. Further, $S(AA)$ is diamond for any AA as mentioned earlier. Then:

$(AC1)$: Let $s, s', t' \in C$ and $(a, b) \in I$ with $s \xrightarrow{a} s'$ and $s \xrightarrow{b} t'$, and s is reachable. That is, there exists $In \xrightarrow{w} s$. We have $In \xrightarrow{wab}$ is defined: indeed, $In \xrightarrow{wa} s' = (s'_p)_{p\in\mathcal{P}}$. Now, notice that for all $p \notin \mathrm{dom}(a)$, $s'_p = s_p$. Also, $(s'_p)_{p\in\mathcal{P}} \xrightarrow{b} (t_p)_{p\in\mathcal{P}}$, with $t_p = s'_p$ for all $p \notin \mathrm{dom}(b)$, and $t_p = t'_p$ for all $p \in \mathrm{dom}(b)$, as $\mathrm{dom}(a) \cap \mathrm{dom}(b) = \emptyset$. Hence (AC1) is satisfied by $S(AA)$.

$(AC2)$: Assume that $s \in C$ and that for each $p \in \mathcal{P}$, there exist tuples of states and words $(r^p, t^p, w^p, (w')^p)_{p\in\mathcal{P}}$ such that $r^p \in C$, $t^p \in Fin$ and $w^p, (w')^p \in (\Sigma \setminus \Sigma_p)^*$ and $r^p \xrightarrow{w^p} s$ and $r^p \xrightarrow{(w')^p} t^p$. Because $w^p, (w')^p \in (\Sigma \setminus \Sigma_p)^*$, and by definition of $S(AA)$, we have that s and t^p have the same p state, say s_p. Now, t^p is final and AA has *local final states*, so it means that s_p is final. As this is true for all $p \in \mathcal{P}$, it means that s itself is final, since it is a product of all local final states, i.e., $s \in Fin$.

$(AC3)$: Assume that $s \in C$ and for some $a \in \Sigma$, for all $p \in \mathrm{dom}(a)$ there are tuples $(r^p, t^p, x^p, w^p, (w')^p)_{p\in\mathrm{dom}(a)}$ such that $r^p, t^p, x^p \in C$ and $w^p, w'^p \in (\Sigma \setminus \Sigma_p)^*$, $r^p \xrightarrow{w^p} s$, $r^p \xrightarrow{(w')^p} t^p \xrightarrow{a} x^p$. Again, by definition of $S(AA)$, s and t^p have the same local p-state s_p. Hence, for every $p \in \mathrm{dom}(a)$, s_p enables a (i.e., a can be fired from s_p), and because AA is locally enabled, there exists s' with $s \xrightarrow{a} s'$, which proves (AC3).

3.6 Corollaries

In many (but not *all*) cases, there is an automaton for L satisfying (ACi) as soon as L is (LCi), for $i = 1, 2, 3$. We first consider the cases where all states are final (see [22]), in which case (LC2) and (AC2) are not useful: The first corollary follows directly by removing (LC2) and (AC2) from Theorem 11, and hence we just state it in full below.

▶ **Corollary 16.** *Let L be a regular language. Then, the following are equivalent:*

1. *There exists a (sequential,finite) deterministic diamond automaton $A = (C, \to, In, Fin)$ such that every state is reachable from In and every state can reach Fin, with $\mathcal{L}(A) = L$, and satisfying (AC1) and (AC3).*
2. *L satisfies (LC1) and (LC3) and is closed under commutation.*
3. *There exists a deterministic, deadend-free, locally enabled asynchronous automaton AA with $\mathcal{L}(AA) = L$, and AA has at most as many states as the asynchronous automaton obtained by applying [8] to the minimal deterministic (sequential) automaton accepting L.*

The following results are useful in section 4 to test if a given asynchronous automaton is realistic, that is, for testing if each of the conditions (LC1),(LC2),(LC3) hold. The following corollary does not follow immediately from what we proved earlier, since it states that we can choose A to be the minimal deterministic automaton. Hence, we state this result in full and also provide an explicit proof.

