Taming Distributed Asynchronous Systems

Anca Muscholl

LaBRI, University Bordeaux, France

Abstract. This extended abstract surveys some analysis techniques for distributed, asynchronous systems with two kinds of synchronization, shared variables and fifo channels.

1 Introduction

Modeling distributed, asynchronous systems so that computer-assisted analysis becomes feasible, is an on-going challenge in both theory and practice. Several automata-based models for such systems have been proposed and studied over the past twenty years, capturing various aspects of distributed behavior. Depending on the motivation, such models fall into two large categories. In the first one we find rather simple models, capturing basic synchronization mechanisms: Petri nets, communicating automata, ... They were studied for algorithmic properties and/or their expressive power. In the second category we see more sophisticated models, that were conceived for supporting practical system design, like Harel's statecharts, or Lynch's I/O automata. It is clear that being able to develop automated verification techniques requires a good understanding of the simpler models, in particular since more complex ones are often built as a combination of basic models.

In this survey we address the issue of analyzing networks of (mostly finite-state) processes with two kinds of communication mechanisms, unbounded fifo channels and shared variables. We also go one step beyond verification, or model-checking, by addressing the synthesis problem in the shared-variable case. Synthesis, and in particular controller synthesis, is a challenging problem even for such simple models as the ones considered in this survey, since it essentially amounts to solve distributed games. This topic is still rather poorly understood and open for future research, in spite of considerable efforts and partial results obtained during the past decade.

2 Models of distributed computation

The architecture of a distributed asynchronous system consists of a set of processes $\mathcal{P}$ related by links, and we will consider it as *fixed*. Such links may correspond for instance to communication channels or to shared variables. We do not discuss here other synchronization mechanisms that appear in the literature, like e.g. state observation or signals.

Zielonka's *asynchronous automata* is an asynchronous model based on shared variables. It has its roots in the theory of Mazurkiewicz traces [28], which came up in the late seventies in connection with the semantics of 1-safe Petri nets (the reader may find in [11] a wealth of results about traces). Asynchronous automata provide one of the first highly non-trivial examples of distributed (closed) synthesis, as expressed in Theorem 1 below.

Given a finite set $\mathcal{P}$ of processes, we consider an alphabet of actions Σ and a location function $dom : \Sigma \to (2^{\mathcal{P}} \setminus \emptyset)$, associating with each action a non-empty set of processes. The location mapping dom defines in a natural way an *independence relation* I: two actions $a, b \in \Sigma$ are independent (denoted as $(a, b) \in I$) if they synchronize disjoint sets of processes, i.e., if $dom(a) \cap dom(b) = \emptyset$. One can define the relation $\sim_I$ on Σ^* as the equivalence generated by all pairs $(uabv, ubav)$, for $(a, b) \in I$ and $u, v \in \Sigma^*$. A *trace* is then a $\sim_I$-equivalence class, and a trace language is a word language closed under $\sim_I$.

Alternatively, traces can be viewed as labeled pomsets (see an example in Figure 1), and the set of (labeled) linearizations of such a pomset corresponds to the $\sim_I$-equivalence class $[u]$ of any of these linearizations $u \in \Sigma^*$.

A (deterministic) *asynchronous automaton* is a tuple

$$\mathcal{A} = \langle \{S_p\}_{p \in \mathcal{P}}, s_0, \{\delta_a\}_{a \in \Sigma}, F \rangle \,,$$

where

- S_p is a finite set of (local) states of process p,
- $s_0 \in \prod_{p \in \mathbb{P}} S_p$ is a (global) initial state,
- $\delta_a : \prod_{p \in dom(a)} S_p \to \prod_{p \in dom(a)} S_p$ is a transition relation; so on a letter $a \in \Sigma$ it is a partial function on tuples of states of processes in $dom(a)$,
- $F \subseteq \prod_{p \in \mathcal{P}} S_p$ is a set of final (accepting) states.

