From Trees to Tree-Like: Distribution and Synthesis for Asynchronous Automata

Citation for the original published paper (version of record):

Lehaut, M., Muscholl, A., Piterman, N. (2026). From Trees to Tree-Like: Distribution and Synthesis for Asynchronous Automata. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 16503 LNCS: 396-417.
http://dx.doi.org/10.1007/978-3-032-22730-0_19

From Trees to Tree-Like: Distribution and Synthesis for Asynchronous Automata

Mathieu Lehaut[1] ⋆, Anca Muscholl[2] ⋆⋆†, and
Nir Piterman[3] ⋆⋆⋆†

[1] University of Warsaw, Poland
[2] LaBRI, University of Bordeaux, France
[3] University of Gothenburg and Chalmers University of Technology, Gothenburg, Sweden

Abstract. We revisit constructions for distribution and synthesis of Zielonka's asynchronous automata in restricted settings. We show first a simple, quadratic, distribution construction for asynchronous automata, where the process architecture is tree-like. An architecture is tree-like if there is an underlying spanning tree of the architecture and communications are local on the tree. This quadratic distribution result generalizes the known construction for tree architectures and improves on an older, exponential construction for triangulated dependence alphabets. Lastly we consider the problem of distributed controller synthesis and show that it is decidable for tree-like architectures. This extends the decidability boundary from tree architectures to tree-like keeping the same $\text{Tower}_d(n)$ complexity bound, where n is the size of the system and $d \geq 0$ the depth of the process tree.

Keywords: distributed synthesis, Zielonka automata, language distribution

1 Introduction

We consider a canonical formalism for distributed systems with a fixed communication structure – Zielonka's *asynchronous automata*. These are a well-known model that supports distributed synthesis under a fixed communication topology, rooted in the theory of Mazurkiewicz traces [21]. In Zielonka automata processes are connected to a specific set of channels, each process to their own

⋆ Supported by Swedish research council (VR) project (No. 2020-04963) and NCN grant 2021/41/B/ST6/00535.
⋆⋆ Supported by ANR-23-CE48-0005 grant PaVeDyS.
⋆⋆⋆ Supported by Swedish research council (VR) project (No. 2020-04963) and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.
† Part of this research was conducted while visiting the Simons Institute for the Theory of Computing.

N. Bertrand and S. Milius (Eds.): FoSSaCS 2026, LNCS 16503, pp. 396–417, 2026.
https://doi.org/10.1007/978-3-032-22730-0_19

subset of the channels. Processes communicate by synchronizing on channels they are connected to. During communication, all involved processes share their local states and then, based on this mutually shared information, move to new states. In addition, this model is rendez-vous: communication occurs only if all participants (processes connected to the channel) agree to it. The model is asynchronous as communications on two channels that do not have a process in common can be performed in parallel. Highlighting the importance of this model is Zielonka's seminal result about distribution of regular languages. Given a communication architecture – which process is connected to which channels – and a regular language that respects the asynchrony of the architecture, it is always possible to distribute this language into an asynchronous automaton [27].

Zielonka's result is a prominent, and rare, example of distributed synthesis, yet computationally expensive. Starting with a regular language accepted by an automaton with n states and a topology with p processes, an equivalent asynchronous automaton of size $O(4^{p^4} n^{p^2})$ can be constructed [11]. Note that the exponential is only due to the number of processes (which is part of the input), but this is indeed needed, as shown by a lower bound in [11]. The construction is quite involved and, even after nearly 40 years, it is not widely or easily understood. It has been the source for much research on how to understand, explain, and improve it (e.g. [22,13,11,2]). Recently, an alternative construction for asynchronous automata has been proposed using a partial-order variant of Propositional Dynamic Logic [1].

The complexity of the general Zielonka result prompted researchers to look at simpler cases where distribution could be easier. One notable example is the construction of Krishna and Muscholl [16] who show that when the communication architecture is restricted to a tree – every channel is listened to by at most two processes and there are no cycles – then every process needs a quadratic number of states in that of the original sequential automaton. Muscholl and Diekert [6] showed a simpler construction, however exponential in the number of states of the sequential automaton, for triangulated dependence alphabets.

