Asynchronous Games over Tree Architectures

Blaise Genest
CNRS, IPAL UMI, Singapore

Hugo Gimbert, Anca Muscholl, Igor Walukiewicz
LaBRI, CNRS/Université Bordeaux, France

***Abstract*—We consider the task of controlling in a distributed way a Zielonka asynchronous automaton. Every process of a controller has access to its causal past to determine the next set of actions it proposes to play. An action can be played only if every process controlling this action proposes to play it. We consider reachability objectives: every process should reach its set of final states. We show that this control problem is decidable for tree architectures, where every process can communicate with its parent, its children, and with the environment. The complexity of our algorithm is l-fold exponential with l being the height of the tree representing the architecture. We show that this is unavoidable by showing that even for three processes the problem is EXPTIME-complete, and that it is non-elementary in general.**

I. Introduction

Constructing as well as verifying distributed systems is often a very demanding task. Distributed synthesis and control aim at providing a systematic way for constructing such systems from specifications. Although the challenge of full synthesis of distributed systems from a given specification is far too ambitious, there is a continuous effort in finding more powerful methods that address this challenge in more realistic settings.

We study in this paper a by now well-established model of distributed computation based on synchronization, namely *Zielonka's asynchronous automata*. Such an automaton is an asynchronous product of finite automata synchronizing on common actions. This simple yet rich model has solid theoretical foundations rooted in the theory of Mazurkiewicz traces. We consider the control problem for such automata: given a Zielonka automaton, a plant, find another Zielonka automaton, a controller, such that the product of the two satisfies a given specification. We show that this problem is decidable for reachability objectives on tree architectures. We also show that the complexity of this problem is bounded from below by a function that is a tower of exponentials of height proportional to the diameter of the communication graph.

Our problem can be seen as a variation of Church's problem. More than half a century ago, Church asked for an algorithm to construct devices transforming (infinite) sequences of input bits to (infinite) sequence of output bits in a way required by a specification [4]. Later Ramadge and Wonham proposed a different formulation where we are given a plant together with a specification and we are required to construct a controller such that the product of the controller with the plant satisfies the specification [21]. So control means restricting the behavior of the plant and synthesis is the particular case where the plant allows for every possible behavior.

In the setting of Ramadge and Wonham both the plant and the specification are finite automata. Pnueli and Rosner have proposed an extension of Church's setting by considering a set of processes working fully synchronously and exchanging messages through one slot communication channels [19]. The control version has also been extended to the distributed case by asking to construct several controllers, each with a different partial view of the plant [23], [22], [25], [1], [2]. In the problem we consider here we ask for just one controller, but both the plant and the controller are themselves distributed devices. In Figure 1 we have represented schematically different settings of synthesis and control problems.

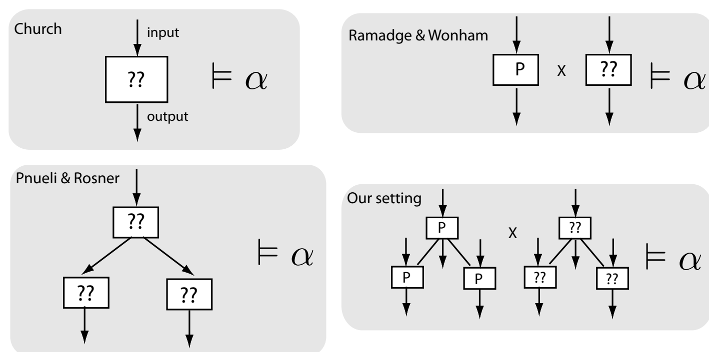

Fig. 1. Different formulations of synthesis/control problems

In short our control problem is as follows. We are given a Zielonka automaton over a fixed set of processes with fixed communication structure. Processes have local actions, that can be uncontrollable, as well as actions that are shared with some other process (binary synchronization actions), that are always controllable. Uncontrollable local actions represent inputs from the environment, controllable ones represent outputs of the system. Synchronization actions are used to gather information about the global state of the system. The synchronization actions define a communication graph, where nodes are processes and edges represent pairs of processes that can share some action. For a given set of final states the objective is to find a controller, that is a Zielonka automaton over the same set of processes and actions, such that every execution of the product of the plant and the controller brings eventually each process into a final state.

We show that our control problem is decidable when the communication graph is acyclic. The idea is simple. If the graph is acyclic and not totally disconnected then there is a leaf process r that communicates only with one other process q. We then make q simulate r thus reducing the number of processes. Repeating this argument we reduce the problem to a situation when the communication graph is totally disconnected, and this is easily solvable. Because the reduction uses a powerset construction, we obtain an algorithm whose complexity is a function that is a tower of exponentials of size proportional to the diameter of the graph. We show that this is essentially the best one can do. We prove that already for 3 processes the problem is EXPTIME-complete. We also give a family of control problems whose complexity is bounded from below by a tower of exponentials of height proportional to the diameter of the communication graph.

Our decidability result includes for example a client-server architecture where we have one server communicating with clients, and at the same time server and clients have their own interactions with the environment (cf. Figure 2). Our reduction method gives an EXPTIME algorithm solving the control problem for this architecture. Notice that since we have inputs at each process this architecture is very different from decidable architectures in the Pnueli & Rosner setting. This positive result is possible due to two factors. First, our model is asynchronous: between two successive synchronizations a client and the server can do a different number of actions. The second reason, that is probably even more important, concerns information flow. We explain this below.

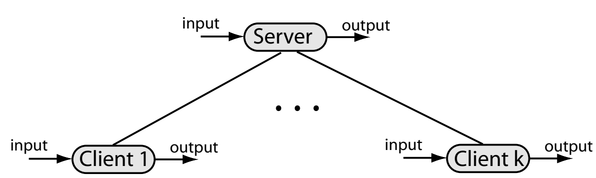

Fig. 2. Server/client architecture

The research effort put into Pnueli & Rosner setting of the distributed synthesis problem justifies the quest for other formulations. By now we understand that suitably using the interplay between specifications and an architecture, one can get undecidability results for most architectures rather easily. Yet the kinds of specifications that lead to undecidability are artificial, like: putting a constraint linking two disconnected parts of the system, or using an output channel to single out one input of unbounded length. Unfortunately, till now we do not know how to eliminate these artificial situations in an elegant way.

One important attempt to get a decidable framework of distributed synthesis is to change the way information is distributed in the system [7], [14]. This is the setting we consider here as well. In the framework of Pnueli and Rosner, every controller sees only its inputs and its outputs. In order to deduce some information about the global state of the system a controller can use only his knowledge about the architecture and the initial state of the system. In particular, controllers are not allowed to exchange additional information during communication. It is clear though that when we allow some transfer of information during communication, we give more power to controllers. Pushing the idea of information exchange to the limit, we obtain a model where two processes involved in a communication share all the information they have about the global state of the system. This point of view is not as unrealistic as it may seem at the first glance. It is rooted in the theory of Mazurkiewicz traces that studies Zielonka asynchronous automata with this kind of information transfer. A fundamental result of Zielonka [26] (see also [16], [9] for algorithmic improvements) implies a bound on the size of additional information that needs to be transferred during synchronization. In our terms, the theory of traces considers the case of synthesis for closed systems, i.e., systems without uncontrollable actions. Distributed synthesis with environment brings us to the setting we consider here. Similarly to Zielonka's Theorem, we give a bound on additional information that needs to be transferred. In case of the architecture from Figure 1 with each transfer between a client and the server we will need to add at most polynomially many bits with respect to the state space of the client.