An asynchronous automaton can be seen as a sequential automaton with the state set $S = \prod_{p \in \mathcal{P}} S_p$ and transitions $s \xrightarrow{a} s'$ if $((s_p)_{p \in dom(a)}, (s'_p)_{p \in dom(a)}) \in \delta_a$, and $s_q = s'_q$ for all $q \notin dom(a)$. By $L(\mathcal{A})$ we denote the set of words labeling accepting runs. This definition has an important consequence. If $(a, b) \in I$ then the same state is reached on the words ab and ba. More generally, whenever $u \sim_I v$ and $u \in L(\mathcal{A})$ then $v \in L(\mathcal{A})$, too. This means that $L(\mathcal{A})$ is a trace language.

Example 1. Let us consider the asynchronous automaton $\mathcal{A}$ defined by $S_p = \{0\}$, $S_q = S_r = \{0, 1\}$, and transition function $\delta_a(s_p, s_q) = (s_p, \neg s_q)$ if $s_q = 1$ (undefined otherwise), $\delta_d(s_r) = \neg s_r$ if $s_r = 1$ (undefined otherwise), $\delta_b(s_q, s_r) = (1, 1)$ if $s_q \wedge s_r = 0$ (undefined otherwise) and $\delta_c(s_p) = s_p$. Starting with $s_0 = (0, 0, 0)$, an accepting run of $\mathcal{A}$ checks that between any two successive b-events, there is either an a or a d (or both), and there is a b-event before all a and d.

One of the deepest results of trace theory is Zielonka's construction of a deterministic asynchronous automaton from a finite-state one. One can see it as an example of distributed *closed* synthesis, i.e., without any environment.

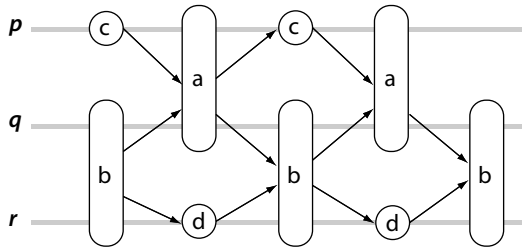


Fig. 1. The pomset associated with the trace $t = [c\,b\,a\,d\,c\,b\,a\,d\,b]$, with $dom(a) = \{p, q\}$, $dom(b) = \{q, r\}$, $dom(c) = \{p\}$, $dom(d) = \{r\}$.

Theorem 1. *[40] Given a finite automaton $\mathcal{A}$ accepting the trace language $L(\mathcal{A})$, a deterministic asynchronous automaton $\mathcal{B}$ can be effectively constructed with $L(\mathcal{A}) = L(\mathcal{B})$.*

The above construction has received a lot of interest, and a series of papers aimed at improving it algorithmically (see e.g. [10, 30, 19, 17]). Currently the best construction starting with a DFA $\mathcal{A}$ is polynomial in the size of $\mathcal{A}$ and simply exponential in the number of processes. Surprisingly, it is rather difficult to come up with a matching lower bound (see [17] for partial results). As explained in Section 3, this construction plays a fundamental role in other settings of distributed synthesis, as for instance for communicating automata, that we present next.

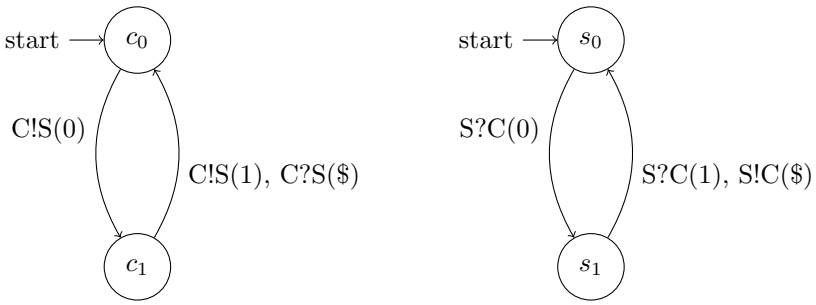
A communicating automaton (*CA* for short) is parametrized by a set $\mathcal{P}$ of processes, a set of point-to-point fifo channels $Ch \subseteq \mathcal{P}^2 \setminus id_{\mathcal{P}}$, and a set of message contents *Msg*. It is a tuple $\mathcal{A} = \langle (\mathcal{A}_p)_{p\in\mathcal{P}}, \Sigma, F\rangle$ where