Conference title on which this volume is based on.
Editors: Anil Seth, Nisheeth Vishnoi - Bill Editors; pp. 11–16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

▶ **Corollary 17.** *Let L be a regular language. Then, the following are equivalent:*

1. *The minimal (sequential, finite) deterministic diamond automaton $A = (C, \rightarrow, In, Fin)$ of L satisfies (AC1).*
2. *L satisfies (LC1) and is closed under commutation.*
3. *There exists a deterministic, deadend-free asynchronous automaton AA with $\mathcal{L}(AA) = L$, and AA has at most as many states as the asynchronous automaton obtained by applying [8] to the minimal deterministic (sequential) automaton accepting L.*

Proof. [*of Corollary 17*] (1) $\implies$ (2) $\iff$ (3) follows from the proof of Theorem 11. We finish the proof by proving (2) $\implies$ (1), that is, for L a regular language closed by commutation satisfying (LC1) and A be the minimal deterministic automaton with $\mathcal{L}(A) = L$, we have that A satisfies (AC1).

Let s, s', t' be three states such that $s \xrightarrow{a} s'$ and $s \xrightarrow{b} t'$ with $(a, b) \in I$. Further let v be any word such that $s_0 \xrightarrow{v} s$ (for initial state s_0) and w, w' words such that $s' \xrightarrow{w} r$ and $t' \xrightarrow{w'} r'$ with $r, r' \in F$. All three words exist by minimality of A. Then $vaw, vbw' \in \mathcal{L}(A)$, and so, by (LC1), $vab \in \text{pref}(\mathcal{L}(A))$. That is, there exists a word u with $vabu \in \mathcal{L}(A)$. Now since A is deterministic, we have the existence of t such that $s' \xrightarrow{b} t$. Further, as $\mathcal{L}(A)$ is closed by commutation, we also have $vbau \in \mathcal{L}(A)$, and thus there exists a state $\bar{t}$ with $t' \xrightarrow{a} \bar{t}$. But now, by the minimality of A, it follows that $\bar{t} = t$. To see this, note that for all words u such that $vabu \in \mathcal{L}(A)$, we also have $vbau \in \mathcal{L}(A)$. This implies that t and $\bar{t}$ are "equivalent" with respect to the language accepted, and hence in the (unique) minimal automaton A, we have $\bar{t} = t$.

◻

In the proof of subsection 3.3, *both* (AC1) and (AC2) are used to prove (LC2). However, if deadends are allowed, one can alternatively use only (AC2) to prove (LC2) if the deterministic (sequential) automaton $A = (C, \rightarrow, In, Fin)$ is *complete*. We state and prove this result below.

▶ **Corollary 18.** *Let L be a regular language. Then, the following are equivalent:*

1. *There exists a deterministic diamond complete automaton $A = (C, \rightarrow, In, Fin)$ with $\mathcal{L}(A) = L$ satisfying (AC2).*
2. *L satisfies (LC2) and is closed under commutation.*
3. *There exists a deterministic (locally enabled) asynchronous automaton AA with local final states and $\mathcal{L}(AA) = L$. Also, if $|S|$ is the number of states of the AA obtained by applying [8] to the minimal deterministic (sequential) automaton accepting L, then AA has at most $|S| - |\mathcal{P}|$ states.*

Proof. (2) $\implies$ (3) $\implies$ (1) follows from the proof of Theorem 11. We finish the proof by proving (1) $\implies$ (2), that is, for any complete diamond deterministic automaton A satisfying (AC2), then $\mathcal{L}(A)$ is closed by commutation and satisfies (LC2).