The asynchronous automata model is notable also for its usage in control, similar to Ramadge and Wonham's supervisory control of discrete event systems [25,18]. For systems communicating synchronously by shared variables, it has been established early on that distributed control is undecidable [24], and only decidable for very restricted communication architectures, moreover with very high complexity [17,8]. In contrast, for distributed control based on asynchronous communication using Zielonka automata researchers have found more complex architectures for which the problem was still decidable. Notably, control for ω-regular winning conditions was shown to be decidable when communication is restricted to a tree [23] (see also [12] for local reachability conditions). Their construction has Tower(d) complexity in the depth d of the tree, but works for an architecture that is undecidable in the synchronous shared-variable world. Similarly, for *uniformly well-connected* architectures, the problem is again decidable but this time in exponential space [9]. On a general note, distributed control based on Zielonka automata, also known as asynchronous games with causal memory, is equivalent to so-called Petri games [3]. However, in spite of earlier hopes emanating from these general architectures having a decidable control problem, the general distributed synthesis has recently been established as undecidable [14].

Recently, Hausmann et al. studied distribution in the model of reconfigurable asynchronous automata [15]. Their model generalizes asynchronous automata by allowing processes to change the set of channels they communicate on at runtime. They established that the simple distribution construction of Krishna and Muscholl [16] is applicable also to a more general architecture they call *tree-like*. That is, processes are allowed to communicate more freely as long as their communication has an underlying spanning tree and communications are local on the tree. Here we present their construction in the context of asynchronous automata removing the complex notations related to the reconfiguration. While their construction was exponential in the number of processes in order to follow the structure of the tree, when the communication architecture is fixed this is no longer required and the quadratic construction of Krishna and Muscholl emerges. Furthermore, we show that this construction also improves a previously known exponential construction by Diekert and Muscholl in the context of triangulated dependence graphs [6], as they turn out to be equivalent to tree-like architectures. Finally, we revisit the distributed synthesis result for tree architectures [23] and show that this construction as well can be generalized to the case of tree-like architectures. Omitted proofs can be found in the long version of this paper [19].

For convenience, technical terms and notations in the electronic version of this manuscript are hyper-linked to their definitions (cf. `https://ctan.org/pkg/knowledge`).

2 Preliminaries

2.1 Deterministic Finite Automata

A deterministic finite automaton (DFA) over alphabet Σ is denoted as $\mathcal{A} = (\Sigma, S, \Delta, s_0, F)$, where S is the finite set of states, $\Delta : S \times \Sigma \to S$ the partial transition function, $s_0 \in S$ the initial state, and $F \subseteq S$ the set of accepting states. Given a word $w = a_0 \cdots a_{n-1}$, an initial run of A on w is a path $s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \dots \xrightarrow{a_{n-1}} s_n$ of $\mathcal{A}$, i.e. for every $0 \leq i < n$ we have $s_{i+1} = \Delta(s_i, a_i)$. An initial run is accepting if $s_n \in F$ and then w is accepted by $\mathcal{A}$. The language of $\mathcal{A}$, denoted by $\mathcal{L}(\mathcal{A})$, is the set of words accepted by $\mathcal{A}$. Often we abuse notation and write $\Delta(s, u)$ for the state s' reached from s on the word u (or just $s \xrightarrow{u} s'$).

An *independence relation* is a symmetric, irreflexive relation $I \subseteq \Sigma \times \Sigma$. Two words $u, v \in \Sigma^*$ are said to be I-indistinguishable, denoted by $u \sim_I v$, if one can start from u, repeatedly switch two consecutive independent letters, and end up with v. That is, $\sim_I$ is the transitive closure of the relation $\{(uabv, ubav) \mid (a, b) \in I,\ u, v \in \Sigma^*\}$. We denote by $[u]_I$ the I-equivalence class of a word $u \in \Sigma^*$. This is a.k.a. a Mazurkiewicz trace [21]. Let $\mathcal{A} = (\Sigma, S, \Delta, s_0, F)$ be a deterministic automaton over Σ. We say that $\mathcal{A}$ is I-*diamond* if for all pairs of independent letters $(a, b) \in I$ and all states $s \in S$, we have $\Delta(s, ab) = \Delta(s, ba)$. If $\mathcal{A}$ has this property, then a word u is accepted by $\mathcal{A}$ if and only if all words in $[u]_I$ are accepted.