Related work. The setting proposed by Pnueli and Rosner has been thoroughly investigated in past years. Results on multi-player games [18], [19] tell us that synthesis in this framework is undecidable, and [20] shows that synthesis w.r.t. properties expressed in LTL is decidable when the communication graph is a (directed) pipeline, with inputs allowed only at the first node. The paper [12] gives an automata-theoretic approach to solving pipeline architectures and at the same time extends the decidability results to CTL* specifications and variations of the pipeline architecture, like one-way ring architectures. The control setting of Ramadge and Wonham is investigated in [13] for local specifications, meaning that each process has its own, linear-time specification. The control problem for local specifications is decidable for pipelines with inputs at both endpoints. The result of [13] is complete in the sense that it shows that an architecture has a decidable control problem if and only if it is a sub-architecture of a clean pipeline. For instance, the 3 process pipeline with inputs on the first two processes is undecidable. The paper [6] proposes the notion of information fork as a uniform notion describing the existing (un)decidability results on distributed synthesis. The paper [8] goes beyond and considers the notion of well-connected architecture, attempting to characterize decidable external specifications.

The setting considered here has been proposed by Gastin, Lerman and Zeitoun [7]. Their model is *action-based*, meaning that actions decide if they are enabled or not. Here we prefer the *process-based* formulation, as it corresponds in a direct way to control in the sense of Ramadge and Wonham. Process-based formulation has been introduced by Madhusudan, Thiagarajan, Yang [14]. In [17] we analyze the relationship between the two versions of distributed control.

Compared with the setting of Pnueli and Rosner, our understanding of distributed synthesis with information exchange between controllers is still quite rudimentary: no undecidable case has been found, so it is possible that the problem is decidable in its full generality. Only two decidability results are known, both very different from our case. The first one [7] is based on a restriction on the alphabet of actions: games with reachability condition are decidable for co-graph alphabets. This restriction is not satisfied as soon as we have local actions for each process, and a process that can communicate with two other ones (a case for which we show here that the control problem is decidable). The second result [14] obtains decidability of the control problem by restricting the plant: roughly speaking, the restriction requires that if two processes do not synchronize during a fixed amount of time, then they will never synchronize again. The proof of [14] 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. The restriction on the form of the plant is crucial there since there are many very simple plants with decidable control problem but undecidable MSO-theory of the associated even structure.

Another approach to distributed synthesis is to distribute a centralized controller. This has been already proposed by Clarke and Emerson in their paper introducing CTL. In two recent papers [3], [10] some variants of asynchronous models are considered. In both papers, the setting is such that it is possible to distribute every centralized controller, sometimes by adding new synchronizations. This is impossible in our formulation.

Due to space constraints most proofs are ommitted. The appendix contains the full version of the paper.

II. Basic definitions and observations

Our control problem can be formulated in the same way as the Ramadge and Wonham control problem but using Zielonka automata instead of standard ones. We start by presenting Zielonka automata and an associated notion of concurrency. Then we briefly recall the Ramadge and Wonham formulation and our variant of it. Finally, we give a more convenient game based formulation of the problem.

A. Zielonka automata

Zielonka automata are simple parallel devices. Such an automaton is a parallel composition of several finite automata, denoted as *processes*, synchronizing on common 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.

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 $dom(a)$.

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* 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 $s \stackrel{a}{\longrightarrow} s'$ 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 set of words labeling runs of this sequential automaton that start from the initial state.

This definition has an important consequence. The location mapping dom defines in a natural way an independence relation I: two actions $a, b \in \Sigma$ are independent (written as $(a,b) \in I$) if they involve different processes, that is, if $dom(a) \cap dom(b) = \emptyset$. Notice that the order of execution of two independent actions $(a,b) \in I$ in 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 $u \sim_I v$ and $u \in L(\mathcal{A})$ then $v \in L(\mathcal{A})$, too.

The idea of describing concurrency by an independence relation on actions goes back to the late seventies, to Mazurkiewicz [15] and Keller [11] (see also [5]). An equivalence class $[w]_I$ of $\sim_I$ is called a Mazurkiewicz *trace*, it can be also viewed as labeled pomset of a special kind. Here, we will often refer to a trace using just a word w instead of writing $[w]_I$. As we have observed $L(\mathcal{A})$ is a sum of such equivalence classes. In other words it is *trace-closed*.

Example 2.1: Consider the following, very simple, example with processes $1, 2, 3$ in Figure 3. Process 1 has local actions a_0, a_1 and synchronization actions $c_{i,j}$ $(i, j = 0, 1)$ shared with process 2. Similarly, process 3 has local actions b_0, b_1 and synchronization actions $d_{i,j}$ $(i, j = 0, 1)$ shared with process 2. Each process is a finite automaton and the Zielonka automaton is the product to the three components synchronizing on common actions (cf. Figure 3, where the symbol $*$ denotes both values, 0 and 1, and i, j, k, l each take both values). We have for instance $(a_i, b_j) \in I$ and $(c_{i,j}, d_{k,l}) \notin I$. The

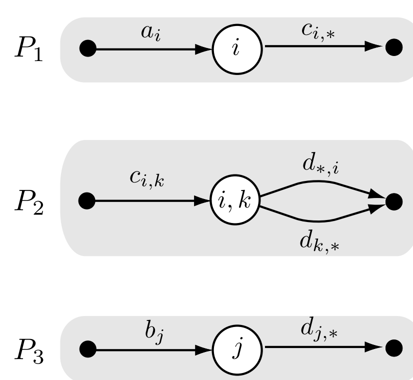

Fig. 3. An example of asynchronous automaton

final states are the rightmost states of each automaton. The automaton accepts traces of the form $a_i b_j c_{i,k} d_{j,l}$ with $i = l$ or $j = k$.

Since the notion of a trace can be formulated without a reference to an accepting device, it is natural to ask if the model of Zielonka automata is powerful enough. Zielonka's theorem says that this is indeed the case, hence these automata are a right model for the simple view of concurrency captured by Mazurkiewicz traces.

Theorem 2.2: [26] Let $dom : \Sigma \to (2^{\mathcal{P}} \setminus \{\emptyset\})$ be a distribution of letters. If a language $L \subseteq \Sigma^*$ is regular and trace-closed then there is a deterministic Zielonka automaton accepting L (of size exponential in the number of processes and polynomial in the size of the minimal automaton for L, see [9]).

One could try to use Zielonka's theorem directly to solve a distributed control problem. For example, one can start with the Ramadge and Wonham control problem, solve it, and if a solution happened to respect the required independence, then distribute it. Unfortunately, there is no reason for the solution to respect the independence. Even worse, the following, relatively simple, result says that it is algorithmically impossible to approximate a regular language by a language respecting a given independence relation.

Theorem 2.3: [24] It is not decidable if, given a distributed alphabet and a regular language $L \subseteq \Sigma^*$, there is a trace-closed language $K \subseteq L$ such that every letter from Σ appears in some word of K.

The condition on appearance of letters above is not crucial for the above undecidability result. Observe that we need some condition in order to make the problem nontrivial, since by definition the empty language is trace-closed.