Firstly, as L is diamond, we get that $\mathcal{L}(A)$ is closed by commutation. Then: Let $w \in \Sigma^*$ such that for all $p \in \mathcal{P}$, there exists $v_p \in \mathcal{L}(A)$ with $\text{view}_p(v_p) = \text{view}_p(w)$. As A is complete, $In \xrightarrow{w} s$ is defined. We let $v_p = \text{view}_p(v_p)v'_p$ and $w = \text{view}_p(w)w_p$ for all $p \in \mathcal{P}$. We define $In \xrightarrow{\text{view}_p(v_p)=\text{view}_p(w)} r^p$ and $r^p \xrightarrow{v'_p} t^p$. As $v_p \in L$ and A deterministic, we have that t^p is final. Also, we have $r^p \xrightarrow{w_p} s$. By (AC2), we thus have $s \in F$, that is, $w \in \mathcal{L}(A)$. This ends the proof. ◻

Notice that the above corollary uses the fact that if deadends are allowed, then one can always ensure that an asynchronous automaton is locally enabled, by adding a state on each

Conference title on which this volume is based on.
Editors: Anil Seth, Nisheeth Vishnoi - Bill Editors; pp. 12–16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

process which is a deadend. This fact also means that L satisfying (LC3) alone is not useful in terms of asynchronous automata.

Finally, if we keep $(LC1)$ *and* $(LC2)$, we can again use the proofs of the previous section to immediately conclude the following easy yet interesting corollary (when considering action based controllers [11]):

▶ **Corollary 19.** *Let L be a regular language. Then, the following are equivalent:*

1. *There exists a (sequential, finite) deterministic diamond automaton $A = (C, \rightarrow, In, Fin)$ such that every state is reachable from In and every state can reach Fin, with $\mathcal{L}(A) = L$, and satisfying (AC1) and (AC2).*
2. *L satisfies (LC1) and (LC2) and is closed under commutation.*
3. *There exists a deterministic, deadend-free asynchronous automaton AA with local final states and $\mathcal{L}(AA) = L$, and AA has at most as many states as the asynchronous automaton obtained by applying [8] to the minimal deterministic (sequential) automaton accepting L.*

4 Testing for Realistic Asynchronous Automata

We now explain how to check each property (LCi) and (ACi) for all $i = 1, 2, 3$.

Testing automata restrictions (ACi): Let A be an automaton, possibly non deterministic. To test (AC1), for each state s we need to check if it has a pair of outgoing transitions on actions that are independent, and if so, test for the existence of a common state that can be reached, giving a complexity quadratic in the number of states and transitions of A.

To test (AC2), we perform one graph search (e.g. DFS) from each state $s \notin Fin$ and for each process $p \in \mathcal{P}$ to compute the set R_p^s of states r with $r \xrightarrow{w_p'} s$ for some $w_p' \in (\Sigma \setminus \Sigma_p)^*$. This is a simple graph search in linear time, done on the graph with reverse edge and where edges using letters of Σ_p have been deleted. Computing R_p^s thus takes time linear in the size of the graph. We then perform another graph search from R_p^s to compute the set T_p^s of *final* states t such that $r \xrightarrow{w_p} t$, for some $r \in R_p^s$ and $w_p \in (\Sigma \setminus \Sigma_p)^*$. Now A does not satisfy (AC2) iff $\exists s, \forall p, T_p^s \neq \emptyset$. Hence, these two graphs search are applied $|\mathcal{P}|$ times (one for each process) and $|A|$ times (one for each state s). Overall, it takes time $O(|\mathcal{P}| \cdot |A|^2)$. The test of (AC3) is similar, with the same complexity.

Testing language restrictions (LCi): We now describe how to test language restrictions. We assume that the language L to be tested is given as an automaton (possibly non deterministic). First, using Corollary 17, one has a simple way to test for (LC1): compute the minimal deterministic automaton A with $\mathcal{L}(A) = L$, and test (AC1) using the polynomial procedure given above at the beginning of the section. This gives a PSPACE algorithm. The complexity is polynomial if the starting automaton is deterministic.