2.2 Asynchronous Automata

A *communication architecture* $\mathbb{C} : \Sigma \to 2^{\mathbb{P}}$ associates with each letter the subset of processes reading it. We put $\mathbb{C}^{-1}(p) = \{a \in \Sigma \mid p \in \mathbb{C}(a)\}$. A *distributed alphabet* is $(\Sigma, \mathbb{C})$, where $\mathbb{C}$ is a communication architecture. It induces an independence relation $I(\mathbb{C}) \subseteq \Sigma \times \Sigma$ by $(a, b) \in I(\mathbb{C})$ iff $\mathbb{C}(a) \cap \mathbb{C}(b) = \emptyset$. The complement of the independence relation is a dependence relation $D(\mathbb{C}) = \Sigma \times \Sigma \setminus I(\mathbb{C})$. It is simple to see that $(a, b) \in D(\mathbb{C})$ iff $\mathbb{C}(a) \cap \mathbb{C}(b) \neq \emptyset$. A dependence relation D induces a graph $G_D = (\Sigma, D)$, where Σ is the set of nodes and D is the set of edges. We say that $\mathbb{C}$ is binary if for each $a \in \Sigma$ we have $|\mathbb{C}(a)| \leq 2$. A binary $\mathbb{C}$ induces a tree if the graph $(\mathbb{P}, E)$, where $(p, q) \in E$ iff $\mathbb{C}(a) = \{p, q\}$ with $p \neq q$ for some $a \in \Sigma$, is a tree. For any process $p \in \mathbb{P}$, let $\Sigma_p = \{a \in \Sigma \mid \{p\} = \mathbb{C}(a)\}$ be the set of p-*local actions*, that is the set of actions involving only p.

An *asynchronous automaton* (in short: *AA*) [27] over a distributed alphabet $(\Sigma, \mathbb{C})$ and processes $\mathbb{P}$ is a tuple $\mathcal{B} = ((S_p)_{p\in\mathbb{P}}, (s_p^0)_{p\in\mathbb{P}}, (\delta_a)_{a\in\Sigma}, Acc)$ such that:

- S_p is the finite set of states for process p, and $s_p^0 \in S_p$ is its initial state,
- $\delta_a : \prod_{p\in\mathbb{C}(a)} S_p \to \prod_{p\in\mathbb{C}(a)} S_p$ is a partial transition function for letter a that only depends on the states of processes in $\mathbb{C}(a)$ and changes them,
- $Acc \subseteq \prod_{p\in\mathbb{P}} S_p$ is a set of accepting states.

A global state of $\mathcal{B}$ is $\mathbf{s} = (s_p)_{p\in\mathbb{P}}$, giving the state of each process. For a global state $\mathbf{s}$ and a subset $P \subseteq \mathbb{P}$, we denote by $\mathbf{s}\downarrow_P = (s_p)_{p\in P}$ the part of $\mathbf{s}$ consisting of states from processes in P.

