Unlimited decidability of distributed synthesis with limited missing knowledge*

Anca Muscholl[1] and Sven Schewe[2]

[1] LaBRI, CNRS, Universiy of Bordeaux, France [2] University of Liverpool, UK

Abstract. We study the problem of controller synthesis for distributed systems modelled by Zielonka automata. While the decidability of this problem in full generality remains open and challenging, we suggest here to seek controllers from a parametrised family: we are interested in controllers that ensure frequent communication between the processes, where frequency is determined by the parameter. We show that this restricted controller synthesis version is affordable for synthesis standards: fixing the parameter, the problem is EXPTIME-complete.

1 Introduction

The synthesis problem has a long tradition that goes back to Church's solvability problem [4], which asks for devices that generate output streams from input streams, such that a given specification is met. Synthesis of sequential systems has been thoroughly studied and driven various results for infinite 2-player games (see [15] for a survey).

Synthesis of *distributed systems* has a bad reputation. Many possible variants of distributed synthesis could be considered. But in the best known and most studied one, initiated by Pnueli and Rosner [26], the problem is undecidable in general – cf. also variations [18] and generalisations [22,9] thereof. These models extend Church's formulation to a fixed architecture of *synchronously* communicating processes that exchange messages through one-slot communication channels. Undecidability in this setting comes mainly from *partial information*: architectures (the communication topology) restrict the flow of information about the global system state. Synthesis in a given architecture is decidable, iff this partial knowledge defines a preorder on the processes [9]. The complexity of the decision problem is non-elementary in the number of quotients. When extended to asynchronous communication with one-slot channels, only systems where a single process needs to be synthesised remain decidable [30].

We use here a different synchronisation model, based on shared variables and known as Zielonka automata [31]. In this model, processes that execute shared actions get full information about the states of the processes with whom they synchronise. Therefore, partial information is reduced to concurrency: the only missing knowledge that a process might have concerns those events that happen concurrently. Partial information in this model is therefore minimalistic, in the sense that it is not driven by the specification or the architecture. As a consequence, establishing the (un)decidability of distributed synthesis in this setting has proven to be challenging and remains open. We know, however, of some non-trivial cases where the problem is decidable. The first one

* The research was supported through a visiting professorship by the University of Bordeaux and by the Engineering and Physical Sciences Research Council grant EP/H046623/1.

[23] imposes a bound on the missing knowledge of a process concerning the evolution of other processes. This restriction mainly says that every event in the system may have only a bounded number of concurrent events. In this setting, the distributed game can be reduced to a 2-player game with complete information. The proof of [23] actually uses Rabin's theorem about decidability of monadic second-order logic over infinite trees. The second decidability result is based on a restriction on the distributed alphabet of actions [11], which needs to be a co-graph, and it applies to global reachability conditions. More recently, it has been shown that distributed synthesis with local reachability conditions is decidable under the assumption that the synchronisation graph is acyclic [13]. The exact complexity is non-elementary in the depth of the synchronisation tree. For instance, it is EXPTIME for trees of depth 1, as for architectures involving one server and several clients. The decidability proof also involves a reduction to a 2-player game.

The complexity of distributed synthesis with shared variables is therefore forbiddingly high, unless the class of strategies under consideration is restricted. The reason for this high complexity is, once again, the partial knowledge a process has about other processes. In the acyclic case studied in [13], partial knowledge is hierarchical. This resembles the situation from Pnueli and Rosner's setting [26,18,22,9], and similarly increases the complexity by one exponent for each additional level of the hierarchy.

With this observation in mind, we reconsider the result of [23] and restrict the class of strategies in such a way, that missing knowledge is uniformly limited. The restriction on strategies is very similar to the notion of N-communicating plants used in [23] to show decidability of monadic second-order logic over the event structure associated with the plant. The main differences are that (1) we do not require that the N-communicating restriction is made explicit in the plant, but more liberally look for strategies that impose N-communication on the controlled system, (2) the bound N applies only to synchronisation events: there is no limitation of local actions, and (3) the winning condition is local on each process. The first condition above is reminiscent of the bounded-context restriction used in model-checking [27], where local computations are unrestricted and only context-switches are limited. To keep the presentation simple, we do not consider divergent infinite plays, where two disjoint groups of processes can synchronise infinitely often in parallel (our result can be adapted to include this case).

Our main result is that the existence of distributed strategies for a system described by a Zielonka automaton $\mathcal{A}$ and a fixed bound N is exponential in the size of $\mathcal{A}$ and doubly exponential in N. If N is fixed, then the problem is EXPTIME-complete.

Related work. The restriction to solutions that obey various bounds in synthesis [10,17,8,6,2] has been inspired by similar restrictions in model-checking, e.g., in bounded model-checking [3] and model-checking with bounded context switches [27,1].

The first two bounds used in synthesis were bounds on the size of the model [10,17] and bounds on the number of rejecting states [19,10] in emptiness equivalent determinisation procedures from universal Co-Büchi automata to deterministic Büchi [19] and safety [10] automata. The latter approach has been implemented by different groups [8,6], while the first has been extended to quantitative specification languages [2], as well as to restrictions on the size of symbolic representations of implementations [7,20]. The implementations of genetic synthesis algorithms in [16] is of the same kind, as the fitness functions used effectively restrict the size of the synthesised programs.

2 Zielonka automata

Informally, Zielonka automata are parallel compositions of finite-state processes that synchronise on shared actions. There is no global clock, so between two synchronisations, two processes can perform a different number of actions. Because of this, Zielonka automata are also called asynchronous automata.