In order to test for (LC2), we use Corollary 18. Indeed, we build the asynchronous automaton AA from A as if $\mathcal{L}(A)$ satisfies (LC2). This can only add executions to the language, as final states are possibly added. Then we test whether $\mathcal{L}(AA) \subseteq \mathcal{L}(A)$. If the inclusion holds, then A satisfies (LC2), else A does not satisfy (LC2). This gives a PSPACE algorithm. If $\mathcal{P}$ is not part of the input and A is deterministic, then it is polynomial time. Notice that one cannot resort, as in the case of (LC1), to using the minimal automaton associated to L. This minimal automaton may not necessarily satisfy (AC2), even if L satisfies (LC2). For instance, consider the language $L_4 = \{\epsilon, a_1, b_1, a_1b_1, b_1a_1\} \cup \{a_ib_jc, b_ja_ic \mid i, j \in \{1, 2\}\}$ with $\text{dom}(a_i) = p, \text{dom}(b_i) = q$ for all $i, j \in \{1, 2\}$ and $\text{dom}(c) = \{p, q\}$. There

Conference title on which this volume is based on.
Editors: Anil Seth, Nisheeth Vishnoi - Bill Editors; pp. 13–16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

is a state t in the minimal automaton with $s_0 \xrightarrow{a_1} s \xrightarrow{b_2} t$ and $s_0 \xrightarrow{b_1} s' \xrightarrow{a_2} t$, with s, s' final, meaning if (AC2) holds that t is final, a contradiction. Finally, to test (LC3), we again implement L into an AA and test if $S(AA)$ satisfies (AC3). As described in the proof of Theorem 11, if $\mathcal{L}(A)$ satisfies (LC3), then $S(AA)$ satisfies (AC3). Conversely, if $S(AA)$ satisfies (AC3), the proof also shows that $\mathcal{L}(A)$ satisfies (LC3). This gives a PSPACE algorithm. The complexity is polynomial if $\mathcal{P}$ is not part of the input and A is deterministic.

Note that while the algorithms to test for (LC1),(LC2),(LC3) may be PSPACE, they are actually polynomial in the size of the asynchronous automaton AA we want to obtain. As shown below, obtaining the global state space for AA is actually feasible in a number of examples, and hence testing for (LC1), (LC2) and (LC3) is also doable in these cases.

5 Experiments

In this section, we report our experiments on the implementation of the results in this paper, based on the construction from [8], which has not been implemented before. To give a point of comparison, we also report results obtained using the only previous implementation prototype for Zielonka constructions from [22], which implements the original synthesis algorithms from [23] and the heuristic in [22].

We report below the results of several systems that are (distributively) implemented using these three algorithms: We will denote by *heuristic* the heuristic from [22], by *original* the original Zielonka's construction from [23], and by *local* and *global* two different metrics for our new implementation as described below. *heuristic* takes into account the structure of the automaton (using ideas from the theory of regions [17]) to identify small asynchronous automata before generating the whole global state space. Since such structural properties cannot be found for every regular commutation-closed language, it uses the equivalence in *original* as an upper bound. Hence, the state space produced by *heuristic* is never bigger than the one of *original*. On the other hand, *original* uses a generic construction which always produces an asynchronous automaton. In contrast to these two algorithms producing global state spaces, our implementation produces the local state space directly. Further, ours is an on-the-fly symbolic algorithm. As argued in [20], on-the-fly computation allows to implement distributed algorithms whose global state space cannot be explicitly enumerated: with 4 processes, the timestamping used in [23, 8] can give rises to 10^7 global states, and to 10^{16} global states with 5 processes. But for symbolic algorithms (e.g., the one from [8] that we implement), 5 processes means maintaining 128 bits of information, which can be updated in time polynomial in the number of bits. To produce and compare the results of all algorithms, we report global state spaces, thus limiting ourselves to less than 4 processes.

The results are compiled in the table below. The first column gives the names of the input systems, while second and third provide their number of states $|A|$ and processes $|P|$, respectively. The fourth column states the syntactical properties (ACi) of the automaton A. The next three columns give the number of *global* states produced by each of the algorithms. As noted earlier, the new prototype does not need to compute the global state space, unlike [22, 23]. The column *local* reports the total number of *local* states generated by our algorithm, which is closer to what would be used in practice (but is still larger than what is explored on-the-fly). The last row describes the properties (DF for deadend-free, LA for locally accepting, LE for locally enabled, and R for realistic) of the obtained asynchronous automaton using the new implementation, all being deterministic.