An initial run of $\mathcal{B}$ on a word $a_1a_2\ldots a_n$ is a path $\mathbf{s}_0 \xrightarrow{a_1} \mathbf{s}_1 \xrightarrow{a_2} \ldots \xrightarrow{a_n} \mathbf{s}_n$ of the product (global) automaton from the initial state $\mathbf{s}_0 = (s_p^0)_{p\in\mathbb{P}}$: for all $0 < i \leq n$, $\mathbf{s}_i \in \prod_{p\in\mathbb{P}} S_p$, $a_i \in \Sigma$, satisfying $\mathbf{s}_i \downarrow_{\mathbb{C}(a_i)} = \delta_{a_i}(\mathbf{s}_{i-1} \downarrow_{\mathbb{C}(a_i)})$ and $\mathbf{s}_i \downarrow_{\mathbb{P}\setminus\mathbb{C}(a_i)} = \mathbf{s}_{i-1} \downarrow_{\mathbb{P}\setminus\mathbb{C}(a_i)}$. An initial run is accepting if $\mathbf{s}_n$ belongs to Acc. The word $a_1a_2\ldots a_n$ is accepted by $\mathcal{B}$ if such an accepting run exists (note that automata are deterministic but runs on certain words may not exist). The language of $\mathcal{B}$, denoted by $\mathcal{L}(\mathcal{B})$, is the set of words accepted by $\mathcal{B}$. Let $\text{Runs}(\mathcal{B})$ denote the set of runs of $\mathcal{B}$, and $\text{Runs}_p(\mathcal{B})$ their projection on the p-component of the state.

We say that a language $\mathcal{L} \subseteq \Sigma^*$ over $(\Sigma, \mathbb{C})$ is *distributively recognized* if there exists an asynchronous automaton $\mathcal{B}$ such that $\mathcal{L}(\mathcal{B}) = \mathcal{L}$.

Theorem 1 ([27,11,16]). *Given an $I(\mathbb{C})$-diamond deterministic automaton $\mathcal{A}$, there exists an AA $\mathcal{B}$ that distributively recognizes the language of $\mathcal{A}$. In general, if $\mathcal{A}$ has n states then every process of $\mathcal{B}$ has $O(4^{|\mathbb{P}|^4} n^{|\mathbb{P}|^2})$ states. If $\mathbb{C}$ induces a tree, then every process of $\mathcal{B}$ has $O(n^2)$ states.*

The size of of an AA $\mathcal{B}$ (written as $|\mathcal{B}|$) is defined as $\max_{p\in\mathbb{P}} |S_p|$. This standard notion of size takes into account the maximal local memory (state space) of a process, because an AA is a *distributed* device.

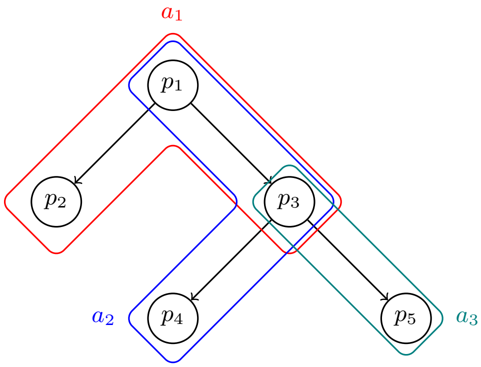

Fig. 1. A tree-like communication architecture: the tree is given by the black edges, while the communication architecture is drawn with one color group for each letter.

2.3 Tree-like communication architectures

As before, let $\mathbb{P}$ be a set of processes and Σ an alphabet, both finite. A communication architecture is tree-like if there is a tree that spans letters that synchronize more than two processes. Formally, we have the following.

Let $|\mathbb{P}| = n$. A *tree* over $\mathbb{P}$ is a $T = (\mathbb{P}, r, E)$ where $\mathbb{P}$ is the set of nodes, of which $r \in \mathbb{P}$ is the *root*, and $E \subseteq \mathbb{P}^2$ is the set of *edges* that is cycle free and connected.

Definition 1. *We say that $\mathfrak{A} = (\mathbb{C}, T)$, where $\mathbb{C}$ is a communication architecture and T is a tree, is a tree-like communication architecture (short: TCA) if*

1. *for all letters $a \in \Sigma$, the set $\mathbb{C}(a)$ is connected in T, i.e. if $p, q \in \mathbb{C}(a)$, then all processes along the path from p to q in T are also in $\mathbb{C}(a)$, and*
2. *if $(p, q) \in E$, then there is a letter $a \in \Sigma$ such that $p, q \in \mathbb{C}(a)$.*

Condition 1 ensures that a communication on a given letter is always "local" in the tree. Condition 2 ensures that the tree does not consist of disconnected parts. In case that condition 2 does not hold we say that $\mathfrak{A}$ is *forest-like*.