B. The control problem

We can now formulate our control problem as a variant of the Ramadge and Wonham formulation. We will then provide an equivalent description of the problem in terms of games. While more complicated to state, this description is easier to work with.

Recall that in Ramadge and Wonham's control problem [21] we are given an alphabet Σ of actions partitioned into system and environment actions: $\Sigma^{sys} \cup \Sigma^{env} = \Sigma$. Given a plant P we are asked to find a controller C such that the product $P \times C$ satisfies a given specification. Here both the plant and the controller are finite deterministic automata over Σ. Additionally, the controller is required not to block environment actions, which in technical terms means that from every state of the controller there should be a transition on every action from Σ^{env}.

The definition of our problem will be the same with the difference that we will take Zielonka automata instead of standard finite automata. Consider a distributed alphabet $\langle \mathbb{P}, dom : \Sigma \to (2^{\mathbb{P}} \setminus \emptyset) \rangle$. We impose two simplifying assumptions. The first one is that all actions are at most binary: $|dom(a)| \leq 2$, for every $a \in \Sigma$. The second requires that all uncontrollable actions are local: $|dom(a)| = 1$, for every $a \in \Sigma^{env}$. So the first restriction says that we allow only binary synchronizations. It makes the technical reasoning much simpler. The second restriction reflects the fact that each process is modeled with its own, local environment.

Our control problem can be formulated as follows: Given a distributed alphabet $(\mathbb{P}, dom)$ as above and a Zielonka automaton $\mathcal{A}_p$, find a Zielonka automaton $\mathcal{A}_c$ over the same alphabet such that $\mathcal{A}_p \times \mathcal{A}_c$ satisfies a given specification. Additionally the controller is required not to block uncontrollable actions: from every state of $\mathcal{A}_c$ every uncontrollable action should be possible.

As in the original formulation, the role of the controller is to restrict the set of possible behaviours of the plant, but it is not allowed to restrict actions of the environment. The important point is that the controller should have the same distributed structure as the environment. The product of the two automata, that is just the standard product, means that plant and controller are totally synchronized, in particular communications between processes happen at the same time. Hence concurrency in the controlled system is the same as in the plant. The major difference between the controlled system and the plant is that the states carry the additional information computed by the controller.

Example 2.4: Reconsider the automaton in Figure 3 and assume that $a_i, b_j \in \Sigma^{env}$ are uncontrollable. So the controller needs to propose controllable actions $c_{i,k}$ and $d_{j,l}$, resp., in such a way that all processes reach their final state. In particular, process 2 should not block. At first sight this may seem impossible to guarantee, as it looks like process 1 needs to know what b_j process 3 has received, or process 3 needs to know about the a_i received by process 1. Nevertheless, a winning strategy exists. It consists of choosing $k = i$ and $l = 1 - j$: if $i = j$ then $k = j$, else $i = l$.

It will be more convenient to work with a game formulation of this problem. Instead of talking about controller we will talk about distributed strategy in a game between *system* and *environment*. A plant defines a game arena, with plays corresponding to initial runs of $\mathcal{A}$. Since $\mathcal{A}$ is deterministic, we can view a play as a word from $L(\mathcal{A})$ - or a trace, since $L(\mathcal{A})$ is trace-closed. Let $Plays(\mathcal{A})$ 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 about 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 during a communication with them. Formally, the p-view of a play u, denoted $view_p(u)$, is the smallest trace $[v]_I$ such that $u \sim_I vy$ and y contains no action from Σ_p. We write $Plays_p(\mathcal{A})$ for the set of plays that are p-views:

$$Plays_p(\mathcal{A}) = \{view_p(u) \mid u \in Plays(\mathcal{A})\}.$$

A *strategy for a process* p is a function $\sigma_p : Plays_p(\mathcal{A}) \to 2^{\Sigma_p}$, where $\Sigma_p = \{a \in \Sigma \mid a \in \Sigma^{sys}, p \in dom(a)\}$. We require in addition, for every $u \in Plays_p(\mathcal{A})$, that $\sigma_p(u)$ is a subset of the actions that are possible in the p-state reached on 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 $Plays(\mathcal{A}, \sigma)$, is the smallest set containing the empty play ε, and such that for every $u \in Plays(\mathcal{A}, \sigma)$:

1) if $a \in \Sigma^{env}$ and $ua \in Plays(\mathcal{A})$ then ua is in $Plays(\mathcal{A}, \sigma)$.
2) if $a \in \Sigma^{sys}$ and $ua \in Plays(\mathcal{A})$ then $ua \in Plays(\mathcal{A}, \sigma)$ provided that $a \in \sigma_p(view_p(u))$ *for all* $p \in dom(a)$.

Intuitively, the definition says that actions of the environment are always possible, whereas actions of the system are possible only if they are allowed by the strategies of all involved processes. Notice that in the distributed setting a process by itself cannot impose controllable actions (unless they are local): a controllable, shared action a *can* be chosen, if proposed by all owners. If some other action b is chosen instead, some owner of a can change his mind, and then a is not eligible anymore.

Before defining winning strategies, we need to introduce infinite plays that are consistent with a given strategy σ. Such plays can be seen as (infinite) traces associated with infinite, initial runs of $\mathcal{A}$ satisfying the two conditions of the definition of $Plays(\mathcal{A}, \sigma)$. We write $Plays^\infty(\mathcal{A}, \sigma)$ for the set of finite or infinite such plays. A play from $Plays^\infty(\mathcal{A}, \sigma)$ is also denoted as a *σ-play*.

A play $u \in Plays^\infty(\mathcal{A}, \sigma)$ is called *maximal*, if there is no action c such that $uc \in Plays^\infty(\mathcal{A}, \sigma)$. In particular u is maximal if $view_p(u)$ is infinite for every process p. Otherwise, if $view_p(u)$ is finite then p cannot have enabled local action (either controllable or uncontrollable). Moreover there should be no communication possible between any two processes with finite views in u.

In this paper we consider *local reachability* winning conditions. For this, every process has a set of target states $F_p \subseteq S_p$. We assume that states in F_p are *blocking*, that is they have no outgoing transitions. This means that if $(s_{dom(a)}, s'_{dom(a)}) \in \delta_a$ then $s_p \notin F_p$ for all $p \in dom(a)$.

Definition 2.5: The *control problem* for a plant $\mathcal{A}$ and a local reachability condition $(F_p)_{p\in\mathbb{P}}$ is to determine if there is a strategy $\sigma = (\sigma_p)_{p\in\mathbb{P}}$ such that every maximal trace $u \in Plays^\infty(\mathcal{A}, \sigma)$ ends in $\prod_{p\in\mathbb{P}} F_p$ (and is thus finite). Such traces and strategies are called *winning*.

This formulation of the problem is almost equivalent to the formulation with a plant and controller we have given in the beginning of this subsection. It is obvious that a controller defines a strategy. It is also true that a strategy defines a controller, but this controller may be infinite. We will show that for our control problem, if there is a strategy then there is one that can be translated to a finite controller.

As mentioned in the introduction, an interesting aspect of our formulation concerns the information exchanged between processes. In the setting of Pnueli and Rosner each process sees only its input and its output channels. For example, if the specification says that a channel should always contain the same letter then no information can be transmitted over this channel. In other words, the information exchanged can be totally controlled by specification. Once we adopt the Ramadge and Wonham formulation with Zielonka automata, the amount and type of exchanged information is determined by the controller. In our game formulation we use views that amount to maximal possible information a process can have. So in our model after a synchronization the two processes have the same (partial) knowledge about the global state of the system.

We do not know if this control problem is decidable in general. In this paper we put one restriction on possible communications between processes expressed in terms of communication graph defined below.

Definition 2.6: A distributed alphabet (Σ, dom) with unary and binary actions defines an undirected graph $\mathcal{CG}$ with node set $\mathbb{P}$ and edges $\{p, q\}$ if there exists $a \in \Sigma$ with $dom(a) = \{p, q\}$, $p \neq q$. Such a graph is called *communication graph*.

To sum up: in this paper we consider the Ramadge and Wonham formulation of the control problem but using Zielonka automata instead of standard ones. As specifications we consider reachability properties. We show that this problem is decidable for acyclic communication graphs (Theorem 3.11). We also provide a tight complexity bound (Theorem 4.4).

III. The upper bound for control problem for acyclic graphs

Let us fix in this section a distributed alphabet (Σ, dom). According to Definition 2.6 the alphabet determines a communication graph $\mathcal{CG}$. We assume that $\mathcal{CG}$ is acyclic and has at least one edge. This allows us to choose a leaf $r \in \mathbb{P}$ in $\mathcal{CG}$, with $\{q, r\}$ an edge in $\mathcal{CG}$. Starting from a control problem with input $\mathcal{A}$, $(F_p)_{p\in\mathbb{P}}$ we define below a control problem over the smaller (acyclic) graph $\mathcal{CG}' = \mathcal{CG}_{\mathbb{P}\setminus\{r\}}$. The construction will be an exponential-time reduction from the control problem over $\mathcal{CG}$ to a control problem over $\mathcal{CG}'$. If we represent $\mathcal{CG}$ as a tree of depth l then applying this construction iteratively we will get an l-fold exponential algorithm to solve the control problem for $\mathcal{CG}$ architecture.

The main idea of the reduction is simple: process q simulates the behavior of process r. The reason why a simulation can work is that after each synchronization between q and r, the views of both processes are identical, and between two such synchronizations r evolves locally. But the construction is more delicate than this simple description suggests, and needs some preliminary considerations about winning strategies.

Some preparatory lemmas: We start with some lemmas showing how to restrict the winning strategies. The first one holds for arbitrary communication graphs, whereas the second one relies on the fact the r is a leaf in $\mathcal{CG}$. For $p, q \in \mathbb{P}$ let $\Sigma_{p,q} = \{a \in \Sigma \mid dom(a) = \{p, q\}\}$. So $\Sigma_{p,q}$ is the set of synchronization actions between p and q. Moreover $\Sigma_{p,p}$ is just the set of local actions of p. We write Σ_p^{loc} instead of $\Sigma_{p,p}$ and $\Sigma_p^{com} = \Sigma_p \setminus \Sigma_p^{loc}$. The first lemma says that a winning strategy can be assumed to propose either a local action, or some communication actions.

Lemma 3.1: If there exists some winning strategy for $\mathcal{A}$, then there is one, say σ, such that for every process p and every σ-play $u \in Plays_p(\mathcal{A})$ with $X = \sigma_p(u)$, we have one of the following: $X = \{a\}$ for some $a \in \Sigma_p^{loc}$ or $X \subseteq \Sigma_p^{com}$.

The following definition captures all the possible evolutions of the leaf process r without communication with its parent process q. For an initial run u of $\mathcal{A}$ we denote by $state_p(u)$ the p-state reached by $\mathcal{A}$ on u.

Definition 3.2: Given a strategy σ and a play u, the set of all possible outcomes of a local play on r before its next communication is:

$$\begin{aligned} Sync_r^\sigma(u) = \{(s_r, A) \mid \exists x \in (\Sigma_r^{loc})^* \,.\, & ux \text{ is a } \sigma\text{-play}, \\ & state_r(ux) = s_r, \\ & \sigma_r(view_r(ux)) = A \subseteq \Sigma_{q,r}\}\,. \end{aligned}$$

Observe that if σ allows r to reach a final state s_r from u without communication, then $(s_r, \emptyset) \in Sync_r^\sigma(u)$. This is so, since final states are assumed to be blocking.

The lemma below talks about the strategy of process q, the parent of the leaf process r. It says that when the strategy offers communication, then it does so either with r exclusively, or only with other processes.

Lemma 3.3: If there exists some winning strategy for $\mathcal{A}$, then there is one, say σ, such that for every σ-play $u \in Plays_q(\mathcal{A})$ with $X = \sigma_q(u)$, we have one of the following: $X = \{a\}$ for some $a \in \Sigma_q^{loc}$, or $X \subseteq \Sigma_{q,r}$ or $X \subseteq \Sigma_q^{com} \setminus \Sigma_{q,r}$.

For the game reduction we need to precalculate all possible sets $Sync_r^\sigma$. These sets will be actually of the special form described below.

Definition 3.4: Let s_r be a state of r. We say that $T \subseteq S_r \times \mathcal{P}(\Sigma_{qr})$ is an *admissible plan in* s_r if there is a play u with $state_r(u) = s_r$, and a (not necessarily winning) strategy σ such that $T = Sync_r^\sigma(u)$, and one of the following holds:

- $A \neq \emptyset$ for every $(t_r, A) \in T$, or
- $t_r \in F_r$ and $A = \emptyset$ for every $(t_r, A) \in T$.

In the second case T is called a *final plan*.

It is not difficult to see that we can compute the set of all admissible plans, since this just amounts to solve a reachability game on process r.

Lemma 3.5 below allows to deduce that the sets $Sync_r^\sigma$ are admissible plans whenever σ is winning.

Lemma 3.5: If σ is a winning strategy satisfying Lemma 3.3 then for every σ-play u in $\mathcal{A}$ we have:

1) if there is some σ-play uy with $y \in (\Sigma \setminus \Sigma_r)^*$ and $state_q(uy) \in F_q$ then $Sync_r^\sigma(u)$ is a final plan;
2) if there is some σ-play uy with $y \in (\Sigma \setminus \Sigma_r)^*$, $\sigma_q(uy) = B \subseteq \Sigma_{q,r}$, and $B \neq \emptyset$ then for every $(t_r, A) \in Sync_r^\sigma(u)$ we have $B \cap A \neq \emptyset$.

In particular, $Sync_r^\sigma(u)$ is always an admissible plan.

The new plant $\mathcal{A}'$. We are now ready to define the reduced plant $\mathcal{A}'$ that is the result of eliminating process r. Let $\mathbb{P}' = \mathbb{P} \setminus \{r\}$. We have $\mathcal{A}' = \langle \{S'_p\}_{p \in \mathbb{P}'}, s'_{in}, \{\delta'_a\}_{a \in \Sigma'} \rangle$ where the components will be defined below.

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

$$\langle s_q, s_r \rangle\,, \quad \langle s_q, T \rangle\,, \quad \langle s_q, T, B \rangle\,,$$

where $s_q \in S_q, s_r \in S_r$, $T \subseteq S_r \times \mathcal{P}(\Sigma_{q,r})$ is an admissible plan, $B \subseteq \Sigma_{q,r}$. The new initial state for q is $\langle (s_{in})_q, (s_{in})_r \rangle$.

For every $p \neq q$, we let $S'_p = S_p$ and $F'_p = F_p$. The local winning condition for q becomes

$$F'_q = F_q \times F_r \cup \{\langle s_q, T \rangle \mid s_q \in F_q, \text{ and } T \text{ is a final plan}\}.$$

The set of actions Σ' is $\Sigma \setminus \Sigma_r$, plus additional local q-actions that we introduce below. All transitions δ_a with $dom(a) \cap \{q, r\} = \emptyset$ are as in $\mathcal{A}$. Regarding q we have the following transitions:

1) If not in a final state then process q chooses an admissible plan:
$$\langle s_q, s_r \rangle \xrightarrow{ch(T)} \langle s_q, T \rangle,$$
where T is an admissible plan in s_r, and $\langle s_q, s_r \rangle \notin F_q \times F_r$.
2) Local action of q:
$$\langle s_q, T \rangle \xrightarrow{a} \langle s'_q, T \rangle, \quad \text{ if } s_q \xrightarrow{a} s'_q \text{ in } \mathcal{A}\,.$$
3) Synchronization between q and $p \neq r$:
$$(\langle s_q, T \rangle, s_p) \xrightarrow{b} (\langle s'_q, T \rangle, s'_p), \quad \text{if } (s_q, s_p) \xrightarrow{b} (s'_q, s'_p)$$
4) Synchronization between q and r. Process q declares the communication actions with r:
$$\langle s_q, T \rangle \xrightarrow{ch(B)} \langle s_q, T, B \rangle, \qquad \text{if } \; B \subseteq \Sigma_{q,r}$$
when s_q is not final, T not a final plan, and for every $(t_r, A) \in T$ we have $A \cap B \neq \emptyset$.
Then the environment can choose the target state of r and a synchronization action $a \in \Sigma_{q,r}$:
$$\langle s_q, T, B \rangle \xrightarrow{(a,t_r)} \langle s'_q, s'_r \rangle \qquad \text{if } (s_q, t_r) \xrightarrow{a} (s'_q, s'_r) \text{ in } \mathcal{A}$$
for every (a, t_r) such that $(t_r, A) \in T$ for some A, and $a \in A \cap B$. Notice that the complicated name of the action (a, t_r) is needed to ensure that the transition is deterministic.

To summarize the new actions of process q in plant $\mathcal{A}'$ are:

- $ch(T) \in \Sigma^{sys}$, for every admissible plan T,
- $ch(B) \in \Sigma^{sys}$, for each $B \subseteq \Sigma_{q,r}$,
- $(a, t_r) \in \Sigma^{env}$ for each $a \in \Sigma_{q,r}, t_r \in S_r$.

Before showing that this construction is correct we will provide a translation from plays in $\mathcal{A}$ to plays in $\mathcal{A}'$. A (finite or infinite) play u in $\mathcal{A}$ is a trace that will be convenient to view as a word of the form

$$u = y_0 x_0 a_1 \;\cdots\; a_i y_i x_i \; a_{i+1} \ldots$$

where for $i \in \mathbb{N}$ we have that: $a_i \in \Sigma_{q,r}$ is communication between q and r; $x_i \in (\Sigma_r^{loc})^*$ is a sequence of local actions of r; and $y_i \in (\Sigma \setminus \Sigma_r)^*$ is a sequence of actions of other processes than r. Note that x_i, y_i are concurrent, for each i. We will write $u|_{a_i}$ for the prefix of u ending in a_i. Similarly $u|_{y_i}$ for the prefix ending with y_i; analogously for x_i.

With a word u as above we will associate the word

$$\chi(u) = ch(T_0)y_0\, ch(B_0)(a_1, t_r^1)\ldots \\ (a_i, t_r^i)\, ch(T_i)\; y_i\, ch(B_i)(a_{i+1}, t_r^{i+1})\ldots$$

where for every $i = 0, 1, \ldots$:

- $T_i = Sync_r^{\sigma}(u|_{a_i})$ and $T_0 = Sync_r^{\sigma}(\varepsilon)$;
- $B_i = \sigma_q(view_q(u|_{y_i}))$;
- $t_r^i = state_r(u|_{x_i})$.

In Figure 4 we have pictorially represented which parts of u determine which parts of $\chi(u)$.

The next lemma follows directly from the definition of the reduction.

Lemma 3.6: If u ends in a letter from $\Sigma_{q,r}$ then we have the following

- $state_q(\chi(u)) = \langle state_q(u), state_r(u)\rangle$.
- $state_p(\chi(u)y) = state_p(uy)$ for every $p \neq q$ and $y \in (\Sigma \setminus \Sigma_{q,r})^*$.
- $state_q(\chi(u)\, ch(T)y) = \langle state_q(uy), T\rangle$ for every $y \in (\Sigma \setminus \Sigma_{q,r})^*$.
- $state_q(\chi(u)\, ch(T)y\, ch(B)) = \langle state_q(uy), T, B\rangle$ for every $y \in (\Sigma \setminus \Sigma_{q,r})^*$.

From σ in $\mathcal{A}$ to σ' in $\mathcal{A}'$. We are now ready to define σ' from a winning strategy σ. We assume that σ satisfies the property stated in Lemma 3.3. We will define σ' only for certain plays and then show that this is sufficient.

Consider u' such that $u' = \chi(u)$ for some σ-play u ending in a letter from $\Sigma_{q,r}$. We have:

- If $state_q(u') \notin F_q$ then $\sigma'_q(view_q(u')) = \{ch(T)\}$ where $T = Sync_r^{\sigma}(u)$.
- For every process $p \neq q$ we put $\sigma'_p(view_p(u'\, ch(T)y)) = \sigma_p(view_p(uy))$ for $y \in (\Sigma \setminus \Sigma_{q,r})^*$.
- For $y \in (\Sigma \setminus \Sigma_{q,r})^*$ and $B = \sigma_q(view_q(uy))$ we define

$$\sigma'_q(view_q(u'\, ch(T)y)) = \begin{cases} B & \text{if } B \cap \Sigma_{q,r} = \emptyset \\ \{ch(B)\} & \text{if } B \subseteq \Sigma_{q,r} \end{cases}$$

- $\sigma'_q(view_q(u'\, ch(T)y\, ch(B))) = \emptyset$.

Observe that in the last case the strategy proposes no move as there are only moves of the environment from a position reached on a play of this form.

Lemma 3.7: If σ is a winning strategy for $\mathcal{A}, (F_p)_{p\in\mathbb{P}}$ then σ' is a winning strategy for $\mathcal{A}', (F'_p)_{p\in\mathbb{P}'}$.