The first four systems come directly from distributed algorithms: a mutual exclusion protocol with semaphores with 2 different distribution alphabets referred to as mutex-a and mutex-b; a simple program with 3 processes denoted simple; and a dining philosopher

Conference title on which this volume is based on.
Editors: Anil Seth, Nisheeth Vishnoi - Bill Editors; pp. 14–16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

protocol phil. All these examples except simple, from [22], satisfy AC1,AC2,AC3.

	$\|A\|$	$\|\mathcal{P}\|$	satisfies	*heuristic*	*original*	*global*	*local*	satisfies
mutex-a	13	3	(AC1,AC2,AC3)	13	1493	271	126	(R)
mutex-b	14	4	(AC1,AC2,AC3)	14	34	22	16	(R)
simple	3	3	(AC1,AC2)	5	12	12	9	(DF+LA)
phil	5	4	(AC1,AC2,AC3)	5	70	71	60	(R)
prop2	6	2	(AC2)	188	188	36	21	(LA+LE)
prop3	11	3	(AC2)	639	639	240	92	(LA+LE)
L4	8	2	(AC1,AC3) + (LC2)	N/A	10	10	5	(R)

For these first 4 systems, the new prototype gives an implementation with lesser states than *original* (up to 10 times), although not as good as *heuristic*. Adapting ideas from *heuristic* [22] might reduce the size of the produced implementation. The two systems propN correspond to a distributed supervisor which detects whether a critical section has been accessed by 2 processes in parallel among N processes. On each process, it observes entry and exit of the critical section and synchronization between processes, and detects if a process which enters the critical section has been informed that other processes have exited it. This supervisor works on any possible (correct or not) mutual exclusion protocol, and detects on-the-fly whether the critical section was accessed by 2 processes concurrently. On this example, *heuristic* does not do better than *original*. The number of local states is around 8 times smaller, while global states are around 4 times smaller than previous implementations. As (LC1) does not hold, a realistic implementation is not possible here.

Notice that *heuristic* is guaranteed to return correct results only when all states are final [22], which is the case for the first 6 systems. The last system we experiment on is the minimal automaton L4 for language L_4 from the previous section (L4 does not satisfy (AC2), although L_4 satisfies (LC2)). Some states of this automaton are not final and the implementation created by *heuristic* is incorrect: its language is strictly larger than L_4. On the other hand, implementations produced by *original* and the new prototype accept exactly L_4. Details on the experiments can be found online at: `http://is.gd/fsttcs13_benchmark`.

6 Related Work and Conclusion

In this paper, we have provided syntactical and semantical characterizations of languages corresponding to several variants of *realistic* asynchronous automata. We designed algorithms to obtain the distributed implementation, test for the different characterizations and showed their experimental effectiveness. Our results subsume past results and answer several open questions. Corollary 17 subsumes what was claimed in [17] and proved in [22] (Theorem 2) in the subcase where the language is prefix closed. It is also worth mentioning that [1] had introduced the notion of causal closure for Message Sequence Graphs, which are a distributed model using message passing for communication. Our notion of causal closure is directly adapted from theirs. However, unlike in Corollary 18, only one direction was proved for their model. Also, they lack the syntactical characterization using (AC2) which holds by Theorem 11. Also, Corollary 18 answers an open question in the conclusion of [2].

As future work, it would be interesting to consider alternative ways of inputing the language, e.g., by giving a set of representatives to represent the language. This would avoid starting from a large automaton, and may lead to a smaller distributed implementation.

Acknowledgments: This work was supported by Romanian NASR project PN-II-ID-PCE-2011-3-0688 (MuVeT), INRIA Associated team DISTOL and DST/INSPIRE faculty award [IFA12-MA-17].

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. Compositional synthesis of asynchronous automata *Theor. Comput. Sci.*, 412(29):3701-3716, 2011.