Example 1. Fix sets $\mathbb{P} = \{p_1, \ldots, p_5\}$ of processes and $\Sigma = \{a_1, a_2, a_3\}$ of letters. Then $\mathfrak{A} = ((\mathbb{C}(a_1), \mathbb{C}(a_2), \mathbb{C}(a_3)), T)$ as given in Figure 1 is a tree-like communication architecture rooted in p_1:

1. Every letter is connected in T.
2. All edges in T are covered by at least one letter.

Notions of parents, children, descendants, leaves, neighbors, and subtrees are defined as usual.

3 Asynchronous Automata with Tree-like Architectures

Hausmann et al. [15] showed that *reconfigurable asynchronous automata* can be distributed when they communicate over tree-like architectures. Reconfigurable asynchronous automata change their communication architectures during the operation. Thus, they are more general than asynchronous automata. Here, we present their construction for the case of a *fixed* tree-like architecture. Equivalently, we show that the construction of Krishna and Muscholl [16] for tree architectures extends, with more complicated transitions, to tree-like architectures.

We extend the definition of independence to sequences and to sets of letters. First we set $C_1, C_2 \subseteq \Sigma$ as independent (and write $(C_1, C_2) \in I(\mathbb{C})$) if for every $a_1 \in C_1$ and $a_2 \in C_2$ we have $(a_1, a_2) \in I(\mathbb{C})$. Equivalently, $C_1 \times C_2 \subseteq I(\mathbb{C})$. We set $w_1, w_2 \in \Sigma^*$ as independent (and write $(w_1, w_2) \in I(\mathbb{C})$) if there exist sets $C_1, C_2 \subseteq \Sigma$ such that $w_1 \in C_1^*$, $w_2 \in C_2^*$, and $(C_1, C_2) \in I(\mathbb{C})$.

Fix a DFA $\mathcal{A} = (\Sigma, S, \Delta, s_0, F)$ and $\mathfrak{A} = (\mathbb{C}, T)$ such that $\mathcal{A}$ is $I(\mathbb{C})$-diamond. If w_1 and w_2 are two $I(\mathbb{C})$-independent words, then we have that $\Delta(s, w_1 w_2) = \Delta(s, w_2 w_1) = \Delta(s, w)$ for any w that is an interleaving of w_1 and w_2. This can be verified by induction on the length of w.

We now study what information is needed in order to compute $\Delta(s, w_1 w_2)$ without memorizing w_1 and w_2 themselves.

Fix a state s of $\mathcal{A}$ and let w_1 and w_2 be two $I(\mathbb{C})$-independent words. Let $s_1 = \Delta(s, w_1)$, $s_2 = \Delta(s, w_2)$, and let C_1, C_2 be sets such that $w_1 \in C_1^*$, $w_2 \in C_2^*$ and $(C_1, C_2) \in I(\mathbb{C})$. The next lemma shows that the state $\Delta(s, w_1 w_2)$ can be computed knowing only s, s_1, s_2, and C_2 (but not the exact words w_1, w_2).

Lemma 1 ([5]). *Let $\mathcal{A}$ be an $I(\mathbb{C})$-diamond DFA. There exists a (partially-defined) function* Diam $: S^3 \times 2^{\Sigma} \to S$ *such that if $s, s_1, w_1, s_2, w_2, C_2$ are all defined as described above, then* $\mathrm{Diam}(s, s_1, s_2, C_2) = \Delta(s, w_1 w_2)$.

Proof. Let C_1' be the maximal set of letters such that $(C_1', C_2) \in I(\mathbb{C})$. It follows that $C_1 \subseteq C_1'$. Consider some arbitrary words w_1' and w_2' such that $w_1' \in C_1'^*$, $w_2' \in C_2^*$ and such that $s_i = \Delta(s, w_i) = \Delta(s, w_i')$ for $i \in \{1, 2\}$. Let us show that $\Delta(s, w_1' w_2') = \Delta(s, w_1 w_2)$.

First, by diamond property and hypothesis we have that $s' := \Delta(s, w_1 w_2) = \Delta(s, w_2 w_1) = \Delta(s_2, w_1)$. We also know that $\Delta(s, w_2') = s_2$, so $s' = \Delta(s, w_2' w_1)$. Then, reusing the diamond property: $s' = \Delta(s, w_2' w_1) = \Delta(s, w_1 w_2')$. We conclude reusing the hypothesis: $s' = \Delta(s, w_1 w_2') = \Delta(s, w_1' w_2')$.

Finally we can set $\mathrm{Diam}(s, s_1, s_2, C_2)$ as the state $\Delta(s, w_1' w_2')$, for some $w_1' \in C_1'^*, w_2' \in C_2^*$. □

In [16], the Diam function has been used for TCA $\mathfrak{A} = (\mathbb{C}, T)$ with binary $\mathbb{C}$ and $I(\mathbb{C})$-diamond DFA $\mathcal{A}$ in the following way. On a communication between a parent and one of its children, let s be the most recent state of $\mathcal{A}$ known by both the parent and the child, s_1 and s_2 be the most recent states known by the parent and the child respectively, and C_2 the set of letters with domains in the subtree rooted in the child. Then one can combine the information known by the parent and the child after the communication by computing $\mathrm{Diam}(s, s_1, s_2, C_2)$. We do not provide more details because we show below the more general case of non-binary $\mathbb{C}$.

In our tree-like setting, communications can include more than two participants. To that end, we naturally extend the Diam function to work on subtrees. Let T be a tree where each node is labeled by a pair (s,t) of states and a set $C^{\downarrow}$ of letters, with the intuition that s is the most recent state of $\mathcal{A}$ known by both the process at this node and its parent, t is the most recent state of $\mathcal{A}$ known by this process, and $C^{\downarrow}$ is the set of letters that can be found in the subtree rooted in that node and that are independent from those shared with the parent. Let r be the root of T, labeled by $(s_r, t_r, C_r^{\downarrow})$, and assume that r has k children labeled $(s_i, t_i, C_i^{\downarrow})$ and leading to tree T_i for $i \in \{1, \ldots, k\}$. Then we define the function TDiam recursively in the following way:

$$\mathrm{TDiam}(T) = \begin{cases} t_r & \text{if } k = 0, \\ \mathrm{Diam}(s_k, \mathrm{TDiam}(T \setminus T_k), \mathrm{TDiam}(T_k), C_k^{\downarrow}) & \text{otherwise.} \end{cases}$$

Note that the state(p) function from [16] is equivalent to $\mathrm{TDiam}(T_p)$ where T_p is the tree rooted in p.

Consider a letter $a \in \Sigma$, where $\mathbb{C}(a) = \{p_1, \ldots, p_k\}$. We define the *tree of a*, denoted by T_a, as the subtree of T whose set of nodes is exactly $\mathbb{C}(a)$; by definition of a TCA this is indeed a tree. Consider now a second letter $b \neq a$ and some p such that $p \in \mathbb{C}(a)$ but $p \notin \mathbb{C}(b)$. Again by definition of a TCA, $\mathbb{C}(b)$ occupies a connected area of T, and this area does not contain p. Therefore, it must either lie “above” p, meaning that the (unique) path from p to this area goes through the parent of p, or “below” p, meaning that it goes through one of p's children. We define $C_p^{\downarrow}$ to be the set of letters for which the second case holds. Now assume that each $p_i \in \mathbb{C}(a)$ is associated with a pair of states (s_i, t_i) of $\mathcal{A}$. Then we label each node p_i of T_a by $(s_i, t_i, C_{p_i}^{\downarrow})$ and define the state $s_a = \mathrm{TDiam}(T_a)$. Intuitively, that means we combine the information of all nodes in T_a to compute the most up-to-date state s_a.