From σ' in $\mathcal{A}'$ to σ in $\mathcal{A}$. From a strategy $\sigma' = (\sigma'_p)_{p\in\mathbb{P}'}$ for $\mathcal{A}'$ we define a strategy $\sigma = (\sigma_p)_{p\in\mathbb{P}}$ for $\mathcal{A}$. We assume that σ' satisfies Lemma 3.3. We consider u ending in an action from $\Sigma_{q,r}$ such that $\chi(u)$ is a σ'-play. First, for every $p \neq q, r$ and every $y \in (\Sigma \setminus \Sigma_r)^*$ we set

$$\sigma_p(view_p(uy)) = \sigma'_p(view_p(\chi(u)y)).$$

If $state_q(\chi(u))$ is not final then $\sigma'(\chi(u)) = \{ch(T)\}$ for some admissible plan T in state $state_r(\chi(u))$. This means that $T = Sync_r^{\rho}(u)$ for some strategy ρ. In this case:

- for every $x \in (\Sigma_r^{loc})^*$ we set $\sigma_r(ux) = \rho_r(ux)$;
- for every $y \in (\Sigma \setminus \Sigma_r)^*$ we consider $X = \sigma'_q(view_q(\chi(u)\, ch(T)y))$ and set

$$\sigma_q(view_q(uy)) = \begin{cases} B & \text{if } X = \{ch(B)\} \\ X & \text{otherwise} \end{cases}$$

Lemma 3.8: If σ' is a winning strategy for $\mathcal{A}', (F'_p)_{p\in\mathbb{P}'}$ then σ is a winning strategy for $\mathcal{A}, (F_p)_{p\in\mathbb{P}}$.

Together the lemmas 3.7 and 3.8 show the correctness of our reduction:

Theorem 3.9: Let r be the chosen leaf process with $\mathbb{P}' = \mathbb{P} \setminus \{r\}$ and q its neighbor process. Then the system has a winning strategy for $\mathcal{A}, (F_p)_{p\in\mathbb{P}}$ iff it has one for $\mathcal{A}', (F'_p)_{p\in\mathbb{P}'}$. All the components of $\mathcal{A}'$ are identical to those of $\mathcal{A}$, apart that for the process q. The size of q in $\mathcal{A}'$ is $\mathcal{O}(M_q 2^{M_r 2^{|\Sigma_{qr}|}})$, where M_q and M_r are the sizes of processes q and r in $\mathcal{A}$, respectively.

Remark 3.10: The bound of the size of the plant $\mathcal{A}'$ can be improved to $\mathcal{O}(M_q 2^{M_r|\Sigma_{qr}|})$ by observing that we can restrict the notion of admissible plans to (partial) functions from S_r into $\mathcal{P}(\Sigma_{q,r})$. That is, one does not need to consider different sets of communication actions for the same state in S_r.

Coming back to the example from Figure 2 of a server with k clients. Applying our reduction k times we reduce out all the clients and obtain the single process plant whose size is $M_s 2^{(M_1+\cdots+M_k)2^c}$ where M_s is the size of the server, M_i is the size of client i, and c is the maximal number of communication actions between a client and the server.

Theorem 3.11: The control problem for distributed alphabets whose communication graph is acyclic, is decidable. There is an algorithm for solving the problem whose working time is bounded by a tower of exponentials of height equal to the diameter of the graph.

Our reduction algorithm can be actually used to compute a (finite-state) controller, as shown below.

Corollary 3.12: There is an algorithm which solves the control problem for distributed alphabets whose communication graph is acyclic and if the answer is positive, the algorithm outputs a controller satisfying the following property: For every process p and every state s of the controller $\mathcal{A}_c$, the set of actions allowed for process p in state s is the set of all uncontrollable local actions plus:

- either a unique controllable local actions,
- or a set of controllable actions shared with a unique neighbour q of p.

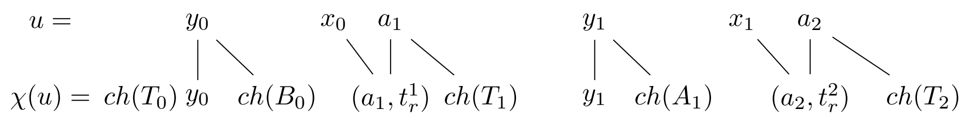

Fig. 4. Definition of $\chi(u)$

IV. The lower bound

In this section we show that the complexity of the distributed control problem grows as a tower of exponentials function with respect to the size of the diameter of the communication graph. Before presenting the general construction we illustrate the proof idea on the simplest non-trivial acyclic communication graph, consisting of a line of three processes. We show that the control problem here is EXPTIME-complete.

Proposition 4.1: For fixed distributed alphabet, the control problem for the communication graph 1 ——— 2 ——— 3 is EXPTIME-complete.

Proof: The upper bound follows from Theorem 3.9. We apply twice the reduction with process 2 first simulating process 1, then process 3. This yields a control problem on one process of exponential size (since the action set is fixed). So this amounts just to solve a reachability game, and therefore we get the EXPTIME upper bound.

For the lower bound we simulate an alternating polynomial space Turing machine M on input w. We assume that M has a unique accepting, blocking configuration (say with blank tape, head leftmost). The goal now is to let processes $1, 3$ guess an accepting computation tree of M on w. The environment will be able to choose a branch in this tree and challenge each proposed configuration. Process 2 will be used to validate tests initiated by the environment. If a test reveals an inconsistency, process 2 blocks and the environment wins. To summarize the idea of the construction, processes 1 and 3 generate sequences of configurations (encoded by local actions), separated by action \$ and $\overline{\$}$, respectively, shared with process 2. Both start with the initial configuration of M on w. Transitions from existential states are chosen by the plant, and those from universal ones by the environment. At a given time, process 1 has generated the same number or one more configuration than process 3. In the first case, the environment can check that it is the same configuration; and in the second, it can check that it is the successor configuration. In this way, 1 and 3 need to generate the same branch of the run tree.

A computation of M with space bound n is a sequence $C_0 \vdash C_1 \vdash \cdots \vdash C_N$, where each configuration C_i is encoded as a word from $\Gamma^*(Q\times\Gamma)\Gamma^*$ of length n. Since M is alternating, its acceptance is expressed by the existence of a tree of accepting computations.

Processes 1 starts by generating the initial configuration on w, followed by a synchronization symbol \$ with process 2. After this, process 1 generates a sequence of configurations separated by \$. When generating a configuration, process 1 remembers M's state q and the symbol A under the head. All transitions so far are controllable. After generating \$ process 1 goes into a state where the outgoing transitions are labeled by M's transitions on (q, A) (if the configuration was not blocking). These transitions are controllable if q is existential, and uncontrollable if q is universal. The transition chosen, either by the plant or the environment, is stored in the state up to the next synchronization symbol. Finally, if the current configuration is final then process 1 synchronizes with 2 on $\$_F$ (instead of \$) and goes into an accepting state.

The description is similar for process 3, with $\overline{\Gamma}, \overline{Q}, \overline{\$}, \overline{\$}_F$ instead of $\Gamma, Q, \$, \$_F$. Finally, process 2 has two main states, eq and $succ$, with transitions $eq \xrightarrow{\overline{\$}} succ$ and $succ \xrightarrow{\$} eq$. From state eq it can go to an accepting state after reading $\overline{\$}_F$ followed by $\$_F$.

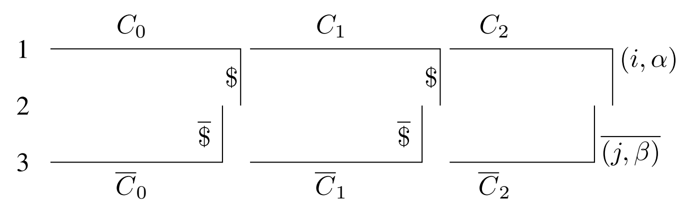

Fig. 5. Environment chooses positions i, j in $C_P, \overline{C}_P$ with $P = 2$. System wins iff $\alpha = \beta$ or $i \neq j$.