- each $\mathcal{A}_p = (S_p, \rightarrow_p, s_p^0)$ is a finite labeled transition system with state space S_p, transition relation $\rightarrow_p \subseteq S_p \times \Sigma_p \times S_p$, and initial state $s_p^0 \in S_p$; the local action alphabet Σ_p consists of send actions (denoted as $p!q(m)$, with $(p, q) \in Ch$, $m \in Msg$), receive actions (denoted as $p?r(m)$, with $(r, p) \in Ch$, $m \in Msg$), and local actions.
- $F \subseteq \prod_{p\in\mathcal{P}} S_p$ is a set of *global* final states.

We denote the product $S := \prod_{p\in\mathcal{P}} S_p$ as set of *global states*.

The behavior of a CA is defined as the behavior of an infinite labeled transition system, by considering the possible (local) transitions on the set of configurations of the CA. A *configuration* of the CA $\mathcal{A}$ consists of a global state, together with a word from Msg^* for each channel $(p, q) \in Ch$. Transitions are defined in the usual way: the effect of an action $a \in \Sigma_p$ is to change the S_p state component according to $\mathcal{A}_p$, and to perform the obvious modification on one channel of p, according to a being a send of message m from p to q (written as $a = p!q(m)$) or a receive of m on p from r (written as $a = p?r(m)$).

Example 2. The CA in the figure below describes the communication between two (finite-state) processes C and S, connected through one channel in each direction. The set of message contents is $Msg = \{0, 1, \$\}$. From the initial configuration $\langle (c_0, s_0), (\varepsilon, \varepsilon)\rangle$ (say, (C, S) is the first channel) one can reach e.g. the configurations $\langle (c_1, s_0), (010, \varepsilon)\rangle$ and $\langle (c_0, s_0), (101, \$)\rangle$, but not $\langle (c_0, s_0), (0101, \$)\rangle$. For instance, $\langle (c_0, s_0), (\varepsilon, \varepsilon)\rangle \xrightarrow{C!S(0)} \langle (c_1, s_0), (0, \varepsilon)\rangle \xrightarrow{C!S(1)} \langle (c_0, s_0), (01, \varepsilon)\rangle \xrightarrow{C!S(0)} \langle (c_1, s_0), (010, \varepsilon)\rangle$.



Like traces being partially ordered representations of runs of asynchronous automata, runs of CA have a natural interpretation in terms of labeled pomsets, too. The pomsets associated with such runs are called *message sequence charts*, and represent in a diagrammatic way messages exchanged between processes.

3 Analyzing communicating automata

In spite of their simplicity, communicating automata are Turing-powerful, as it can be easily seen (by simulating e.g. Post tag systems). From the verification viewpoint this immediately implies that one needs to accept approximated or semi-algorithmic solutions.

Simple approximated solutions, like ignoring the order of messages in the channels or imposing a limit on their size, are of course too coarse. Acceleration methods using some finitary representation of possibly infinite sets of configurations (called symbolic representations), are a more powerful example of under-approximation. In the case of communicating automata, such symbolic representations are based on finite automata or some extended automata models with good algorithmic properties [4, 5, 7]. The general idea is to speed-up the naive enumeration of reachable configurations, by computing the result of loop iteration.

A nice example for over-approximating methods are lossy channel systems. Of course, such a model may be interesting in its own right, since it allows to model imperfect channels. Lossy channels are a particular instance of well-structured transition systems [13, 2]. In particular, questions like control-state reachability and termination are decidable [2, 14], albeit of non-primitive recursive complexity [36]. On the other hand, liveness properties or boundedness of lossy channels are undecidable [1, 27].