We are now ready to state our construction. We distribute $\mathcal{A} = (\Sigma, S, \Delta, s_0, F)$ by defining $\mathcal{B} = ((S_p)_{p\in\mathbb{P}}, (s_p^0)_{p\in\mathbb{P}}, (\delta_a)_{a\in\Sigma}, Acc)$. Namely, for each $p \in \mathbb{P}$ we define the set S_p and its initial state s_p^0, the transition function δ_a for each $a \in \Sigma$ and the acceptance set Acc. For each $p \in \mathbb{P}$ we set $S_p = S \times S$ and $s_p^0 = (s_0, s_0)$.

For $a \in \Sigma$ we describe now the transition $\delta_a : \prod_{p\in\mathbb{C}(a)} S_p \to \prod_{p\in\mathbb{C}(a)} S_p$. The transition computes the diamond closure over the states held by all other processes in the correct order. Based on this computation each process updates its state.

Let $\mathbb{C}(a) = \{p_1, \ldots, p_k\}$ and for all $1 \le i \le k$ let (s_i, t_i) be the state of p_i. Let $s_a = \mathrm{TDiam}(T_a)$ as described earlier, and let $s' = \Delta(s_a, a)$. We set

$$\delta_a((s_1, t_1), \ldots, (s_k, t_k)) = ((u_1, v_1), \ldots, (u_k, v_k))$$

where the new state of process p_i is $u_i = \textsc{is-root}_a^{p_i}?s_i{:}s'$ and $v_i = s'$ with

$$\textsc{is-root}_a^{p}?s{:}t = \begin{cases} s & \text{if } p \text{ is the root of } T_a, \\ t & \text{otherwise.} \end{cases}$$

If $\Delta(s_a, a)$ is undefined, then so is δ_a.

We now define Acc. Consider a global state $\mathbf{s} = ((s_1, t_1), \ldots (s_n, t_n))$. Recall that the TCA is $\mathfrak{A} = (\mathbb{C}, T)$. For the tree T, consider the labeling of every node i in T by the pair (s_i, t_i) and the set $C_i^{\downarrow}$ of letters found "below" node i, as defined earlier. We then set $\mathbf{s} \in Acc$ if and only if $\mathrm{TDiam}(T) \in F$.

We state our main result below.

Theorem 2. *If $\mathcal{A}$ is $\mathbb{C}$-diamond and $\mathbb{C}$ is tree-like, then $\mathcal{B}$ recognizes distributively $\mathcal{L}(\mathcal{A})$. The size of $\mathcal{B}$ is in $O(n^2)$, where n is the number of states of $\mathcal{A}$.*

3.1 Proof of correctness

This section is dedicated to the proof of Theorem 2.

First, and completely independently from the construction, we need to define what is the view of a (set of) process(es), as in [16], and then show a useful property relating TDiam and those views. Let $\mathcal{A}$ be a $\mathbb{C}$-diamond DFA, $w \in \Sigma^*$ a word, $\rho = s_0 \xrightarrow{a_1} \ldots \xrightarrow{a_k} s_k$ the initial run of $\mathcal{A}$ on $w = a_1 \ldots a_k$, and $X \subseteq \mathbb{P}$ a set of processes. As processes in X are not necessarily part of every communication in w, there may be some parts of ρ that are happening outside of X's knowledge and processes in X may not know the state reached at the end of ρ just by themselves. We define recursively the *view* of X on w, which intuitively corresponds to the subword of w that processes in X can see through shared actions, and denote it by $\mathrm{view}_X(w)$:

$$\mathrm{view}_X(\varepsilon) = \varepsilon$$
$$\mathrm{view}_X(w \cdot a_k) = \begin{cases} \mathrm{view}_{X \cup \mathbb{C}(a_k)}(w) \cdot a_k & \text{if } X \cap \mathbb{C}(a_k) \neq \emptyset, \\ \mathrm{view}_X(w) & \text{otherwise.} \end{cases}$$