The environment can initiate 2 kinds of tests: equality and successor test. The equality test checks that $C_P = \overline{C}_P$ and the successor test checks that $C_P \vdash \overline{C}_{P+1}$.

For the equality test, the environment can choose a position i within C_P and a position j in $\overline{C}_P$. Formally, for each (controllable) outgoing transition $s \xrightarrow{\alpha}$ of process 1 with $\alpha \in \Gamma \cup (Q \times \Gamma)$ there is a transition $s \xrightarrow{(\downarrow,\alpha)} (\downarrow, i, \alpha)$ with $(\downarrow, \alpha)$ uncontrollable. The target state $(\downarrow, i, \alpha)$ records the tape position i (known from s) and the tape symbol α. In state $(\downarrow, i, \alpha)$ process 1 synchronizes with 2 on action $(\downarrow, i, \alpha)$, and then stops (accepting). The same for process 3 with uncontrollable actions $\overline{(\downarrow, \beta)}$, and synchronization action $\overline{(\downarrow, j, \beta)}$.

From state eq process 2 can perform a synchronization $\overline{(\downarrow, j, \beta)}$ with process 3 and then one with process 1 on any $(\downarrow, i, \alpha)$, provided $i \neq j$ or $\alpha = \beta$, and then accept. This is the case where the environment has chosen positions on both lines 1 and 3 (see Figure 5). If the environment has chosen a test transition in C_P but not in $\overline{C}_P$ (or vice-versa), process 2 will accept (and stop), too.

The successor test is similar, it consists in choosing a position within C_P and one within $\overline{C}_{P+1}$. The information checked by process 2 includes the symbols $\alpha_-, \alpha, \alpha_+$ of C_P at positions $i-1, i, i+1$ resp., so process 1 goes on transition $\overline{(\searrow, \alpha)}$ into a state of the form $(i, \alpha, \alpha_-, \alpha_+)$. In state $\overline{t}$

process 2 can perform a synchronization on $(\searrow, i, \alpha, \overline{\alpha_-, \alpha_+})$ with process 1, and then one with process 3 on $\overline{(\searrow, j, \beta)}$, provided $i \neq j$ or the symbols $\alpha_-, \alpha, \alpha_+$ are inconsistent with the new middle symbol β according to M's transition relation.

The reader may notice that we need to guarantee that the universal transitions chosen by the environment are the same, for processes 1 and 3. This can be enforced by communicating the transitions with actions $\$, \overline{\$}$ to process 2, who is in charge of checking. Moreover, note that the action alphabet above is not constant, in particular it depends on n. This can be fixed by replacing each action of type $(\downarrow, i, \alpha)$ (or alike) by a sequence of synchronization actions where i is transmitted bitwise. By alternating the bits transmitted by 1 and 3, respectively, process 2 can still compare indices i, j.

Note also that configurations $C_P, \overline{C}_P$ are generated in parallel, and so are C_P and $\overline{C}_{P+1}$. This is crucial for the correctness, as we show in the lemma below. ■

Lemma 4.2: The control problem defined in Proposition 4.1 has a winning strategy if and only if M accepts w.

A. Lower bound: general construction

Our main objective now is to show how using a communication architecture of diameter l one can code a counter able to represent numbers of size $Tower(2, l)$ (with $Tower(n, l) = 2^{Tower(n,l-1)}$ and $Tower(n, 1) = n$). Then an easy adaptation of the construction will allow to code computations of Turing machines with the same space bound as the capabilities of counters.

We fix n and will be first interested to define n-counters. Let $\Sigma_i = \{a_i, b_i\}$ for $i = 1, \ldots, n$. We will think of a_i as 0 and b_i as 1, mnemonically: 0 is round and 1 is tall. Let $\Sigma_i^{\#} = \Sigma_i \cup \{\#_i\}$ be the alphabet extended with an end marker.

A 1-counter is just a letter from Σ_1 followed by $\#_1$. The value of a_1 is 0, and the one of b_1 is 1. Following this intuition we write $(1 - c)$ to denote b if $c = a$ and vice versa.

An $(l + 1)$-*counter* is a word

$$x_0 u_0 x_1 u_1 \cdots x_{k-1} u_{k-1} \#_{l+1} \tag{1}$$

where $k = Tower(2, l)$ and for every i, letter $x_i \in \Sigma_{l+1}$ and u_i is an l-counter with value i. The value of the above $(l + 1)$-counter is $\sum_{i=0,\ldots,k} x_i 2^i$. The end marker $\#_{l+1}$ will be convenient in the construction that follows. An *iterated* $(l + 1)$-*counter* is a nonempty sequence of $(l + 1)$-counters.

For every l we will define a plant $\mathcal{C}^l$ such that the winning strategy for the system in $\mathcal{C}^l$ will need to produce an iterated l-counter.

The case $l = 1$ is easy. Suppose that we have already constructed $\mathcal{C}^l$. We want now to define $\mathcal{C}^{l+1}$, a plant producing an iterated $(l + 1)$-counter, i.e., a sequence of l-counters with values $0, 1, \ldots, (Tower(2, l)-1), 0, 1, \ldots$. We assume that the communication graph of $\mathcal{C}^l$ has the distinguished root process r_l. Process r_l is in charge of generating an iterated l-counter. From $\mathcal{C}^l$ we will construct two plants $\mathcal{D}^l$ and $\overline{\mathcal{D}}^l$, over disjoint sets of processes. The plant $\mathcal{D}^l$ is obtained by adding a new root process r_{l+1} that communicates with r_l, similarly for the plant $\overline{\mathcal{D}}^l$ with root process $\overline{r_{l+1}}$. The plant $\mathcal{C}^{l+1}$ will be the composition of $\mathcal{D}^l$ and $\overline{\mathcal{D}}^l$ with a new *verifier process* that we name $\mathcal{V}_{l+1}$. The root process of the communication graph of $\mathcal{C}^{l+1}$ will be r_{l+1}. The schema of the construction is presented in Figure 6. Process r_{l+1}, as well as $\overline{r_{l+1}}$, are in charge of generating an iterated $(l + 1)$-counter. That they behave indeed this way is guaranteed by a construction similar to the one of Proposition 4.1, with the help of the verifier $\mathcal{V}_{l+1}$: the environment gets a chance of challenging each l-counter of the sequence of r_{l+1} (and similarly for $\overline{r_{l+1}}$). These challenges correspond to two types of tests, equality and successor. If there is an error in one of these sequences then the environment can place a challenge and win. Conversely, if there is no error no challenge of the environment can be successful; this means then that the sequences of l-counters have correct values $0, 1, \ldots, (Tower(2, l) - 1), 0, 1, \ldots$. The detailed construction can be found in the appendix.

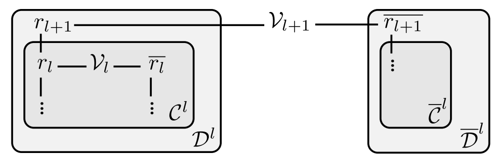

Fig. 6. Architecture of the plant $\mathcal{C}^{l+1}$

Proposition 4.3: For every l, the system has a winning strategy in $\mathcal{C}^l$. For every such winning strategy σ, if we consider the unique σ-play without questions then its projection on $\bigcup_{i=1,\ldots,l} \Sigma_i^{\#}$ is an iterated l-counter.