3 B. Bollig, J.-P. Katoen, C. Kern and M. Leucker. Learning Communicating Automata from MSCs. *IEEE Trans. Software Eng.* 36(3):390-408, 2010.

4 N. Baudru and R. Morin. Unfolding Synthesis of Asynchronous Automata. Proc. of *CSR'06*, LNCS 3967, pp. 46-57, 2006.

5 N. Baudru and R. Morin. Synthesis of Safe Message-Passing Systems. Proc. of *FSTTCS'07*, LNCS 4855, pp. 277-289, 2007.

6 R. Cori, Y. Métivier and W. Zielonka. Asynchronous Mappings and Asynchronous Cellular Automata. *Inf. and Comput.*, 106(2):159-202, 1993.

7 V. Diekert and G. Rozenberg, editors. *The Book of Traces.* In particular, Chapter 8 by V. Diekert and A. Muscholl. World Scientific, Singapore, 1995.

8 B. Genest, H. Gimbert, A. Muscholl and I. Walukiewicz. Optimal Zielonka-like Construction. Proc. of *ICALP'10*, LNCS 6199, pp. 52-63, 2010.

9 B. Genest, H. Gimbert, A. Muscholl, I. Walukiewicz. Asynchronous Games over Tree Architectures. Proc. of *ICALP'13*, LNCS 7966, pp. 275-286, 2013.

10 B. Genest, D. Kuske and A. Muscholl. A Kleene Theorem and Model Checking for a Class of Communicating Automata. *Inf. and Comput.*204(6):920-956, 2006.

11 P. Gastin, B. Lerman and M. Zeitoun. Distributed Games with Causal Memory Are Decidable for Series-Parallel Systems. Proc. of *FSTTCS'04*, LNCS 3328, 2004.

12 B. Genest and A. Muscholl. Constructing Exponential-size Deterministic Zielonka Automata. Proc. of *ICALP'06*, LNCS 4051, pp. 565-576, 2006.

13 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.

14 N. Klarlund, M. Mukund and M. Sohoni. Determinizing Asynchronous Automata. Proc. of *ICALP'94*, LNCS 820, pp. 130-141, 1994.

15 D. Kuske. Regular sets of infinite message sequence charts. In *Inf. and Comput.*, 187(1):80-109, 2003.

16 A. Mazurkiewicz. Concurrent program schemes and their interpretation. Technical report, DAIMI Report PB-78, Aarhus University, 1977.

17 M. Mukund. From global specification to local implementations. In *Synthesis and Control of Discrete Event Systems*, Kluwer, pp. 19-34, 2002.

18 S. Krishna and A. Muscholl. A quadratic construction for Zielonka automata with acyclic communication structure. In TCS, 503: 109-114, 2013.

19 M. Mukund, K. Narayan Kumar and M. Sohoni. Synthesizing Distributed Finite-State Systems from MSCs. *TCS* 290(1):221-239, 2003.

20 M. Mukund and M. Sohoni. Keeping Track of the Latest Gossip in a Distributed System. In *Distr. Computing* 10(3):137-148, 1997.

21 G. Pighizzini. Synthesis of Nondeterministic Asynchronous Automata. In *Algebra, Logic and Applications* 5, pp. 109-126, 1993.

22 A. Stefanescu, J. Esparza, and A. Muscholl. Synthesis of distributed algorithms using asynchronous automata. Proc. of *CONCUR'03*, LNCS 2761, pp. 27-41, 2003.

23 W. Zielonka. Notes on finite asynchronous automata. In *R.A.I.R.O. - Informatique Théorique et Applications*, 21:99-135, 1987.

24 W. Zielonka. Safe Executions of Recognizable Trace Languages by Asynchronous Automata. *Proc. of Logic at Botik 1989*, LNCS 363, pp. 278-289, 1989.

Conference title on which this volume is based on.
Editors: Anil Seth, Nisheeth Vishnoi - Bill Editors; pp. 16–16
Leibniz International Proceedings in Informatics
LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany