Distributed Synthesis for Acyclic Architectures

Anca Muscholl[1] and Igor Walukiewicz[2]

1 **Université de Bordeaux, France**
2 **CNRS, Université de Bordeaux, France**

Abstract

The distributed synthesis problem is about constructing correct distributed systems, i.e., systems that satisfy a given specification. We consider a slightly more general problem of distributed control, where the goal is to restrict the behavior of a given distributed system in order to satisfy the specification. Our systems are finite state machines that communicate via rendez-vous (Zielonka automata). We show decidability of the synthesis problem for all ω-regular local specifications, under the restriction that the communication graph of the system is acyclic. This result extends a previous decidability result for a restricted form of local reachability specifications.

1998 ACM Subject Classification D.2.4 [Software Engineering] Software/Program Verification, F.3.1 [Logics and Meanings of Programs] Specifying and Verifying and Reasoning about Programs

Keywords and phrases Distributed synthesis, distributed control, causal memory

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2014.639

1 Introduction

Synthesizing distributed systems from specifications is an attractive objective, since distributed systems are notoriously difficult to get right. Unfortunately, there are very few known decidable frameworks for distributed synthesis. We study a framework for synthesis of open systems that is based on rendez-vous communication and causal memory. In particular, causal memory implies that although the specification can say when a communication takes place, it cannot limit the information that is transmitted during communication. This choice is both realistic and avoids some pathological reasons for undecidability. We show a decidability result for acyclic communication graphs and local ω-regular specifications.

Instead of synthesis we actually work in the more general framework of distributed control. Our setting is a direct adaptation of the supervisory control framework of Ramadge and Wonham [17]. In this framework we are given a plant (a finite automaton) where some of the actions are uncontrollable, and a specification. The goal is to construct a controller (another finite automaton) such that its product with the plant satisfies the specification. The controller is not allowed to block uncontrollable actions, in other words, in every state there is a transition on every uncontrollable action. The controlled plant has less behaviours, resulting from restricting controllable actions of the plant. In our case the formulation is exactly the same, but we consider Zielonka automata instead of finite automata, as plants and controllers. Considering parallel devices, such as Zielonka automata, in the standard definition of control gives an elegant formulation of the distributed control problem.

Zielonka automata [19, 14] are by now a well-established model of distributed computation. Such a device is an asynchronous product of finite-state processes synchronizing on shared actions. Asynchronicity means that processes can progress at different speed. The synchronization on shared actions allows the synchronizing processes to exchange information, in particular the controllers can transfer control information with each synchronization. This

34th Int'l Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2014).
Editors: Venkatesh Raman and S. P. Suresh; pp. 639–651

Leibniz International Proceedings in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

model can encode some common synchronization primitives available in modern multi-core processors for implementing concurrent data structures, like compare-and-swap.

We show decidability of the control problem for Zielonka automata where the communication graph is acyclic: a process can communicate (synchronize) with its parent and its children. Our specifications are conjunctions of ω-regular specifications, one for each of the component processes. We allow uncontrollable communication actions – the only restriction is that all communication actions must be binary. Uncontrollable communications add expressive power to the setting, for instance it is possible to model asymmetric situations where communication can be refused by one partner, but not by the other one.

Our result extends [7] that showed decidability for a restricted form of local reachability objectives (blocking final states). We still get the same complexity as in [7]: non-elementary in general, and EXPTIME for architectures of depth 1. The extension to all ω-regular objectives allows to express fairness constraints but at the same time introduces important technical obstacles. Indeed, for our construction to work it is essential that we enrich the framework by uncontrollable synchronization actions. This makes a separation into controllable and uncontrollable states impossible. In consequence, we are led to abandon the game metaphor, to invent new arguments, and to design a new proof structure.

Most research on distributed synthesis and control has been done in the setting proposed by Pnueli and Rosner [16]. This setting is also based on shared-variable communication, however the controllers there are not free to pass additional information between processes. So the latter model leads to partial information games, and decidability of synthesis holds only for variants of pipelines [10, 11, 4]. While specifications leading to undecidability are very artificial, no elegant solution to eliminate them exists at present. The synthesis setting is investigated in [11] for local specifications, meaning that each process has its own, linear-time specification. More relaxed variants of synthesis have been proposed, where the specification does not fully describe the communication of the synthesized system. One approach consists in adding communication in order to combine local knowledge, as proposed for example in [8]. Another approach is to use specifications only for describing external communication, as done in [6] on strongly connected architectures where processes communicate via signals.

Apart from [7], three related decidability results for synthesis with causal memory are known. The first one [5] restricts the alphabet of actions: control with reachability condition is decidable for co-graph alphabets. This restriction excludes client-server architectures, which are captured by our setting. The second result [12] shows decidability by restricting the plant: roughly speaking, the restriction says that every process can have only bounded missing knowledge about the other processes, unless they diverge (see also [15] that shows a doubly exponential upper bound). The proof of [12] 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 plant holds. Unfortunately, very simple plants have a decidable control problem and at the same time an undecidable MSO-theory. Safety games on Petri nets are considered in [3], where decidability in EXPTIME is shown in the case when there is a single environment player. Note that the same complexity is achieved in our model with client-server architectures and no restriction on the environment. Game semantics and asynchronous games played on event structures are introduced in [13]. Such games are investigated in [9] from a game-theoretical viewpoint, showing a Borel determinacy result under some restrictions.

Overview. In Section 2 we state and motivate our control problem. In Section 3 we give the main lines of the proof. A complete version of the paper is available at hal-00946554.

2 Control for Zielonka automata

In this section we introduce our control problem for Zielonka automata, adapting the definition of supervisory control [17] to our model.

A Zielonka automaton [19, 14] is a simple distributed finite-state device. Such an automaton is a parallel composition of several finite automata, called *processes*, synchronizing on shared actions. There is no global clock, so between two synchronizations, two processes can do a different number of actions. Because of this, Zielonka automata are also called asynchronous automata.

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 $dom : \Sigma \to (2^{\mathbb{P}} \setminus \emptyset)$ 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. Actions from $\Sigma_p = \{a \in \Sigma \mid p \in dom(a)\}$ are called *p-actions*. We write $\Sigma_p^{loc} = \{a \mid dom(a) = \{p\}\}$ for the set of *local* actions of p.

A (deterministic) *Zielonka automaton* $\mathcal{A} = \langle \{S_p\}_{p\in\mathbb{P}}, s_{in}, \{\delta_a\}_{a\in\Sigma}\rangle$ 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 $\delta_a : \prod_{p\in dom(a)} S_p \stackrel{\cdot}{\to} \prod_{p\in dom(a)} S_p$ on tuples of states of processes in $dom(a)$.

▶ **Example 1.** Boolean multi-threaded programs with shared variables can be modelled as Zielonka automata. As an example we describe the translation for the *compare-and-swap* (CAS) instruction. This instruction has 3 parameters: CAS(x: variable; *old*, *new*: int). Its effect is to return the value of x and at the same time set the value of x to *new*, but only if the previous value of x was equal to *old*. The compare-and-swap operation is a widely used primitive in implementations of concurrent data structures, and has hardware support in most contemporary multiprocessor architectures.

Suppose that we have a thread t, and a shared variable x that is accessed by a CAS operation in t via $y := \mathsf{CAS}_x(i,k)$. So y is a local variable of t. In the Zielonka automaton we will have one process modelling thread t, and one process for variable x. The states of t will be valuations of local variables. The states of x will be the values x can take. The CAS instruction above becomes a synchronization action. We have the following two types of transitions on this action:

Notice that in state s', we have $y = i$, whereas in s'', we have $y = j$.

For convenience, we abbreviate a tuple $(s_p)_{p\in P}$ of local states by s_P, where $P \subseteq \mathbb{P}$. We also talk about S_p as the set of *p-states*.

A Zielonka automaton can be seen as a sequential automaton with the state set $S = \prod_{p\in\mathbb{P}} S_p$ and transitions $s \xrightarrow{a} s'$ if $(s_{dom(a)}, s'_{dom(a)}) \in \delta_a$, and $s_{\mathbb{P}\setminus dom(a)} = s'_{\mathbb{P}\setminus dom(a)}$. So the states of this automaton are the tuples of states of the processes of the Zielonka automaton. For a process p we will talk about the *p-component* of the state. A run of $\mathcal{A}$ is a finite or infinite sequence of transitions starting in s_{in}. Since the automaton is deterministic, a run is determined by the sequence of labels of the transitions. We will write $run(u)$ for the run determined by the sequence $u \in \Sigma^\infty$. Observe that $run(u)$ may be undefined since the transition function of $\mathcal{A}$ is partial. We will also talk about the projection of the run on component p, denoted $run_p(u)$, that is the projection on component p of the subsequence of

the run containing the transitions involving p. We will assume that every local state of $\mathcal{A}$ occurs in some run. By $dom(u)$ we denote the union of $dom(a)$, for all $a \in \Sigma$ occurring in u.

We will be interested in maximal runs of Zielonka automata. For parallel devices the notion of a maximal run is not that evident, as one may want to impose some fairness conditions. We settle here for a minimal sensible fairness requirement. It says that a run is maximal if processes who have only finitely many actions in the run cannot perform any additional action.

▶ **Definition 2** (Maximal run)**.** For a word $w \in \Sigma^\infty$ such that $run(w)$ is defined, we say that $run(w)$ is *maximal* if there is no decomposition $w = uv$, and no action $a \in \Sigma$ such that $dom(v) \cap dom(a) = \emptyset$ and $run(uav)$ is defined.

Automata can be equipped with a *correctness condition*. We prefer to talk about correctness condition rather than acceptance condition since we will be interested in the set of runs of an automaton rather than in the set of words it accepts. We will consider local regular correctness conditions: every process has its own correctness condition $Corr_p$. A run of $\mathcal{A}$ is *correct* if for every process p, the projection of the run on the transitions of $\mathcal{A}_p$ is in $Corr_p$. Condition $Corr_p$ is specified by a set $T_p \subseteq S_p$ of terminal states and an ω-regular set $\Omega_p \subseteq (S_p \times \Sigma_p \times S_p)^\omega$. A sequence $(s_p^0, a_0, s_p^1)(s_p^1, a_1, s_p^2)\dots$ satisfies $Corr_p$ if either: (i) it is finite and ends with a state from T_p, or (ii) it is infinite and belongs to Ω_p. At this stage the set of terminal states T_p may look unnecessary, but it will simplify our constructions later.

Finally, we will need the notion of *synchronized product* $\mathcal{A} \times \mathcal{C}$ of two Zielonka automata. For $\mathcal{A} = \langle \{S_p\}_{p\in\mathbb{P}}, s_{in}, \{\delta_a^A\}_{a\in\Sigma}\rangle$ and $\mathcal{C} = \langle \{C_p\}_{p\in\mathbb{P}}, c_{in}, \{\delta_a^C\}_{a\in\Sigma}\rangle$ let $\mathcal{A} \times \mathcal{C} = \langle \{S_p \times C_p\}_{p\in\mathbb{P}}, (s_{in}, c_{in}), \{\delta_a^\times\}_{a\in\Sigma}\}\rangle$ where there is a transition from $(s_{dom(a)}, c_{dom(a)})$ to $(s'_{dom(a)}, c'_{dom(a)})$ in $\delta_a^\times$ iff $(s_{dom(a)}, s'_{dom(a)}) \in \delta_a^A$ and $(c_{dom(a)}, c'_{dom(a)}) \in \delta_a^C$.

To define the control problem for Zielonka automata we fix a distributed alphabet $\langle \mathbb{P}, dom : \Sigma \to (2^\mathbb{P} \setminus \emptyset)\rangle$. We partition Σ into the set of *system actions* Σ^{sys} and *environment actions* Σ^{env}. Below we will introduce the notion of controller, and require that it does not block environment actions. For this reason we speak about *controllable/uncontrollable* actions when referring to system/environment actions. We impose three simplifying assumptions: (1) All actions are at most binary ($|dom(a)| \leq 2$ for every $a \in \Sigma$); (2) every process has some controllable action; (3) all controllable actions are local. Among the three conditions only the first one is indeed a restriction of our setting. The other two are not true limitations, in particular controllable shared actions can be simulated by a local controllable choice, followed by non-controllable local or shared actions (see full version of the paper)

▶ **Definition 3** (Controller, Correct Controller)**.** A *controller* is a Zielonka automaton that cannot block environment (uncontrollable) actions. In other words, from every state every environment action is possible: for every $b \in \Sigma^{env}$, δ_b is a total function. We say that a controller $\mathcal{C}$ *is correct for* a plant $\mathcal{A}$ if all maximal runs of $\mathcal{A} \times \mathcal{C}$ satisfy the correctness condition of $\mathcal{A}$.

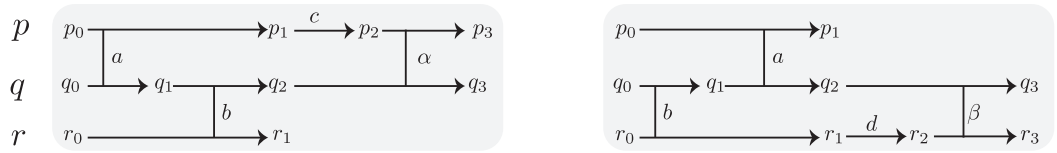
Recall that an action is possible in $\mathcal{A} \times \mathcal{C}$ iff it is possible in both $\mathcal{A}$ and $\mathcal{C}$. By the above definition, environment actions are always possible in $\mathcal{C}$. The major difference between the controlled system $\mathcal{A} \times \mathcal{C}$ and and $\mathcal{A}$ is that the states of $\mathcal{A} \times \mathcal{C}$ carry the additional information computed by $\mathcal{C}$, and that $\mathcal{A}\times\mathcal{C}$ may have less behaviors, resulting from disallowing controllable actions by $\mathcal{C}$.

The correctness of $\mathcal{C}$ means that all the runs of $\mathcal{A}$ that are *allowed* by $\mathcal{C}$ are correct. In particular, $\mathcal{C}$ does not have a correctness condition by itself. Considering only maximal runs of $\mathcal{A} \times \mathcal{C}$ imposes some minimal fairness conditions: for example it implies that if a process can do a local action almost always, then it will eventually do some action.

▶ **Definition 4** (Control problem). Given a distributed alphabet $\langle \mathbb{P}, dom : \Sigma \to (2^{\mathbb{P}} \setminus \emptyset)\rangle$ together with a partition of actions $(\Sigma^{sys}, \Sigma^{env})$, and given a Zielonka automaton $\mathcal{A}$ over this alphabet, find a controller $\mathcal{C}$ over the same alphabet such that $\mathcal{C}$ is correct for $\mathcal{A}$.

The important point in our definition is that the controller has the same distributed alphabet as the automaton it controls, in other words the controller is not allowed to introduce additional synchronizations between processes.

▶ **Example 5.** We give an example showing how causal memory works and helps to construct controllers. Consider an automaton $\mathcal{A}$ with 3 processes: p, q, r. We would like to control it so that the only two possible runs of $\mathcal{A}$ are the following:



So p and q should synchronize on α when action a happened before b, otherwise q and r should synchronize on β. Communication actions are uncontrollable, but the transitions of $\mathcal{A}$ are such that there are local controllable actions c and d that enable communication on α and β respectively. So the controller should block either c or d depending on the order between a and b. The transitions of $\mathcal{A}$ are as follows:

$$\begin{array}{llll} \delta_a(p_0,q_0)=(p_1,q_1) & \delta_a(p_0,q_1)=(p_1,q_2) & \delta_b(q_0,r_0)=(q_1,r_1) & \delta_b(q_1,r_0)=(q_2,r_1) \\ \delta_c(p_1)=p_2 & \delta_\alpha(p_2,q_2)=(p_3,q_3) & \delta_d(r_1)=r_2 & \delta_\beta(q_2,r_2)=(q_3,r_3) \end{array}$$

These transitions allow the two behaviors depicted above but also two unwanted ones, as say, when a happens before b and then we see β. Note that the specification of desired behaviors is a local condition on q. So by encoding some information in states of q this condition can be expressed by a set of terminal states T_q. We will not do this for readability.

The controller $\mathcal{C}$ for $\mathcal{A}$ will mimic the structure of $\mathcal{A}$: for every state of $\mathcal{A}$ there will be in $\mathcal{C}$ a state with over-line. So, for example, the states of q in $\mathcal{C}$ will be $\overline{q}_0, \ldots, \overline{q}_3$. Moreover $\mathcal{C}$ will have two new states $\underline{p_1}$ and $\underline{r_1}$. The transitions will be

$$\begin{array}{llll} \delta_a(\overline{p}_0,\overline{q}_0)=(\overline{p}_1,\overline{q}_1) & \delta_a(\overline{p}_0,\overline{q}_1)=(\underline{p_1},\overline{q}_2) & \delta_b(\overline{q}_0,\overline{r}_0)=(\overline{q}_1,\overline{r}_1) & \delta_b(\overline{q}_1,\overline{r}_0)=(\overline{q}_2,\underline{r_1}) \\ \delta_c(\overline{p}_1)=\overline{p}_2 & \delta_c(\underline{p_1})=\bot & \delta_d(\overline{r}_1)=\overline{r}_2 & \delta_d(\underline{r_1})=\bot \\ \delta_\alpha(\overline{p}_2,\overline{q}_2)=(\overline{p}_3,\overline{q}_3) & & \delta_\beta(\overline{q}_2,\overline{r}_2)=(\overline{q}_3,\overline{r}_3) & \end{array}$$

Observe that c is blocked in $\underline{p_1}$, and so is d from $\underline{r_1}$. It is easy to verify that the runs of $\mathcal{A}\times\mathcal{C}$ are as required, so $\mathcal{C}$ is a correct controller for $\mathcal{A}$. (Actually the definition of a controller forces us to make transitions of $\mathcal{C}$ total on uncontrollable actions. We can do it in arbitrary way as this will not add new behaviors to $\mathcal{A}\times\mathcal{C}$.)

This example shows several phenomena. The states of $\mathcal{C}$ are the states of $\mathcal{A}$ coupled with some additional information. We formalize this later under a notion of covering controller. We could also see above a case where a communication is decided by one of the parties. Process p, thanks to a local action, can decide if it wants to communicate via α, but process q has to accept α always. This shows the flexibility given by uncontrollable communication actions. Finally, we could see information passing during communication. In $\mathcal{C}$ process q passes to p and r information about its local state (cf. transitions on a and b).

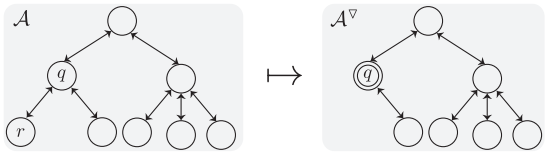


Figure 1 Eliminating process r: r is glued with q.

3 Decidability for acyclic architectures

In this section we present the main result of the paper. We show the decidability of the control problem for Zielonka automata with acyclic architecture. A *communication architecture* of a distributed alphabet is a graph where nodes are processes and edges link processes that have common actions. An *acyclic architecture* is one whose communication graph is acyclic. For example, the communication graph of the alphabet from the example on page 643 is a tree with root q and two successors, p and r.

▶ **Theorem 6.** *The control problem for Zielonka automata over distributed alphabets with acyclic architecture is decidable. If a controller exists, then it can be effectively constructed.*

The remaining of this section is devoted to the outline of the proof of Theorem 6. This proof works by induction on the number $|\mathbb{P}|$ of processes in the automaton. A Zielonka automaton over a single process is just a finite automaton, and the control problem is then just the standard control problem as considered by Ramadge and Wonham but extended to all ω-regular conditions [1]. If there are several processes that do not communicate, then we can solve the problem for each process separately.

Otherwise we choose a leaf process r and its parent q and construct a new plant $\mathcal{A}^\triangledown$ over $\mathbb{P} \setminus \{r\}$. We will show that the control problem for $\mathcal{A}$ has a solution iff the one for $\mathcal{A}^\triangledown$ does. Moreover, for every solution for $\mathcal{A}^\triangledown$ we will be able to construct a solution for $\mathcal{A}$.

For the rest of this section let us fix the distributed alphabet $\langle \mathbb{P}, dom : \Sigma \to (2^{\mathbb{P}} \setminus \emptyset)\rangle$, the leaf process r and its parent q, and a Zielonka automaton with a correctness condition $\mathcal{A} = \langle \{S_p\}_{p\in\mathbb{P}}, s_{in}, \{\delta_a\}_{a\in\Sigma}, \{Corr_p\}_{p\in\mathbb{P}}\rangle$.

The first step in proving Theorem 6 is to simplify the problem. First, we can restrict to controllers of a special form called covering controllers. Next, we show that the component of $\mathcal{A}$ to be eliminated, that is $\mathcal{A}_r$, can be assumed to have a particular property (r-short). After these preparatory results we will be able to present the reduction of $\mathcal{A}$ to $\mathcal{A}^\triangledown$ (Theorem 13).

3.1 Covering controllers and r-short automata

In this subsection we introduce the notions of covering controllers and r-short automata. We show that in the control problem we can restrict to covering controllers (Lemma 8), and that we can always transform a plant to an r-short one without affecting controllability (Theorem 10).

The notion of a covering controller will simplify the presentation because it will allow us to focus on the runs of the controller instead of a product of the plant and the controller. We have seen already a covering controller in Example 5.

▶ **Definition 7** (Covering controller)**.** Let $\mathcal{C}$ be a Zielonka automaton over the same alphabet as $\mathcal{A}$; let C_p be the set of states of process p in $\mathcal{C}$. Automaton $\mathcal{C}$ is a *covering controller* for $\mathcal{A}$ if there is a function $\pi : \{C_p\}_{p\in\mathbb{P}} \to \{S_p\}_{p\in\mathbb{P}}$, mapping each C_p to S_p and satisfying

two conditions: (i) if $c_{dom(b)} \xrightarrow{b} c'_{dom(b)}$ then $\pi(c_{dom(b)}) \xrightarrow{b} \pi(c'_{dom(b)})$; (ii) for every uncontrollable action a: if a is enabled from $\pi(c_{dom(a)})$ then it is also enabled from $c_{dom(a)}$.

▶ Remark. Strictly speaking, a covering controller $\mathcal{C}$ may not be a controller since we do not require that every uncontrollable action is enabled in every state, but only those actions that are enabled in $\mathcal{A}$. From $\mathcal{C}$ one can get a controller $\hat{\mathcal{C}}$ by adding self-loops for all missing uncontrollable transitions.

Notice that thanks to the projection π, a covering controller can inherit the correctness condition of $\mathcal{A}$. Therefore it is enough to look at the runs of $\mathcal{C}$ instead of $\mathcal{A} \times \mathcal{C}$:

▶ **Lemma 8.** *There is a correct controller for $\mathcal{A}$ if and only if there is a covering controller $\mathcal{C}$ for $\mathcal{A}$ such that all the maximal runs of $\mathcal{C}$ satisfy the inherited correctness condition.*

We will refer to a covering controller with the property that all its maximal runs satisfy the inherited correctness condition, as *correct covering controller*.

Now we will focus on making the automaton r-short.

▶ **Definition 9** (r-short)**.** An automaton $\mathcal{A}$ is *r-short* if there is a bound on the number of actions that r can perform without doing a communication with q.

▶ **Theorem 10.** *For every automaton $\mathcal{A}$, we can construct an r-short automaton $\mathcal{A}^{\circledS}$ such that there is a correct controller for $\mathcal{A}$ iff there is one for $\mathcal{A}^{\circledS}$.*

In the rest of this subsection we sketch the proof of this theorem (details are provided in the appendix). This theorem bears some resemblance with the fact that every parity game can be transformed into a finite game: when a loop is closed the winner is decided looking at the ranks on the loop. In order to use a similar construction we must ensure that there is always a controller that is in some sense memoryless on the component r (Lemma 12).

Observe that we can make two simplifying assumptions. First, we assume that the correctness condition on r is a parity condition. That is, it is given by a rank function $\Omega_r : S_r \to \mathbb{N}$ and the set of terminal states T_r. We can assume this since every regular language of infinite sequences can be recognized by a deterministic parity automaton. The second simplification is to assume that the automaton $\mathcal{A}$ is *r-aware* with respect to the parity condition on r. This means that from the state of r one can read the biggest rank that process r has seen the last communication of r with q. It is easy to transform an automaton to an r-aware one.

Given $\mathcal{A}$ we define an r-short automaton $\mathcal{A}^{\circledS}$. All its components will be the same but for the component r. The states $S_r^{\circledS}$ of r will be sequences $w \in S_r^+$ of states of $\mathcal{A}_r$ without repetitions, plus two new states $\top, \bot$. For a local transition $s'_r \xrightarrow{b} s''_r$ in $\mathcal{A}_r$ we have in $\mathcal{A}_r^{\circledS}$ transitions:

$ws'_r \xrightarrow{b} ws'_r s''_r$	if $ws'_r s''_r$ a sequence without repetitions
$ws'_r \xrightarrow{b} \top$	if s''_r appears in w and the resulting loop is even
$ws'_r \xrightarrow{b} \bot$	if s''_r appears in w and the resulting loop is odd

$(s_q, ws'_r) \xrightarrow{b} (s'_q, s''_r)$ if $(s_q, s'_r) \xrightarrow{b} (s'_q, s''_r)$ in $\mathcal{A}$

To the right we have displayed communication transitions between q and r. Notice that w disappears in communication transitions. The parity condition for $\mathcal{A}^{\circledS}$ is also rather straightforward: it is the same for the components other than r, and for $\mathcal{A}_r$ we have $\Omega^{\circledS}(ws_r) = \Omega(s_r)$, and $T_r^{\circledS} = \{\top\} \cup \{ws_r \mid s_r \in T_r\}$.

It is clear that the length of every sequence of local actions of process r in $\mathcal{A}^{\circledS}$ is bounded by the number of states of $\mathcal{A}_r$.

For the correctness of the reduction we first show how to construct a correct controller $\mathcal{C}$ for $\mathcal{A}$ from a correct controller controller $\mathcal{C}^{\circledS}$ for $\mathcal{A}^{\circledS}$. The idea is that $\mathcal{C}$ simulates $\mathcal{C}^{\circledS}$ but when the execution of the latter gets to $\top$, then the execution detects a loop, so $\mathcal{C}$ can restart $\mathcal{C}^{\circledS}$ from the prefix of the sequence obtained by removing the loop (cf. the definition of $\mathcal{C}^{\circledS}$).

For the other direction, given a correct covering controller $\mathcal{C}$ for $\mathcal{A}$ we show that $\mathcal{C}$ is also a correct controller for $\mathcal{A}^{\circledS}$ provided that $\mathcal{C}$ is r-memoryless as defined below. Then we can conclude by Lemma 12 below, allowing us to assume that $\mathcal{C}$ is memoryless.

Recall that if $\mathcal{C}$ is a covering controller for $\mathcal{A}$ (cf. Definition 7) then there is a function $\pi : \{C_p\}_{p\in\mathbb{P}} \to \{S_p\}_{p\in\mathbb{P}}$, mapping each C_p to S_p and respecting the transition relation: if $c_{dom(b)} \xrightarrow{b} c'_{dom(b)}$ then $\pi(c_{dom(b)}) \xrightarrow{b} \pi(c'_{dom(b)})$.

▶ **Definition 11** (r-memoryless controller)**.** A covering controller $\mathcal{C}$ for $\mathcal{A}$ is *r-memoryless* when for every pair of states $c_r \neq c'_r$ of $\mathcal{C}_r$: if there is a path on local r-actions from c_r to c'_r then $\pi(c_r) \neq \pi(c'_r)$.

Intuitively, a controller can be seen as a strategy, and r-memoryless means that it does not allow the controlled automaton to go twice through the same r-state between two consecutive communication actions of r and q.

▶ **Lemma 12.** *Fix an r-aware automaton $\mathcal{A}$ with a parity correctness condition for process r. If there is a correct controller for $\mathcal{A}$ then there is also one that is covering and r-memoryless.*

The proof of Lemma 12 uses the notion of signatures [18, 2], that is classical in 2-player parity games, for defining a r-memoryless controller $\mathcal{C}^m$ from $\mathcal{C}$. The idea is to use representative states of C_r, defined in each strongly connected component according to a given signature and covering function π.

3.2 The reduced automaton $\mathcal{A}^\nabla$

Equipped with the notions of covering controller and r-short strategy we can now present the construction of the reduced automaton $\mathcal{A}^\nabla$. We suppose that $\mathcal{A} = \langle \{S_p\}_{p\in\mathbb{P}^\nabla}, s_{in}, \{\delta_a\}_{a\in\Sigma}\rangle$ is r-short and we define the reduced automaton $\mathcal{A}^\nabla$ that results by eliminating process r (cf. Figure 1). Let $\mathbb{P}^\nabla = \mathbb{P} \setminus \{r\}$. We construct $\mathcal{A}^\nabla = \langle \{S_p^\nabla\}_{p\in\mathbb{P}^\nabla}, s_{in}^\nabla, \{\delta_a^\nabla\}_{a\in\Sigma^\nabla}\rangle$ where the components are defined below.

All the processes $p \neq q$ of $\mathcal{A}^\nabla$ will be the same as in $\mathcal{A}$. This means: $S_p^\nabla = S_p$, and $\Sigma_p^\nabla = \Sigma_p$. Moreover, all transitions δ_a with $dom(a) \cap \{q, r\} = \emptyset$ are as in $\mathcal{A}$. Finally, in $\mathcal{A}^\nabla$ the correctness condition of $p \neq q$ is the same as in $\mathcal{A}$.

Before defining process q in $\mathcal{A}^\nabla$ let us introduce the notion of r-local strategy. An *r-local strategy from a state $s_r \in S_r$* is a partial function $f : (\Sigma_r^{loc})^* \to \Sigma_r^{sys}$ mapping sequences from Σ_r^{loc} to actions from Σ_r^{sys}, such that if $f(v) = a$ then $s_r \xrightarrow{va}$ in $\mathcal{A}_r$. Observe that since the automaton $\mathcal{A}$ is r-short, the domain of f is finite.

Given an r-local strategy f from s_r, a local action $a \in \Sigma_r^{loc}$ is *allowed by f* if $f(\epsilon) = a$, or a is uncontrollable. For a allowed by f we denote by $f_{|a}$ the r-local strategy defined by $f_{|a}(v) = f(av)$; this is a strategy from s'_r, where $s_r \xrightarrow{a} s'_r$.

The states of process q in $\mathcal{A}^\nabla$ are of one of the following types:

$$\langle s_q, s_r\rangle\,, \quad \langle s_q, s_r, f\rangle\,, \quad \langle s_q, a, s_r, f\rangle\,,$$

where $s_q \in S_q, s_r \in S_r$, f is a r-local strategy from s_r, and $a \in \Sigma_q^{loc}$. The new initial state for q is $\langle (s_{in})_q, (s_{in})_r\rangle$. Recall that that since $\mathcal{A}$ is r-short, any r-local strategy in $\mathcal{A}_r$ is necessarily finite, so S_q^∇ is a finite set. Recall also that controllable actions are local.

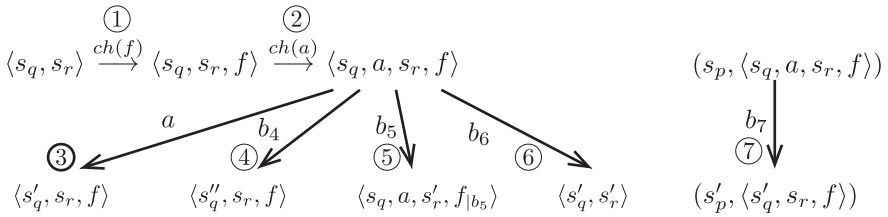


Figure 2 Transitions of $\mathcal{A}^\triangledown$.

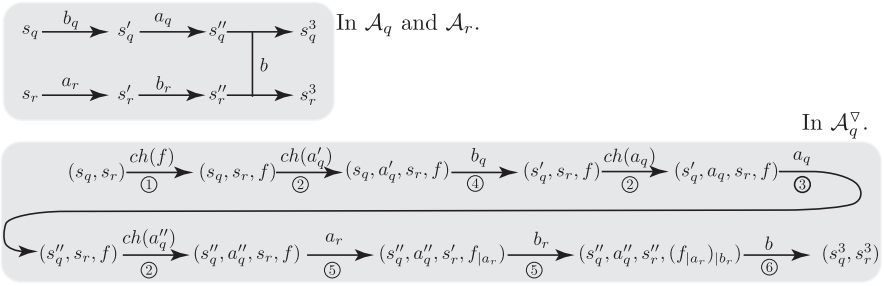


Figure 3 Simulation of $\mathcal{A}_q$ and $\mathcal{A}_r$ by $\mathcal{A}_q^\triangledown$.

The transitions of $\mathcal{A}_q^\triangledown$ are presented in Figure 2. Transition ① chooses an r-local strategy f. It is followed by transition ② that declares a controllable action $a \in \Sigma_q^{sys}$ that is enabled from s_q. Transition ③ executes the chosen action a; we require $s_q \xrightarrow{a} s'_q$ in $\mathcal{A}_q$. Transition ④ executes an uncontrollable local action $b_4 \in \Sigma_q^{env}$; provided $s_q \xrightarrow{b_4} s''_q$ in $\mathcal{A}_q$. Transition ⑤ executes a local action $b_5 \in \Sigma_r^{loc}$, provided that b_5 is allowed by f and $s_r \xrightarrow{b_5} s'_r$. Transition ⑥ simulates a synchronization on b_6 between q and r; provided $(s_q, s_r) \xrightarrow{b_6} (s'_q, s'_r)$ in $\mathcal{A}$. Finally, transition ⑦ simulates a synchronization between q and $p \neq r$. An example of a simulation of $\mathcal{A}_q$ and $\mathcal{A}_r$ by $\mathcal{A}_q^\triangledown$ is presented in Figure 3. The numbers below transitions refer to the corresponding cases from the definition.

To summarize, in $\Sigma_q^\triangledown$ we have all actions of Σ_r and Σ_q, but they become uncontrollable. All the new actions of process q in plant $\mathcal{A}^\triangledown$ are *controllable*:

- action $ch(f) \in \Sigma^{sys}$, for every local r-strategy f,
- action $ch(a)$, for every $a \in \Sigma_q^{sys}$.

The correctness condition for process q in $\mathcal{A}^\triangledown$ is:

1. The correct infinite runs of q in $\mathcal{A}^\triangledown$ are those that have the projection on transitions of $\mathcal{A}_q$ correct with respect to $Corr_q$, and either: *(i)* the projection on transitions of $\mathcal{A}_r$ is infinite and correct with respect to $Corr_r$; or *(ii)* the projection on transitions of $\mathcal{A}_r$ is finite and for f, s_r appearing in almost all states of q of the run we have that from s_r all sequences respecting strategy f end in a state from T_r.
2. $T_q^\triangledown$ contains states $\langle s_q, s_r, f\rangle$ such that $s_q \in T_q$, and $s_r \in T_r$.

Item 1(*ii*) in the definition above captures the case where q progresses alone till infinity and blocks r, even though r could reach a terminal state in a couple of moves. Clearly, item 1 can be expressed as an ω-regular condition. The definition of correctness condition is one of the principal places where the r-short assumption is used. Without this assumption we

would need to cope with the situation where we have an infinite execution of $\mathcal{A}_q$, and at the same time an infinite execution of $\mathcal{A}_r$ that do not communicate with each other. In this case $\mathcal{A}_q^\nabla$ would need to fairly simulate both executions in some way.

The reduction is rather delicate since in concurrent systems there are many different interactions that can happen. For example, we need to schedule actions of process q, using $ch(a)$ actions, before the actions of process r. The reason is the following. First, we need to make all r-actions uncontrollable, so that the environment could choose any play respecting the chosen r-local strategy. Now, if we allowed controllable q-actions to be eligible at the same time as r-actions, then the control strategy for automaton $\mathcal{A}^\nabla$ would be to propose nothing and force the environment to play the r-actions. This would allow the controller of $\mathcal{A}^\nabla$ to force the advancement of the simulation of r and get information that is impossible to obtain by the controller of $\mathcal{A}$.

Together with Theorem 10, the theorem below implies our main Theorem 6.

▶ **Theorem 13.** *For every r-short Zielonka automaton $\mathcal{A}$ and every local, ω-regular correctness condition: there is a correct covering controller for $\mathcal{A}$ iff there is a correct covering controller for $\mathcal{A}^\nabla$. The size of $\mathcal{A}_q^\nabla$ is polynomial in the size of $\mathcal{A}_q$ and exponential in the size of $\mathcal{A}_r$.*

3.3 Proof of Theorem 13

Given the space limit we can only give a sketch of one direction of the proof of Theorem 13. We consider the right-to-left direction. Given a correct controller $\mathcal{D}^\nabla$ for $\mathcal{A}^\nabla$, we show how to construct a correct controller $\mathcal{D}$ for $\mathcal{A}$. By Lemma 8 we can assume that $\mathcal{D}^\nabla$ is covering.