A Zielonka automaton has a (fixed) assignment of actions to sets of processes. A *distributed action alphabet* on a finite set $\mathbb{P}$ of processes is a pair (Σ, dom), where Σ is the finite set of *actions* and $dom : \Sigma \to (2^{\mathbb{P}} \setminus \emptyset)$ is the *location function*. The location $dom(a)$ of an action $a \in \Sigma$ comprises all processes that synchronise in order to perform a. Similar to other classical synchronisation mechanisms, e.g., CCS-like rendez-vous or Petri net transitions, executing a shared action is only possible if the states of all processes in $dom(a)$ allow to execute a. In addition, the execution of a shared action allows to "broadcast" some information between its processes: for instance, an action shared between processes p and q may produce a swap between the states of p and q. Related concepts are used in multithreaded programming, where atomic instructions like compare-and-swap (CAS) allow to exchange values between two processes.

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

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

For convenience, we abbreviate a tuple $(s_p)_{p\in P}$ of local states by s_P, where $P \subseteq \mathbb{P}$. We also refer to S_p as the set of *p-states* and of $\prod_{p\in\mathbb{P}} S_p$ as *global 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 $s_{\mathbb{P}\setminus dom(a)} = s'_{\mathbb{P}\setminus dom(a)}$. By $L(\mathcal{A})$ we denote the language of this sequential automaton.

This definition has an important consequence. The location mapping dom defines in a natural way an independence relation $I \subseteq \Sigma \times \Sigma$: two actions $a, b \in \Sigma$ are independent (written as $(a, b) \in I$) if the processes they involve are disjoint, that is, if $dom(a) \cap dom(b) = \emptyset$. Note that the order of execution of two independent actions $(a, b) \in I$ in a Zielonka automaton is irrelevant, they can be executed as a, b, or b, a – or even concurrently. More generally, we can consider the congruence $\sim_I$ on Σ^* generated by I, and observe that, whenever $u \sim_I v$, the state reached from the initial state on u and v, respectively, is the same. Hence, $u \in L(\mathcal{A})$ if, and only if, $v \in L(\mathcal{A})$. We denote u, v as *trace-equivalent* whenever $u \sim_I v$ (and write $u \sim v$ for simplicity).

The idea of describing concurrency by an independence relation on actions was introduced in the late seventies by Mazurkiewicz [24] (see also [5]). An equivalence class $[w]$ of $\sim$ is called a Mazurkiewicz *trace*, it can be viewed as a labelled pomset. We will often refer to a trace using just a word w instead of writing $[w]$. As we have observed $L(\mathcal{A})$ is a sum of such equivalence classes. In other words, the language of a Zielonka automaton is *trace-closed*.

Actions a with $|dom(a)| = 1$ are called *local*, and Σ^{loc} is the set of local actions. If $|dom(a)| > 1$ then a is called *synchronisation action*, and Σ^{sync} is the set of such

actions. Actions from $\Sigma_p = \{a \in \Sigma \mid p \in dom(a)\}$ are denoted as *p-actions*. We write $\Sigma_p^{sync} = \Sigma^{sync} \cap \Sigma_p$ and $\Sigma_p^{loc} = \Sigma^{loc} \cap \Sigma_p$. For $u \in \Sigma^*$ we write $state_p(u)$ for the p-state reached by $\mathcal{A}$ on u.

Example 1. Consider the example automaton with processes $P_1, \ldots, P_n$, $S_1, \ldots, S_m$ as shown in Figure 1. Here, processes $S_1, \ldots, S_m$ are backup servers and each of the processes P_i loops on a sequence of internal actions, followed by backup actions on some server. We abstract this by the following distributed alphabet: ℓ_i, b_i are local actions of P_i (i.e., $dom(\ell_i) = dom(b_i) = \{P_i\}$), where b_i denotes a backup request on P_i, and $s_{i,k}$ is a shared (backup) action with $dom(s_{i,k}) = \{P_i, S_k\}$. Action $s_{i,k}$ is enabled if P_i is in state 1_i. Actions $\ell_i, b_i, s_{i,k}$ $(k = 1, \ldots, m)$ are P_i-actions, and $\Sigma^{sync} = \{s_{i,k} \mid i, k\}$. Note also that s_{i_1,k_1}, s_{i_2,k_2} are independent iff $i_1 \neq i_2$ and $k_1 \neq k_2$.

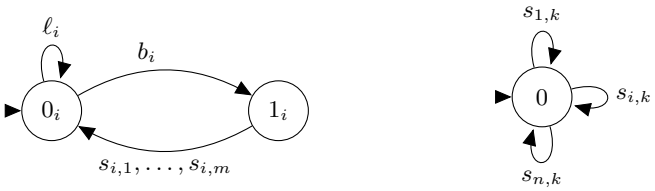

Fig. 1. Example Zielonka automaton; Process P_i on the left and server S_k on the right.

A major result about Zielonka automata is stated in the theorem below. Note that it is one of the few examples of synthesis of (*closed*) distributed systems.

Theorem 1. *[31] Let $dom : \Sigma \to (2^{\mathbb{P}} \setminus \{\emptyset\})$ be a distribution of letters. If a language $L \subseteq \Sigma^*$ is regular and trace-closed, then there is a deterministic Zielonka automaton recognising L. Its size is exponential in the number of processes and polynomial in the size of the minimal automaton for L [12].*

3 Distributed control and games

The synthesis problem considered here was proposed in [23]. It can be viewed as a distributed instantiation of supervisory control, as considered in the framework of Ramadge and Wonham [28]. In supervisory control, one is given a plant $\mathcal{A}$, together with a partition $\Sigma = \Sigma^{sys} \dot\cup \Sigma^{env}$ of Σ into controllable actions Σ^{sys} and uncontrollable actions Σ^{env}. As in [23,13] we assume that all uncontrollable actions are local, $\Sigma^{env} \subseteq \Sigma^{loc}$. The goal is to synthesise a controller $\mathcal{C}$, which is a device that never blocks uncontrollable actions. The controlled plant is then the product of $\mathcal{A}$ and $\mathcal{C}$, and it needs to satisfy additional conditions like safety, reachability, or parity conditions.

We will work with the game description of the controller problem, and start by illustrating it on an example.

Example 2. Reconsider the automaton from Figure 1 and assume that local actions are uncontrollable, whereas synchronisation actions are controllable. In this model, the environment decides whether a process P_i continues to use local transitions or needs

backup, while the system is in charge of deciding each time, on which server(s) the backup is made. One possible objective of the control strategy could be to achieve a balanced use of the servers, which avoids using certain servers more often than others. A round-robin strategy on each process P_i, e.g., that asks each time for backup on the next server, guarantees that a server S_k can be "behind" the other servers by at most $O(nm)$ backup actions.

The game formulation refers to a game between the distributed *system* and *local environments*, one for each process. A Zielonka automaton $\mathcal{A}$ defines a game arena, with plays corresponding to initial runs. Since $\mathcal{A}$ is deterministic, we can view a play x as a word from $L(\mathcal{A})$ – or a trace, since $L(\mathcal{A})$ is trace-closed. Let $\mathit{Plays}(\mathcal{A})$ denote the set of traces associated with words from $L(\mathcal{A})$.

A strategy for the system will be a collection of individual strategies for each process. The important notion here is the view each process has of the global state of the system. Intuitively, this is the part of the current play that the process could see or learn about from other processes by synchronising with them. Formally, the p-view of a play x, denoted $\mathit{view}_p(u)$, is the smallest trace $[v]$ such that $u \sim vy$ and y contains no action from Σ_p. We write $\mathit{Plays}_p(\mathcal{A})$ for the set of plays that are p-views: $\mathit{Plays}_p(\mathcal{A}) = \{\mathit{view}_p(u) \mid u \in \mathit{Plays}(\mathcal{A})\}$.

A *strategy for a process* p is a function $\sigma_p : \mathit{Plays}_p(\mathcal{A}) \to 2^{\Sigma_p^{sys}}$, where $\Sigma_p^{sys} = \{a \in \Sigma^{sys} \mid p \in \mathit{dom}(a)\}$. We require in addition, for every $u \in \mathit{Plays}_p(\mathcal{A})$, that $\sigma_p(u)$ is a subset of the set enabled(s_p) of actions that are enabled in $s_p = \mathit{state}_p(u)$. A *strategy* is a family of strategies $\{\sigma_p\}_{p\in\mathbb{P}}$, one for each process.

The set of plays respecting a strategy $\sigma = \{\sigma_p\}_{p\in\mathbb{P}}$, denoted $\mathit{Plays}(\mathcal{A}, \sigma)$, is the smallest set that contains the empty play ε and that satisfies, for every $u \in \mathit{Plays}(\mathcal{A}, \sigma)$, the following two conditions: (1) if $a \in \Sigma^{env}$ and $ua \in \mathit{Plays}(\mathcal{A})$, then ua is in $\mathit{Plays}(\mathcal{A}, \sigma)$, and (2) if $a \in \Sigma^{sys}$ and $ua \in \mathit{Plays}(\mathcal{A})$, then $ua \in \mathit{Plays}(\mathcal{A}, \sigma)$ provided that $a \in \sigma_p(\mathit{view}_p(u))$ *for all* $p \in \mathit{dom}(a)$. So this definition says that actions of the environment are always possible / enabled, whereas actions of the system are possible only if they are allowed by the strategies of all involved processes.

Before defining winning (control) strategies, we need to introduce infinite plays that are consistent with a given strategy σ. Such plays can be viewed as (infinite) traces associated with infinite initial runs of $\mathcal{A}$ that satisfy both conditions of the definition of $\mathit{Plays}(\mathcal{A}, \sigma)$. The precise definition is very intuitive when using pomsets, here we just give an example: the infinite play $a^\omega b^\omega$ is the set of all ω-words with infinitely many as and infinitely many bs. We write $\mathit{Plays}^\infty(\mathcal{A}, \sigma)$ for the set of such *finite or infinite* plays. A play from $\mathit{Plays}^\infty(\mathcal{A}, \sigma)$ is also denoted as σ*-play*. A play $u \in \mathit{Plays}^\infty(\mathcal{A}, \sigma)$ is called *maximal*, if there is no action c such that $uc \in \mathit{Plays}^\infty(\mathcal{A}, \sigma)$.

Winning conditions. In analogy to regular 2-player games, winning conditions in these games can be provided by regular, trace-closed languages [23]. In this paper, we consider simpler conditions, namely *local parity* conditions, because we are interested in the game complexity and do not want to add the specification as an extra parameter.

Our system $\mathcal{A}$ is thus a deterministic Zielonka automaton with local states, coloured by integers from $[k] = \{0, \ldots, k-1\}$: let $\mathcal{A} = \langle (S_p)_{p\in\mathbb{P}}, (\delta_a)_{a\in\Sigma}, s^0, \chi\rangle$, $\chi : \bigcup_{p\in\mathbb{P}} S_p \to [k]$. A maximal play $u \in \mathit{Plays}^\infty(\mathcal{A}, \sigma)$ is winning, if the following holds for every process p. Write $\mathit{view}_p(u)$ as $u_0u_1\ldots$, for $u_0, u_1, \ldots$ such that, for every n,

we have that $view_p(u_0 \cdots u_n) = u_0 \cdots u_n$ and either u_n is empty or it has only one p-action (which is the last one). Then we require that $\liminf_{n\to\infty} \chi(state_p(u_0 \cdots u_n))$ is even. Equivalently, if $view_p(u)$ is infinite, then the local parity condition should hold, and if $view_p(u)$ is finite, then the colour of the last state reached by p needs to be even. A strategy σ is winning, if every maximal play in $Plays^\infty(\mathcal{A}, \sigma)$ is winning. Maximality is a sort of fairness condition for such automata. Requiring infinite plays as in [23] is also possible, but it does not guarantee fairness for each process. A more refined notion of fairness can be found in [14].

Remark 1. The decidability of the existence of a winning distributed control strategy for systems modelled by Zielonka automata is an open problem. It is worth noting that slight modifications of the problem statement lead to undecidability. First, if one uses regular, but *not trace-closed* specifications, then the problem is known to be undecidable (see e.g. [21]). Second, if the individual strategy σ_p only depends only on the local history of process p (i.e., $\sigma_p : (\Sigma_p)^* \to 2^{\Sigma_p^{sys}}$), then the problem is again undecidable [21]. In both cases, undecidability stems from the *restricted* partial knowledge of the processes.

4 Resuming local behaviour

Recall that $\Sigma^{env} \subseteq \Sigma^{loc}$, i.e., environment actions are local. As shown in this section, this allows to summarise local behaviour, such that one can reason about distributed strategies only w.r.t. synchronisation actions.

Lemma 1. *Let $\mathcal{A}$ be a Zielonka automaton. If there is a winning control strategy $\sigma = (\sigma_p)_{p\in\mathbb{P}}$ for $\mathcal{A}$, then there also exists a winning one that satisfies, for every process $p \in \mathbb{P}$ and every play $t \in Plays_p(\mathcal{A}, \sigma)$, either $\sigma_p(t) = \{a\}$ for some $a \in \Sigma_p^{loc} \cap \Sigma^{sys}$, or $\sigma_p(t) \subseteq \Sigma_p^{sync}$. In addition, $\sigma_p(t) = \emptyset$ if enabled$(state_p(t)) \cap \Sigma_p^{env} \neq \emptyset$.*

The proof exploits that, when both local and synchronisation actions are enabled, disabling the synchronisation actions reduces the set of plays, but they are still winning.

Definition 1. *Fix some process p. A* local p-play *is a word from $(\Sigma_p^{loc})^*$. A* p-context *is a play from $Plays_p(\mathcal{A})$ that ends with an action from Σ_p^{sync} (unless it is empty).*

Given a distributed strategy $(\sigma_p)_{p\in\mathbb{P}}$, we associate with a p-context u a local strategy *from u: this is the mapping $\sigma_p[u] : (\Sigma_p^{loc})^* \to 2^{\Sigma_p}$ defined as*

$$\sigma_p[u](x) := \sigma_p(ux) \qquad \text{for all } x \in (\Sigma_p^{loc})^*.$$

We assume in the following that $\sigma = (\sigma_p)_{p\in\mathbb{P}}$ satisfies Lemma 1, thus $\sigma_p[u] : (\Sigma_p^{loc})^* \to (\Sigma_p^{loc} \cap \Sigma^{sys}) \cup 2^{\Sigma_p^{sync}}$. We are interested in the configurations that result after a *maximal* local run of process p from a given p-context u with $s_p = state_p(u)$. We define:

$$Sync^\sigma(p, u) = \{(s'_p, A, c) \mid \exists x \in (\Sigma_p^{loc})^* : s'_p = state_p(ux), A = \sigma_p(ux) \subseteq \Sigma_p^{sync}, \\ \text{enabled}(s'_p) \cap \Sigma_p^{env} = \emptyset, \text{ and the minimal colour seen on } s_p \xrightarrow{x} s'_p \text{ is } c\}.$$

A local strategy $\sigma_p[u]$ is called *simple* if, for every $(s'_p, A, c), (s''_p, A', c') \in Sync^\sigma(p,u)$, we have that $s'_p = s''_p$ implies $A = A'$. In this case $Sync^\sigma(p,u)$ is a partial mapping $Sync^\sigma(p,u) : S_p \stackrel{\cdot}{\to} 2^{\Sigma_p^{sync}} \times 2^{[k]}$.

A local strategy $\sigma_p[u]$ from context u is computable with memory M if $\sigma_p[u](x)$ can be computed from $state_p(ux)$ using an additional *finite* memory M. In this case, $\sigma_p[u]$ is a mapping from $S_p \times M$ to $(\Sigma_p^{loc} \cap \Sigma^{sys}) \cup 2^{\Sigma_p^{sys}}$.

Lemma 2. *If there is a winning control strategy $\sigma = (\sigma_p)_{p\in\mathbb{P}}$ for $\mathcal{A}$ with a local parity condition with k colours, then there is also a winning one, say $\tau = (\tau_p)_{p\in\mathbb{P}}$, where, for each process p and every p-context u, the local strategy $\tau_p[u]$ is simple, computable with memory of size k, and such that every infinite (and thus local) $\tau_p[u]$-play satisfies the parity condition for process p.*

The proof exploits that, for every s'_p that can occur at the end of a run, one can select a triple (s'_p, A, c) with worst color among the elements of $Sync^\sigma(p,u)$ and then change the decision for each such end-point s'_p to A. It is then easy to turn the resulting simple local strategy into one, where the decision is only based on the state and the minimal colour that occurred so far.

We denote local strategies $\tau_p[u]$ as in Lemma 2, as *good* strategies. In Section 6 we will compose good strategies, and we therefore define their *outcomes*.

Definition 2. *Let u be a p-context. The outcome of a simple strategy $\tau_p[u]$ is a partial mapping $f : S_p \stackrel{\cdot}{\to} 2^{\Sigma_p^{sync}} \times [k]$ that satisfies the following side constraints:*

1. *f and $Sync^\tau(p,u)$ have the same domain, and*
2. *for each state s_p in the domain of f: if $Sync^\tau(p,u)(s_p) = (A, C)$ for some $A \subseteq \Sigma_p^{sync}$ and $C \subseteq [k]$ then $f(s_p) = (A, c)$ where:*
   - *either $C \not\subseteq 2\mathbb{N}$, c is odd and $c \leq d$ for every odd colour $d \in C$, or*
   - *$C \subseteq 2\mathbb{N}$, c is even, and $c \geq \max(C)$.*

Remark 2. Note that we can test, for given $s_p \in S_p$ and partial mapping $f : S_p \stackrel{\cdot}{\to} 2^{\Sigma_p^{sync}} \times [k]$, whether f is the outcome of a good local strategy from state s_p. The test amounts to solving a 2-player game with parity condition on infinite plays. Finite plays are won if the last state, say t_p, is in the domain of f. In addition, if $f(t_p) = (A, c)$ for some A, then t_p can be reached only with even minimal colours $d \leq c$ if c is even. If c is odd, and t_p is reached with odd minimal colour d, then $d \geq c$. The condition on colours can be checked using additional memory k.

5 Well-informed strategies

We start by defining the distributed strategies we are interested in. They are very similar to the notion of N-communicating plants used in [23], with two exceptions. First, our bound N applies only to synchronisation actions. That is, there is no limitation on local actions. Second, our definition implies that infinite plays are non-divergent, which is a restriction that we impose only for simplifying the presentation (see also Remark 4).

Definition 3. *Let $N > 0$ be an integer. A strategy $\sigma = (\sigma_p)_{p\in\mathbb{P}}$ is called* N-informed *if, for every play $u \in Plays^{\infty}(\mathcal{A}, \sigma)$ such that $u = u'av$ with $a \in \Sigma^{sync}$, $v \in \Sigma^{\infty}$ and $dom(a) \cap dom(v) = \emptyset$, it holds that v has at most N actions from Σ^{sync}.*

The round-robin strategy mentioned in Example 2 is N-informed with $N \in O(nm)$. Note that Lemma 2 preserves N-informedness, since only local strategies are modified and Def. 3 refers only to synchronisation actions. By abuse of notations, we will call a sequence from $(\Sigma^{sync})^{\infty}$ *N-informed*, if it satisfies the above definition.

Let us fix some total order $<$ on Σ^{sync}. A sequence $u \in (\Sigma^{sync})^*$ is said to be in lexicographic normal form (w.r.t. $<$) if there is no trace-equivalent sequence $u' \sim u$ such that $u = vbw$, $u' = vaw'$ with $a < b$. We denote by $\mathrm{lnf}(u)$ the trace-equivalent sequence $v \sim u$ that is in lexicographic normal form. Sequences in lexicographic normal form build a regular set: a sequence $u \in (\Sigma^{sync})^*$ is *not* in lexicographic normal form iff, for some $x, y, z \in (\Sigma^{sync})^*$:

$$u = xbyaz \quad \text{with } dom(a) \cap dom(by) = \emptyset \text{ and } a < b\,. \tag{1}$$

Let $\mathfrak{s} = \max_{p\in\mathbb{P}} |S_p|$, $\mathfrak{c} = \max_{p\in\mathbb{P}} |\Sigma_p^{sync}|$, $\mathfrak{p} = |\mathbb{P}|$, and recall that k is the number of colours. A deterministic safety automaton of size $O(\mathfrak{c} \cdot 2^{\mathfrak{p}})$ exists that accepts the set of sequences in lexicographic normal form. This automaton records, for every $a \in \Sigma^{sync}$, the set of processes in whose view the last a occurs.

The next lemma considers how the lexicographic normal form changes when extending a sequence from $(\Sigma^{sync})^*$, showing that only a bounded suffix is modified. The lemma is essentially the same as Lemma 3 in [23]:

Lemma 3. *[23] Let $u \in (\Sigma^{sync})^*$ be an N-informed sequence and $a \in \Sigma^{sync}$. Then $lnf(u) = zx$ and $lnf(u \cdot a) = zy$ for some x, y, z with $|x| \leq N$.*

The next lemma makes the statement of Lemma 3 more precise:

Lemma 4. *Let $u, v \in (\Sigma^{sync})^*$ be N-informed and $p \in \mathbb{P}$ such that $u = view_p(v)$ and both u, v are in lexicographic normal form. Then we can write $u = zx$ and $v = zy$, with $y = y_0x_1y_1x_2 \cdots y_{m-1}x_my_m$ for some $m \leq \mathfrak{c} \cdot \mathfrak{p}$, such that (1) $|x| \leq N$, (2) $x = x_1 \cdots x_m$, and (3) $dom(y_i) \cap dom(x_{i+1} \cdots x_m) = \emptyset$ for every $i < m$ hold.*