Theorem 4.4: Let $l > 0$. There is an acyclic architecture of diameter $4l + 1$ and with $(2^{l+2} - 3)$ processes such that the space complexity of the control problem for it is $\Omega(Tower(n, l))$-complete.

Proof: First observe that the plant $\mathcal{C}^l$ has $(2^{l+1} - 3)$ processes and diameter $4l - 3$. It is straightforward to make the l-counter count till $Tower(n, l)$ and not to $Tower(2, l)$ as we have done in the above construction. For this it is enough to make the 1-counter count to n instead of just to 2.

We will simulate space bounded Turing machines. Take a machine M and a word w of length n. We want to reduce the problem of deciding if w is accepted by M to the problem of deciding if the system has a winning strategy for a plant $\mathcal{C}(M, w)$ of size polynomial in the sizes of M and w.

A $Tower(n, l)$ size configuration can be encoded by an $(l + 1)$-counter. In an iterated $(l + 1)$-counter we can encode a sequence of such configurations. The plant $\mathcal{C}(M, w)$ is obtained by a rather straightforward modification of the construction of $\mathcal{C}^{l+1}$. Instead of ensuring that the value of the first counter is 0, it needs to ensure that it represents the initial configuration. Instead of ensuring that the two successive counters represent two successive numbers, it needs to ensure that they represent two successive configurations. Using Proposition 4.3, the problem of deciding if a $Tower(n, l)$-space bounded Turing machine M accepts w is polynomially reducible to the problem of deciding if the system has a winning strategy in the so obtained $\mathcal{C}(M, w)$. The size of $\mathcal{C}(M, w)$ is exponential in l and polynomial in M, w, n. The game can be constructed in the time proportional to its size. ■

V. Conclusions

The distributed synthesis problem is a very difficult and at the same time promising problem, since distributed systems are intrinsically complex to construct. Among many possible settings we have looked at the one that is at the same time pure and realistic: we have taken a simple and well established model for concurrent systems and put it into a classical framework of control. We could have done the same in the Church setting but then we would need to talk about logics for trace closed properties. The setting of Ramadge and Wonham allows to avoid this, since parts of the specification are hidden in the plant. In our opinion Zielonka asynchronous automata are at least as interesting as the fully synchronous model of Pnueli and Rosner. Of course, in the long run it would be desirable to consider even richer models, say 1-safe Petri nets and beyond, but even asynchronous automata are challenging enough at present.

In our model we have insisted that control does not introduce new synchronizations: it does not reduce parallelism of the controlled system. It seems undesirable to have a solution that removes completely parallelism from the system. Even if one accepts to limit parallelism, it is not clear how to measure how much of it is left afterwards.

The choice of transmitting additional information with communication is a consequence of the definitions we have adopted. We think that it is interesting from a practical point of view. It is also interesting theoretically since it allows to avoid simple, and unrealistic, reasons for undecidability.

Our lower bound result is somehow surprising. Since we have perfect information sharing, all the complexity has to be hidden in the uncertainty of what other processes are doing in parallel. The proof shows that even with three processes this uncertainty can be used to encode complex problems.

Of course the general case, the one without restriction to acyclic communication graphs, is an important open problem. A more immediate task is to examine other conditions than reachability. The reduction we have used to obtain decidability is rather delicate and cannot be easily extended to, say, Büchi conditions.

References

[1] A. Arnold, A. Vincent, and I. Walukiewicz. Games for synthesis of controllers with partial observation. *Theor. Comput. Sci.*, 303(1):7–34, 2003.

[2] A. Arnold and I. Walukiewicz. Nondeterministic controllers of nondeterministic processes. In *Logic and Automata*, volume 2 of *Texts in Logic and Games*, pages 29–52. Amsterdam University Press, 2007.

[3] T. Chatain, P. Gastin, and N. Sznajder. Natural specifications yield decidability for distributed synthesis of asynchronous systems. In *Proceedings of SOFSEM'09*, volume 5404 of *LNCS*, pages 141–152. Springer, 2009.

[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] B. Finkbeiner and S. Schewe. Uniform distributed synthesis. In *Proceedings of LICS'05*, pages 321–330. IEEE, 2005.

[7] P. Gastin, B. Lerman, and M. Zeitoun. Distributed games with causal memory are decidable for series-parallel systems. In *FSTTCS*, volume 3328 of *LNCS*, pages 275–286. Springer, 2004.

[8] P. Gastin, N. Sznajder, and M. Zeitoun. Distributed synthesis for well-connected architectures. *Formal Methods in System Design*, 34(3):215–237, 2009.

[9] B. Genest, H. Gimbert, A. Muscholl, and I. Walukiewicz. Optimal Zielonka-type construction of deterministic asynchronous automata. In *Proceedings ICALP'10*, volume 6199 of *LNCS*. Springer, 2010.

[10] G. Katz, D. Peled, and S. Schewe. Synthesis of distributed control through knowledge accumulation. In *Proceedings of CAV'11*, volume 6806 of *LNCS*, pages 510–525. Springer, 2011.

[11] R. M. Keller. Parallel program schemata and maximal parallelism I. Fundamental results. *Journal of the Association of Computing Machinery*, 20(3):514–537, 1973.

[12] O. Kupferman and M. Vardi. Synthesizing distributed systems. In *Proc. 16th IEEE Symp. on Logic in Computer Science*, 2001.

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

[14] P. Madhusudan, P. S. Thiagarajan, and S. Yang. The MSO theory of connectedly communicating processes. In *Proceedings of FSTTCS'05*, volume 3821 of *LNCS*, pages 201–212. Springer, 2005.

[15] A. Mazurkiewicz. Concurrent program schemes and their interpretations. DAIMI Rep. PB 78, Aarhus University, Aarhus, 1977.

[16] M. Mukund and M. A. Sohoni. Keeping Track of the Latest Gossip in a Distributed System. *Distributed Computing*, 10(3):137–148, 1997.

[17] A. Muscholl, I. Walukiewicz, and M. Zeitoun. A look at the control of asynchronous automata. In *Perspectives in Concurrency Theory, IARCS-Universities*. Universities Press, 2009.

[18] G. L. Peterson and J. H. Reif. Multi-person alternation. In *Proc. IEEE FOCS*, pages 348–363, 1979.

[19] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proc. ACM POPL*, pages 179–190, 1989.

[20] 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.

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

[22] S. Ricker and K. Rudie. Know means no: Incorporating knowledge into discrete-event control systems. *IEEE Transactions on Automatic Control*, 45(9), 2000.

[23] K. Rudie and W. Wonham. Think globally, act locally: Decentralized supervisory control. *IEEE Trans. on Automat. Control*, 37(11):1692–1708, 1992.

[24] A. Stefanescu, J. Esparza, and A. Muscholl. Synthesis of distributed algorithms using asynchronous automata. In *CONCUR*, number 2761 in LNCS, pages 27–41, 2003.

[25] T. Yoo and S. Lafortune. A general architecture for decentralized supervisory control of discrete-event systems. *Discrete Event Dynamic Systems: Theory and Applications*, 12(3):335–377, 2002.

[26] W. Zielonka. Notes on finite asynchronous automata. *R.A.I.R.O. — Informatique Théorique et Applications*, 21:99–135, 1987.