The components $\mathcal{D}_p$ for $p \neq q, r$ are the same as in $\mathcal{D}^\nabla$. So it remains to define $\mathcal{D}_q$ and $\mathcal{D}_r$. The states of $\mathcal{D}_q$ and $\mathcal{D}_r$ are obtained from states of $\mathcal{D}_q^\nabla$. We need only certain states of $\mathcal{D}_q^\nabla$, namely those d_q whose projection $\pi^\nabla(d_q)$ in $\mathcal{A}_q^\nabla$ has four components, we call them *true states* of $\mathcal{D}_q^\nabla$:

$$ts(\mathcal{D}_q^\nabla) = \{d_q \in \mathcal{D}_q^\nabla \mid \pi^\nabla(d_q) \text{ is of the form } (s_q, a, s_r, f)\}.$$

Figure 4 presents an execution of $\mathcal{A}^\nabla$ controlled by $\mathcal{D}^\nabla$. We can see that d_2 is a true state, and d_3 is not.

The set of states of $\mathcal{D}_q$ is just $ts(\mathcal{D}_q^\nabla)$, while the states of $\mathcal{D}_r$ are pairs (d_q, x) where d_q is a state from $ts(\mathcal{D}_q^\nabla)$ and $x \in (\Sigma_r^{loc})^*$ is a sequence of local r-actions that is possible from d_q in $\mathcal{D}^\nabla$, in symbols $d_q \xrightarrow{x}$. We will argue later that such sequences are uniformly bounded. The initial state of $\mathcal{D}_q$ is the state d_q^1 reached from the initial state of $\mathcal{D}_q^\nabla$ by the (unique) transitions of the form $ch(f_0), ch(a_0)$. The initial state of $\mathcal{D}_r$ is (d_q^1, ε). The local transitions for $\mathcal{D}_r$ are $(d_q, x) \xrightarrow{b} (d_q, xb)$, for every $b \in \Sigma_r^{loc}$ and $d_q \xrightarrow{xb}$.

Before defining the transitions of $\mathcal{D}_q$ let us observe that if $d_q \in \mathcal{D}_q^\nabla$ is not in $ts(\mathcal{D}_q^\nabla)$ then only one controllable transition is possible from it. Indeed, as $\mathcal{D}^\nabla$ is a covering controller, if $\pi^\nabla(d_q)$ is of the form (s_q, s_r) then there can be only an outgoing transition on a letter of the form $ch(f)$. Similarly, if $\pi^\nabla(d_q)$ is of the form (s_q, s_r, f) then only a $ch(a)$ transition is possible. Since both $ch(f)$ and $ch(a)$ are controllable, we can assume that in $\mathcal{D}_q^\nabla$ there is no state with two outgoing transitions on a letter of this form. For a state $d_q \in \mathcal{D}_q^\nabla$ not in $ts(\mathcal{D}_q^\nabla)$ we will denote by $ts(d_q)$ the unique state of $ts(\mathcal{D}_q^\nabla)$ reachable from d_q by one or two transitions of the kind $\xrightarrow{ch(f)}$ or $\xrightarrow{ch(a)}$, depending on the cases discussed above. For example, going back to Figure 4, we have $ts(d_3) = d_4$.

We now describe the q-actions possible in $\mathcal{D}$.

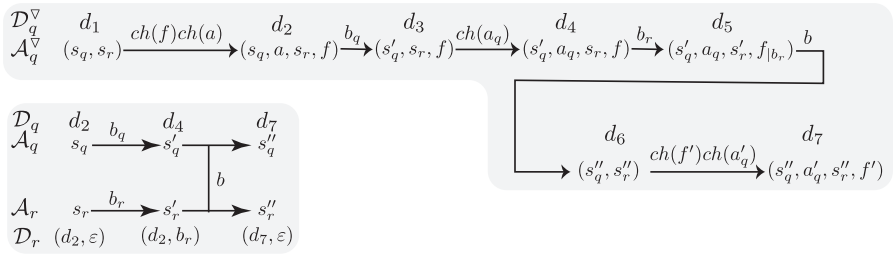


Figure 4 Decomposing controller $\mathcal{D}_q^\nabla$ into $\mathcal{D}_q$ and $\mathcal{D}_r$.

- Local q-action $b \in \Sigma_q^{loc}$: $d_q \xrightarrow{b} ts(d'_q)$ if $d_q \xrightarrow{b} d'_q$ in $\mathcal{D}_q^\nabla$.
- Communication $b \in \Sigma_q \cap \Sigma_p$ between q and $p \neq r$: $(d_p, d_q) \xrightarrow{b} (d'_p, ts(d'_q))$ if $(d_p, d_q) \xrightarrow{b} (d'_p, d'_q)$ in $\mathcal{D}^\nabla$.
- Communication $b \in \Sigma_q \cap \Sigma_r$ of q and r: $(d_q^1, (d_q^2, x)) \xrightarrow{b} (ts(d''_q), (ts(d''_q), \varepsilon))$ if $d_q^1 \xrightarrow{x} d'_q \xrightarrow{b} d''_q$ in $\mathcal{D}_q^\nabla$; observe that $\xrightarrow{x}$ is a sequence of transitions.

In Figure 4 transitions on b_1 and b are examples of transitions of the first and the third type, respectively. In the last item the transition does not depend on d_q^2 since, informally, d_q^1 has been reached from d_q^2 by a sequence of actions independent of r. The condition $d_q^1 \xrightarrow{x} d'_q \xrightarrow{b} d''_q$ simulates the order of actions where all local r-actions come after the other actions of q, then we add a communication between q and r.

The next lemma says that $\mathcal{D}$ is a covering controller for $\mathcal{A}$. Since $\mathcal{A}$ is assumed to be r-short, the lemma also gives a bound on the length of sequences in the states of $\mathcal{D}_r$.

▶ **Lemma 14.** *If $\mathcal{D}^\nabla$ is a covering controller for $\mathcal{A}^\nabla$ then $\mathcal{D}$ is a covering controller for $\mathcal{A}$.*

As $\mathcal{D}$ is covering, to prove that $\mathcal{D}$ is correct we need to show that all its maximal runs satisfy the correctness condition. For this we will construct for every run of $\mathcal{D}$ a corresponding run of $\mathcal{D}^\nabla$. The following definition and lemma tells us that it is enough to look at the runs of $\mathcal{D}$ of a special form.