6 Strategy trees

Let $\Omega = \prod_{p\in\mathbb{P}} \Omega_p$, where Ω_p is a set of tuples (s_p, f, b), where f is outcome of some good local p-strategy τ from state s_p and $b \in \{0, 1\}$ says whether τ allows infinite (local) plays from s_p. Let $\Delta = \bigcup_{a\in\Sigma^{sync}} \delta_a$. A *strategy tree* is an infinite tree with directions $\Gamma = \Sigma^{sync} \times \Delta$ and nodes labelled by elements of $\Omega \cup \{\bot\}$. Note that $|\Gamma| \leq \mathfrak{c} \cdot |\mathcal{A}|$. A node in the tree is identified with the sequence from Γ^* labelling the path from the root to that node. We require for every pair of nodes $u, u \cdot \langle a, d\rangle \in \Gamma^*$:

1. $d \in \delta_a$, and
2. if the labels of u and $u \cdot \langle a, d\rangle$ are ω' and ω, resp., then $\omega'_q = \omega_q$ for all $q \notin dom(a)$.

For $u \in \Gamma^*$, we denote by $state_p(u)$ the state of process p after u. This is namely the state that occurs in the transition d of the last pair $\langle a, d\rangle$ with $p \in dom(a)$.

Nodes in a strategy tree that do not correspond to *realisable* summarised plays, are called *sink* nodes and are labelled by $\bot$. A node $u \cdot \langle a, d\rangle$ with label ω' is a sink, but u is not, if, and only if, (1) either the action a is not allowed by $\omega_{dom(a)}$, where ω is the label of u, (2) or the a-transition d does not result from $\omega_{dom(a)}$, (3) or $\omega'_p \neq (t_p, *, *)$ for some $p \in dom(a)$ with $t_p = state_p(u \cdot \langle a, d\rangle)$.

Remark 3. If we fix local good strategies for every pair of local states and outcomes from $S_p \times \Omega_p$, then every initial path $\omega_0, \langle a_0, d_0\rangle, \omega_1, \langle a_1, d_1\rangle \ldots$ in a strategy tree such that all $\omega_i \neq \bot$ can be "expanded" in a natural way to a set of plays from $Plays(\mathcal{A})$.

Lemma 5. *A deterministic safety automaton with $O(\mathfrak{s}^{\mathfrak{p}})$ states can check that the label $\bot$ correctly identifies sink nodes.*

A strategy tree is *winning* if all plays that are expansions of maximal initial paths of the tree, are winning. Checking consistency of the labels ω requires to include in the label of each node $u \in \Gamma^*$ a bit that reflects whether or not the projection of u on Σ^{sync} is in lexicographic normal form. As mentioned in Section 5, a deterministic safety automaton with $O(\mathfrak{c} \cdot 2^{\mathfrak{p}})$ states can check that this labelling is correct. In the following we will focus on non-sink nodes in lexicographic normal form. We will refer to them as *normalised* nodes.

We will need to ensure that a strategy tree has a consistent node labelling. The next definition tells when two nodes $u, u' \in \Gamma^*$ correspond to the same summarised play.

Definition 4. *Two nodes $u, u' \in \Gamma^*$ are called* play-equivalent *if the following hold:*

1. *The projections of u, u' onto Σ^{sync} are trace-equivalent.*
2. *For all $a \in \Sigma^{sync}$ and $k \geq 0$: suppose that $u = u_1\langle a, d\rangle u_2$ and $u' = u'_1\langle a, d'\rangle u'_2$, with $\langle a, d\rangle, \langle a, d'\rangle$ being the k-th occurrence of a in u and u', respectively. Assume also that u_1 and u'_1 is labelled by ω and ω', respectively. Then we require that*

$$d = d' \qquad \text{and} \qquad \omega_{dom(a)} = \omega'_{dom(a)}\,.$$

We ensure that the strategy tree is labelled consistently by local strategies by comparing normalised nodes that are play-equivalent. By abuse of notation, we write $view_p(u)$ for the p-view of $u \in \Gamma^*$.

Definition 5. *A strategy tree is labelled consistently if, for all normalised non-sink nodes $u, v \in \Gamma^*$ and every process p such that u and $view_p(v)$ are play-equivalent, it holds that $\omega_p = \omega'_p$, where ω and ω' are the labels of u and v, respectively.*

Informally, the strategy tree is labelled consistently if the choice of the next outcomes $\omega_{dom(a)}$ after some synchronisation $a \in \Sigma^{sync}$ depends only on the history associated with the views of processes $p \in dom(a)$ after a.

Lemma 6. *Every good control strategy (cf. Lemma 2) maps to a consistently labelled strategy tree. Conversely, from every consistently labelled strategy tree we can construct a good control strategy.*

Lemma 7. *An alternating safety automaton with $O(|\Gamma|^N \cdot N \cdot \mathfrak{s} \cdot \mathfrak{c} \cdot k)$ states can check that a strategy tree associated with an N-informed strategy is labelled consistently.*

Proof. Let $u, v \in \Gamma^*$ be as in Definition 5, so in particular normalised, non-sink, and such that u and $view_p(v)$ are play-equivalent. We can apply Lemma 4 to (the projections on Σ^{sync} of) u, v. Thus, we can write $u = zx$, $v = zy$ with $y = y_0x_1y_1x_2 \cdots y_{m-1}x_my_m$ for some $m \leq \mathfrak{c} \cdot \mathfrak{p}$, such that (1) $|x| \leq N$, (2) $x = x_1 \cdots x_m$, and (3) $dom(y_i) \cap dom(x_{i+1} \cdots x_m) = \emptyset$ for every $i < m$ hold.

An alternating (reachability) tree automaton can check that some $u, v \in \Gamma^*$ as above *do not* satisfy $\omega_p = \omega'_p$, with ω, ω' the labels of u, v. The automaton first guesses (and moves to) node z. It then guesses $x \in \Gamma^{\leq N}$ and two directions where to proceed; it also guesses the difference between the labels of zx and zy, e.g., a state from S_p, an action from Σ_p^{sync}, and some colour. In the first direction it checks that a path labelled by x ends with a p-label consistent with the guessed difference. In the second direction, it checks that the path is of the form $y_0x_1y_1x_2 \cdots y_{m-1}x_m$, with $x = x_1 \cdots x_m$ and $dom(y_i) \cap dom(x_{i+1} \cdots x_m) = \emptyset$ for every $i < m$, and that it ends with a p-label consistent with the guessed difference.

Note that we do not need to remember the intermediate labels ω on the path x, because we can look for a shortest u that witnesses the inconsistency. Then we can assume that u and $view_p(v)$ are play-equivalent (and not only trace-equivalent). The alternating automaton has $O(|\Gamma|^N \cdot N \cdot \mathfrak{s} \cdot \mathfrak{c} \cdot k)$ states, $|\Gamma|^N \cdot N$ for matching x inside $y_0x_1y_1x_2 \cdots y_{m-1}x_m$ and $\mathfrak{s} \cdot \mathfrak{c} \cdot k$ for the guessed difference between node labels.

Lemma 8. *A deterministic safety automaton with $O(\mathfrak{c} \cdot \mathfrak{p}! \cdot 2^{\mathfrak{p}} \cdot (N+1)^{\mathfrak{p}})$ states can check that a strategy tree corresponds to an N-informed control strategy.*

Proof. The state records, at each node u and for each $u = u_1\langle a, d\rangle u_2$, how many synchronisation actions in u are concurrent to this a (up to N), and the set of processes in the causal future of a in u. Note that, if $u = u_1\langle a, d\rangle u_2 = u'_1\langle a, d'\rangle u'_2$ with $|u_1| < |u'_1|$, then the set of processes in the causal future of $\langle a, d\rangle$ is a superset of the set of processes in the future of $\langle a, d'\rangle$. In addition, we need to count, for each $a \in \Sigma^{sync}$, the length of $u \setminus view_{dom(a)}(u)$ (up to $N+1$). Thus, $O(\mathfrak{c} \cdot \mathfrak{p}! \cdot 2^{\mathfrak{p}} \cdot (N+1)^{\mathfrak{p}})$ states suffice.

Lemma 9. *For a given consistently labelled strategy tree for an N-informed strategy, a universal Co-Büchi automaton with $1 + \mathfrak{p}(k+2)$ states can check that each run satisfies the parity condition.*

The proof idea is to construct an automaton that rejects if it can guess, for some process p, a path where the minimal colour occurring infinitely often is an odd colour o. It can guess a point where no lower colour than o occurs and verify (1) this (safety) and (2) that o occurs infinitely often (Co-Büchi). To test the corner case of a process p being scheduled finitely often, the automaton can guess a point where p is not scheduled again and verify (1) this and (2) that it might end in a state with odd colour (both safety).

Remark 4. As mentioned, we consider strategies that produce only non-divergent plays. A divergent play is a play where, whenever a synchronisation event a has more than N synchronisation events $b_1, \ldots, b_M$ in parallel, then the processes of a and those of the

b_i are henceforth separated. It is easy to extend Lemma 8 to the divergent case, we merely need to check N-informedness on non-divergent plays. Extending the winning condition (Lemma 9) requires more care, since we need to consider processes that are scheduled finitely often and to show that a play is maximal for them.

Summing up, we showed how to check the following properties of strategy trees:

1. Sink nodes are identified correctly: deterministic safety automaton $\mathcal{B}_1$ with $O(|S|) = O(\mathfrak{s}^\mathfrak{p})$ states.
2. Normalised nodes are identified correctly: deterministic safety automaton $\mathcal{B}_2$ with $O(\mathfrak{c} \cdot 2^\mathfrak{p})$ states.
3. Strategy tree is labelled consistently: alternating safety automaton $\mathcal{B}_3$ with $O((\mathfrak{c} \cdot |\mathcal{A}|)^N \cdot N \cdot \mathfrak{s} \cdot \mathfrak{c} \cdot k)$ states (Lemma 7).
4. Control strategy is N-informed: deterministic safety automaton $\mathcal{B}_4$ with $O(\mathfrak{c} \cdot \mathfrak{p}! \cdot 2^\mathfrak{p} \cdot (N+1)^\mathfrak{p})$ states (Lemma 8).
5. The parity condition is satisfied: universal Co-Büchi automaton $\mathcal{B}_5$ with $O(\mathfrak{p} \cdot k)$ states (Lemma 9).

Theorem 2. *Given a Zielonka automaton $\mathcal{A}$ with local parity condition and an integer N, the existence of a winning N-informed control strategy can be decided in time doubly exponential in N and exponential in $\mathcal{A}$. For fixed N, the problem is* EXPTIME*-complete. The same bounds apply to the construction of a winning strategy (if it exists).*

The proof exploits the correspondence between N-informed control strategies and trees accepted by the intersection of $\mathcal{B}_1$ through $\mathcal{B}_5$. We intersect them in two steps. Invoking the simulation theorem [25], we first construct a nondeterministic parity automaton $\mathcal{B}'_5$, which is language equivalent to $\mathcal{B}_5$, with polynomially many colours and exponentially many states in the states of $\mathcal{B}_5$. We likewise construct a nondeterministic safety automaton $\mathcal{B}'_3$, which is language equivalent to and exponential in $\mathcal{B}_3$. We can then intersect $\mathcal{B}_1$, $\mathcal{B}_2$, $\mathcal{B}'_3$, $\mathcal{B}_4$, and $\mathcal{B}'_5$ to a nondeterministic parity automaton $\mathcal{B}$ with the same colours as $\mathcal{B}'_5$, whose states are the product states of these five automata.