Whereas the above mentioned approaches emphasize symbolic representations of sets of (reachable) configurations, there is a complementary, language-oriented approach based on *partial orders*. The language-theoretical viewpoint emphasizes the (partially-ordered) executions, instead of the channel contents. This kind of event-based reasoning arises very naturally when communicating automata are viewed as sequential automata synchronizing over communication events. The main advantage it offers is that the synthesis problem can be stated in a natural way.

Undecidability of various questions about communicating automata has actually two sources: the first, obvious one, is the unboundedness of channels. The second, more subtle, comes up when the specification formalism (e.g. regular ones like LTL) is incompatible with the partially-ordered model. As a consequence, getting solutions for model-checking or synthesis requires both *channel restrictions* and *partial order specifications*.

A universally channel-bounded automaton is one where there is a uniform bound on the size of channels, over all reachable configurations. So a universally bounded automaton is just a finite state system. A much less restrictive notion is an existential channel-bound. Such a bound roughly means that any execution can be *rescheduled* in such a way that it can be executed with bounded channels. In particular, existential bounds admit channels of arbitrary size. A simple example illustrating the idea is a pair of processes, a producer and a consumer, where the producer keeps sending messages to the consumer, who is supposed to accept every message. Since there is no control on the relative speed of these two processes, there is no bound on the number of messages in transit. But for verifying many properties, like e.g. control-state reachability, it suffices to reason about schedulings where messages are consumed without delay, i.e. where executions can be scheduled with a channel of size one.

The main result obtained in this setting is a solution for closed synthesis, that can be stated as a Kleene-Büchi theorem about communicating automata with channel bounds [21, 18]. A main ingredient of these constructions is the link between automata with channel bounds and trace languages and in particular, Zielonka's construction of asynchronous (trace) automata. Model-checking existentially bounded automata w.r.t. partial order specifications like MSO [24], closed regular specifications [20] or PDL [6], is also decidable.

Several promising, recent research directions can be mentioned. One of them is motivated by the need of analyzing distributed recursive programs, and aims at identifying reasonable, tractable subclasses of communicating automata extended by additional capabilities for the single processes, like for instance pushdown storage [3, 39, 22]. A second, quite challenging perspective for future work is the general synthesis problem for communicating systems. This problem can be stated in many different ways, depending on the degree of completeness of the specification (specifications may e.g. talk only about external messages). However, one probably needs first a solution for the problem described in the next section, before working out a general solution for synthesizing or controlling communicating automata.

4 Distributed control for asynchronous automata

In the simplest case, the synthesis problem asks to find a model for a given specification, so it is just a satisfiability problem, where one is given some formalism for the specification (e.g. logics) and one for the model (e.g. finite automata). In a more refined version one is also given a system (or plant, usually an automaton) and is asked to find a controller such that the controlled system satisfies the given specification. The control consists in forbidding some actions of the plant, but not every action can be forbidden, and the control has also to ensure that the system does not block completely.

Synthesis was first considered in a synchronous (hardware) setting by Church [9]. In his model, the specification is a relation between input variables, which are controlled by the environment, and output variables, controlled by the (centralized) system. Church's problem stimulated a fruitful research direction on 2-person, zero-sum infinitary games, starting with the fundamental results of [8, 34, 35] (see also [37, 38] for recent surveys).

Distributed controller synthesis is a more recent research topic, that was initiated by Pnueli and Rosner [33], who show that only very restricted architectures admit a decidable synthesis problem. Undecidability of distributed synthesis follows already from the work of Peterson and Reif on "multiple-person alternating machines" [32].

Various versions of distributed synthesis appear in the literature. One important distinction is to be made between synchronous and asynchronous systems, respectively. In the synchronous case, processes execute a step at each (global) clock tick, whereas in the asynchronous case they are decoupled. Another distinction is how much information is allowed to be exchanged between processes. At least two different classes of models were studied here. In the model considered by [33, 23, 12, 16], a distributed system is given by an architecture describing (synchronous) channels between processes, and the information conveyed via the channels between processes, is finite. In the model studied in [15, 26, 31], the distributed system is an asynchronous automaton (and the controller is also required to be such an automaton). Here, the information exchanged between processes corresponds to the causal past of events, therefore it is unbounded.

Decidability for synchronous synthesis basically requires a pipeline architecture where information flows in a single direction (see [33, 23, 25, 29, 12, 16] for various refinements). To state it informally, the reasons for undecidability are either global specifications or "information forks", like the case where two independent processes can be "observed" by a third one.

Compared with the synchronous case, our understanding of asynchronous controller synthesis is still unsatisfactory. For instance, it is open whether this problem is decidable! Two decidability results are known in this setting. The first one [15] was obtained by restricting the (in)dependencies between letters of the input alphabet. The second paper [26] shows decidability by restricting the plant: roughly speaking, the restriction requires that if two processes do not synchronize during a long amount of time, then they won't synchronize ever again. The proof of [26] goes beyond the controller synthesis problem, by coding

it into monadic second-order theory of event structures and showing that this theory is decidable when the criterion on the asynchronous automaton holds.

References

1. P. A. Abdulla, K. Cerans, B. Jonsson, and Y.-K. Tsay. General decidability theorems for infinite state systems. In *LICS'96*, pages 313–323. IEEE Computer Society, 1996.
2. P. A. Abdulla and B. Jonsson. Verifying programs with unreliable channels. *Inform. and Comput.*, 127(2):91–101, 1996.
3. M. F. Atig, A. Bouajjani, and T. Touili. On the reachability analysis of acyclic networks of pushdown systems. In *CONCUR'08*, number 5201 in LNCS, pages 356–371. Springer, 2008.
4. B. Boigelot and P. Godefroid. Symbolic verification of communication protocols with infinite state spaces using QDDs. In *CAV'96*, volume 1102 of *LNCS*, pages 1–12. Springer, 1996.
5. B. Boigelot, P. Godefroid, B. Willems, and P. Wolper. The power of QDDs. In *SAS'97*, number 1302 in LNCS, pages 172–186. Springer, 1997.
6. B. Bollig, D. Kuske, and I. Meinecke. Propositional Dynamic Logic for message-passing systems. *Logical Methods in Computer Science*, To appear 2010.
7. A. Bouajjani and P. Habermehl. Symbolic reachability analysis of FIFO-channel systems with nonregular sets of configurations. *Theor. Comp. Science*, 221(1-2):211–250, 1999.
8. J. Büchi and L. Landweber. Definability in the monadic second order theory of successor. *J. of Symb. Logic*, 34(2):166–170, 1969.
9. A. Church. Applications of recursive arithmetic to the problem of circuit synthesis. In *Summaries of the Summer Institute of Symbolic Logic*, volume 1, pages 3–50. Cornell Univ., 1957.
10. R. Cori, Y. Métivier, and W. Zielonka. Asynchronous mappings and asynchronous cellular automata. *Inform. and Comput.*, 106:159–202, 1993.
11. V. Diekert and G. Rozenberg, editors. *The Book of Traces*. World Scientific, Singapore, 1995.
12. B. Finkbeiner and S. Schewe. Uniform distributed synthesis. In *LICS'05*, pages 321–330. IEEE Computer Society Press, 2005.
13. A. Finkel. A generalization of the procedure of Karp and Miller to well structured transition systems. In *ICALP'87*, number 267 in LNCS, pages 499–508. Springer, 1987.
14. A. Finkel and Ph. Schnoebelen. Well-structured transition systems everywhere! *Theor. Comp. Science*, 256(1-2):63–92, 2001.
15. P. Gastin, B. Lerman, and M. Zeitoun. Distributed games with causal memory are decidable for series-parallel systems. In *FSTTCS'04*, number 3328 in LNCS, pages 275–286. Springer, 2004.
16. P. Gastin, N. Sznajder, and M. Zeitoun. Distributed synthesis for well-connected architectures. *Formal Methods in System Design*, 34(3):215–237, 2009.
17. B. Genest, H. Gimbert, A. Muscholl, and I. Walukiewicz. Optimal Zielonka-type construction of deterministic asynchronous automata. In *ICALP'10*, LNCS. Springer, 2010.
18. B. Genest, D. Kuske, and A. Muscholl. A Kleene theorem and model checking algorithms for existentially bounded communicating automata. *Inform. and Comput.*, 204(6):920–956, 2006.