▶ **Definition 15** (*slow*). We define $slow_r(\mathcal{D})$ as the set of all sequences labeling runs of $\mathcal{D}$ of the form $y_0x_0a_1 \cdots a_k y_k x_k a_{k+1} \ldots$ or $y_0x_0a_1 \cdots y_{k-1}x_{k-1}a_k x_k y_\omega$, where $a_i \in \Sigma_q \cap \Sigma_r$, $x_i \in (\Sigma_r^{loc})^*$, $y_i \in (\Sigma \setminus \Sigma_r)^*$, and $y_\omega \in (\Sigma \setminus \Sigma_r)^\omega$

▶ **Lemma 16.** *A covering controller $\mathcal{D}$ is correct for $\mathcal{A}$ iff for all $w \in slow_r(\mathcal{D})$, $run(w)$ satisfies the correctness condition inherited from $\mathcal{A}$.*

For every sequence $w \in slow_r(\mathcal{D})$ as in Definition 15 we define the sequence $\chi(w) \in (\Sigma^\nabla)^\infty$, and show that it is a run of $\mathcal{D}^\nabla$. The definition is by induction on the length of w. Let $\chi(\varepsilon) = ch(f_0)\, ch(a_0)$, where f_0 and a_0 are determined by the initial q-state of $\mathcal{D}^\nabla$.

$$\text{For } w \in \Sigma^*, b \in \Sigma \text{ let} \qquad \chi(wb) = \begin{cases} \chi(w)b & \text{if } b \notin \Sigma_q \\ \chi(w)b\, ch(a) & \text{if } b \in \Sigma_q \setminus \Sigma_r \\ \chi(w)b\, ch(f)\, ch(a) & \text{if } b \in \Sigma_q \cap \Sigma_r. \end{cases}$$

where a and f are determined by the state reached by $\mathcal{D}^\nabla$ on $\chi(w)b$. The next lemma implies the correctness of the construction, and at the same time confirms that the above definition makes sense, that is, the needed runs of $\mathcal{D}^\nabla$ are defined.

▶ **Lemma 17.** *For every sequence $w \in slow_r(\mathcal{D})$ we have that $\mathcal{D}^\nabla$ has a run on $\chi(w)$. This run is maximal in $\mathcal{D}^\nabla$ if $run(w)$ is maximal in $\mathcal{D}$. In consequence, if $\mathcal{D}^\nabla$ is a correct covering controller for $\mathcal{A}^\nabla$, then $\mathcal{D}$ is a correct covering controller for $\mathcal{A}$.*

4 Conclusion

We have considered a model obtained by instantiating Zielonka automata into the supervisory control framework of Ramadge and Wonham [17]. The result is a distributed synthesis framework that is both expressive and decidable in interesting cases. To substantiate we have sketched how to encode threaded boolean programs with compare-and-swap instructions into our model. Our main decidability result shows that the synthesis problem is decidable for hierarchical architectures and for all local omega-regular specifications. Recall that in the Pnueli and Rosner setting essentially only pipeline architectures are decidable, with an additional restriction that only the first and the last process in the pipeline can handle environment inputs. In our case all processes can interact with the environment.

The synthesis procedure presented here is in k-EXPTIME for architectures of depth k, in particular it is EXPTIME for the case of a one server communicating with clients who do not communicate between each other. From [7] we know that these bounds are tight.

This paper essentially closes the case of tree architectures introduced in [7]. The long standing open question is the decidability of the synthesis problem for all architectures [5].

References

1 A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controllers with partial observation. *Theoretical Computer Science*, 303(1):7–34, 2003.

2 Julian Bradfield and Colin Stirling. Modal mu-calculi. In P. Blackburn, J. van Benthem, and F. Wolter, editors, *The Handbook of Modal Logic*, pages 721–756. Elsevier, 2006.

3 Bernd Finkbeiner and Ernst-Rüdiger Olderog. Petri games: Synthesis of distributed systems with causal memory. In *Proc. of GandALF*, EPTCS, pages 217–230, 2014.

4 Bernd Finkbeiner and Sven Schewe. Uniform distributed synthesis. In *LICS*, pages 321–330, 2005.

5 Paul Gastin, Benjamin Lerman, and Marc Zeitoun. Distributed games with causal memory are decidable for series-parallel systems. In *FSTTCS*, volume 3328 of *LNCS*, pages 275–286, 2004.

6 Paul Gastin and Nathalie Sznajder. Fair synthesis for asynchronous distributed systems. *ACM Transactions on Computational Logic*, 2013.

7 Blaise Genest, Hugo Gimbert, Anca Muscholl, and Igor Walukiewicz. Asynchronous games over tree architectures. In *Proceedings of ICALP'13*, 2013.

8 Susanne Graf, Doron Peled, and Sophie Quinton. Achieving distributed control through model checking. *Formal Methods in System Design*, 40(2):263–281, 2012.

9 Julian Gutierrez and Glynn Winskel. Borel determinacy of concurrent games. In *Proceedings of CONCUR'13*, 2013.

10 O. Kupferman and M. Y. Vardi. Synthesizing distributed systems. In *LICS*, 2001.

11 P. Madhusudan and P. S. Thiagarajan. Distributed control and synthesis for local specifications. In *ICALP'01*, volume 2076 of *LNCS*, pages 396–407. Springer, 2001.

12 P. Madhusudan, P. S. Thiagarajan, and S. Yang. The MSO theory of connectedly communicating processes. In *FSTTCS*, volume 3821 of *LNCS*, pages 201–212, 2005.

13 Paul-André Melliès. Asynchronous games 2: The true concurrency of innocence. *TCS*, 358(2-3):200–228, 2006.

14 Madhavan Mukund and Milind A. Sohoni. Keeping Track of the Latest Gossip in a Distributed System. *Distributed Computing*, 10(3):137–148, 1997.

15 Anca Muscholl and Sven Schewe. Unlimited decidability of distributed synthesis with limited missing knowledge. In *Proceedings of MFCS'13*, 2013.

16 A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *31th IEEE Symposium Foundations of Computer Science (FOCS 1990)*, pages 746–757, 1990.

17 P. J. G. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(2):81–98, 1989.

18 Igor Walukiewicz. Pushdown processes: Games and model checking. *Information and Computation*, 164(2):234–263, 2001.

19 W. Zielonka. Notes on finite asynchronous automata. *RAIRO–Theoretical Informatics and Applications*, 21:99–135, 1987.