The emptiness of $\mathcal{B}$ can be checked (and a control strategy constructed) by solving the resulting emptiness parity game, which is polynomial in the number of states, and exponential only in the number of colours [29]. This provides the claimed complexity.

References

1. M. F. Atig, A. Bouajjani, and S. Qadeer. Context-bounded analysis for concurrent programs with dynamic creation of threads. *Logical Methods in Computer Science*, 7(4), 2011.
2. N. Bertrand, J. Fearnley, and S. Schewe. Bounded satisfiability for PCTL. In *Proc. of CSL 2012*, pages 92–106, 2012.
3. A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu. Bounded model checking. *Advances in Computers*, 58:118–149, 2003.
4. A. Church. Logic, arithmetics, and automata. In *Proceedings of the International Congress of Mathematicians*, pages 23–35, 1962.
5. V. Diekert and G. Rozenberg, editors. *The Book of Traces*. World Scientific, 1995.
6. R. Ehlers. Symbolic bounded synthesis. In *Proc. of CAV 2010*, pages 365–379, 2010.

7. J. Fearnley, D. Peled, and S. Schewe. Synthesis of succinct systems. In *Proc. of ATVA 2012*, pages 208–222, 2012.
8. E. Filiot, N. Jin, and J.-F. Raskin. An antichain algorithm for LTL realizability. In *Proc. of CAV 2009*, pages 263–277, 2009.
9. B. Finkbeiner and S. Schewe. Uniform distributed synthesis. In *Proc. of LICS 2005*, pages 321–330, 2005.
10. B. Finkbeiner and S. Schewe. Bounded synthesis. *International Journal on Software Tools for Technology Transfer*, online-first:1–12, 2012.
11. P. Gastin, B. Lerman, and M. Zeitoun. Distributed games with causal memory are decidable for series-parallel systems. In *Proc. of FSTTCS 2004*, pages 275–286, 2004.
12. B. Genest, H. Gimbert, A. Muscholl, and I. Walukiewicz. Optimal Zielonka-type construction of deterministic asynchronous automata. In *Proc. of ICALP 2010*, pages 52–63, 2010.
13. B. Genest, H. Gimbert, A. Muscholl, and I. Walukiewicz. Asynchronous games over tree architectures. In *Proc. of ICALP 2013*.
14. P. Gastin and N. Sznajder. Fair Synthesis for Asynchronous Distributed Systems. *ACM Transactions on Computational Logic*, 2013.
15. E. Grädel, W. Thomas, and Th. Wilke, editors. *Automata, Logics, and Infinite Games*, volume 2500 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.
16. G. Katz and D. Peled. Model checking-based genetic programming with an application to mutual exclusion. In *Proc. of TACAS 2008*, pages 141–156. Springer, 2008.
17. O. Kupferman, Y. Lustig, M. Y. Vardi, and M. Yannakakis. Temporal synthesis for bounded systems and environments. In *Proc. of STACS 2011*, pages 615–626, 2011.
18. O. Kupferman and M. Y. Vardi. Synthesizing distributed systems. In *Proc. of LICS 2001*, pages 389–398, 2001.
19. O. Kupferman and M. Y. Vardi. Safraless decision procedures. In *Proc. of FOCS 2005*, pages 531–540, 2005.
20. P. Madhusudan. Synthesizing Reactive Programs. In *Proc. of CSL*, pages 428–442, 2011.
21. P. Madhusudan and P. Thiagarajan. A decidable class of asynchronous distributed controllers. In *Proc. of CONCUR 2002*, pages 145–160, 2002.
22. P. Madhusudan and P. S. Thiagarajan. Distributed controller synthesis for local specifications. In *Proc. of ICALP 2001*, pages 396–407, 2001.
23. P. Madhusudan, P. S. Thiagarajan, and S. Yang. The MSO theory of connectedly communicating processes. In *Proc. of FSTTCS 2005*, pages 201–212, 2005.
24. A. Mazurkiewicz. Concurrent program schemes and their interpretations. DAIMI Rep. PB 78, Aarhus University, Aarhus, 1977.
25. D. E. Muller and P. E. Schupp. Simulating alternating tree automata by nondeterministic automata: New results and new proofs of the theorems of Rabin, McNaughton and Safra. *Theoretical Computer Science*, 141:69–107, 1995.
26. A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *Proc. of FOCS 1990*, pages 746–757, 1990.
27. S. Qadeer and J. Rehof. Context-Bounded Model Checking of Concurrent Software. In *Proc. of TACAS'05*, pages 93–107, 2005.
28. P. J. G. Ramadge and W. M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(2):81–98, 1989.
29. S. Schewe. Solving parity games in big steps. In *Proc. of FSTTCS*, pages 449–460, 2007.
30. S. Schewe and B. Finkbeiner. Synthesis of asynchronous systems. In *Proc. of LOPSTR 2006*, pages 127–142, 2006.
31. W. Zielonka. Notes on finite asynchronous automata. *RAIRO–Theoretical Informatics and Applications*, 21:99–135, 1987.