19. B. Genest and A. Muscholl. Constructing exponential-size deterministic Zielonka automata. In *ICALP'06*, number 4052 in LNCS, pages 565–576. Springer, 2006.
20. B. Genest, A. Muscholl, H. Seidl, and M. Zeitoun. Infinite-state high-level MSCs: Model-checking and realizability. *J. Comput. Syst. Sci.*, 72(4):617–647, 2006.
21. J. G. Henriksen, M. Mukund, K. N. Kumar, M. Sohoni, and P. Thiagarajan. A theory of regular MSC languages. *Inform. and Comput.*, 202(1):1–38, 2005.
22. A. Heußner, J. Leroux, A. Muscholl, and G. Sutre. Reachability analysis of communicating pushdown systems. In *FoSSaCS'10*, number 6014 in LNCS, pages 267–281. Springer, 2010.
23. O. Kupferman and M. Vardi. Synthesizing distributed systems. In *LICS'01*. IEEE Computer Society Press, 2001.
24. P. Madhusudan and B. Meenakshi. Beyond message sequence graphs. In *FSTTCS'01*, number 2245 in LNCS, pages 256–267. Springer, 2001.
25. P. Madhusudan and P. Thiagarajan. Distributed control and synthesis for local specifications. In *ICALP'01*, number 2076 in LNCS, pages 396–407. Springer, 2001.
26. P. Madhusudan, P. S. Thiagarajan, and S. Yang. The MSO theory of connectedly communicating processes. In *FSTTCS'05*, number 3821 in LNCS, pages 201–212. Springer, 2005.
27. R. Mayr. Undecidable problems in unreliable computations. *Theor. Comp. Science*, 297(1-3):337–354, 2003.
28. A. Mazurkiewicz. Concurrent program schemes and their interpretations. DAIMI Rep. PB 78, Aarhus University, Aarhus, 1977.
29. S. Mohalik and I. Walukiewicz. Distributed games. In *FSTTCS'03*, number 2914 in LNCS, pages 338–351. Springer, 2003.
30. 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.
31. A. Muscholl, I. Walukiewicz, and M. Zeitoun. A look at the control of asynchronous automata. In *Perspectives in Concurrency Theory*. IARCS-Universities, Universities Press, 2009.
32. G. L. Peterson and J. H. Reif. Multi-person alternation. In *FOCS'79*, pages 348–363. IEEE Computer Society Press, 1979.
33. A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *FOCS'90*, pages 746–757. IEEE Computer Society Press, 1990.
34. M. Rabin. Decidability of second-order theories and automata on infinite trees. *Trans. Amer. Math. Soc.*, 141:1–35, 1969.
35. M. O. Rabin. *Automata on Infinite Objects and Church's Problem*. American Mathematical Society, Providence, RI, 1972.
36. Ph. Schnoebelen. Verifying lossy channel systems has nonprimitive recursive complexity. *Inform. Proc. Lett.*, 83(5):251–261, 2002.
37. W. Thomas. On the synthesis of strategies in infinite games. In *STACS'95*, number 900 in LNCS, pages 1–13. Springer, 1995.
38. W. Thomas. Church's problem and a tour through automata theory. In *Pillar's of Computer Science*, number 4800 in LNCS, pages 635–655. Springer, 2008.
39. S. L. Torre, P. Madhusudan, and G. Parlato. Context-bounded analysis of concurrent queue systems. In *TACAS'08*, number 4963 in LNCS, pages 299–314. Springer, 2008.
40. W. Zielonka. Notes on finite asynchronous automata. *RAIRO - Informatique Théorique et Applications*, 21:99–135, 1987.