We compute what is also known as the causal past of processes in X: We start from the last communication in which at least one process in X participated. Then, the $\mathrm{view}_X(w)$ computes backwards the subsequence of w seen by processes in X through shared actions. Abusing notations, we identify any tree $T = (\mathbb{P}, r, E)$ with its set of processes $\mathbb{P}$ and write $\mathrm{view}_T(w)$ instead of $\mathrm{view}_{\mathbb{P}}(w)$. We also define the state of $\mathcal{A}$ that those processes can compute as $\sigma_X(w) = \Delta(s_0, \mathrm{view}_X(w))$. Furthermore, with $\mathcal{A}$ being $\mathbb{C}$-diamond, we can deduce that for any letter a, $\Delta(s_k, a)$ is defined if and only if $\Delta(s_0, \mathrm{view}_{\mathbb{C}(a)}(w)a)$ is defined. That is, whether a communication with letter a can happen cannot depend on communications made outside of the view of those listening to a.

Let $p \in \mathbb{P}$ be some process with parent q in T. The *shared parent view* of p on w, denoted by $\mathrm{view}_p^{\uparrow}(w)$, is defined as $\mathrm{view}_{\{p\}}(w')$ where w' is the minimal prefix of w containing all occurrences of all letters b such that $\{p, q\} \subseteq \mathbb{C}(b)$. In other words this is the sequence of actions, as seen from p's point of view, until the last communication involving both p and its parent. This is not necessarily the same as $\mathrm{view}_{\{p\}}(w)$, because p may have communicated with its children after the end of w'. For similar reasons, $\mathrm{view}_p^{\uparrow}(w)$ is not necessarily the same as $\mathrm{view}_{\{q\}}(w)$. If p is the root of T, we set $\mathrm{view}_p^{\uparrow}(w) = \varepsilon$ for all w. Finally, we set $\sigma_p^{\uparrow}(w) = \Delta(s_0, \mathrm{view}_p^{\uparrow}(w))$ to be the state reached in $\mathcal{A}$ after reading that sequence.

The Diam function, and by extension TDiam, interplay nicely with the views of independent parts of the system. More specifically, we show that if each node in T is correctly labeled, then TDiam(T) correctly computes the view according to every process in T.

Lemma 2. *For all words w and for all subtrees T' of T where each node p of T' is labeled by $(\sigma_p^{\uparrow}(w), \sigma_{\{p\}}(w), C_p^{\downarrow})$, we have* $\mathrm{TDiam}(T') = \sigma_{T'}(w)$.

Proof. By induction on the structure of T', same as in [16].

If T' is a single node p labeled by $(s_p, t_p, C_p^{\downarrow})$ then $\mathrm{TDiam}(T') = t_p = \sigma_{\{p\}}(w)$ by assumption. Otherwise, assume T' is made of a root p labeled by $(s_p, t_p, C_p^{\downarrow})$ with k children $p_1, \ldots, p_k$ labeled by $(s_1, t_1, C_1^{\downarrow}), \ldots, (s_k, t_k, C_k^{\downarrow})$ and leading to subtrees $T_1, \ldots, T_k$, respectively. Assume that the property holds on those subtrees, i.e. that for all $i \leq k$, we have $\mathrm{TDiam}(T_i) = \sigma_{T_i}(w)$. Then with T_i' denoting the subtree of T' composed of p, its first i children, and their respective descendants, we recursively show that

$$(P_i) \qquad\qquad \mathrm{TDiam}(T_i') = \sigma_{T_i'}(w)$$

For the initial step $i = 0$, $\mathrm{TDiam}(T_0') = t_p = \sigma_{\{p\}}(w)$ by assumption. Now suppose (P_{i-1}) holds for $1 \leq i \leq k$. By construction

$$\mathrm{TDiam}(T_i') = \mathrm{Diam}(s_i, \mathrm{TDiam}(T_{i-1}'), \mathrm{TDiam}(T_i), C_i^{\downarrow}).$$

By (P_{i-1}) we know that $\mathrm{TDiam}(T_{i-1}') = \sigma_{T_{i-1}'}(w)$, and by assumption we have that $s_i = \sigma_{p_i}^{\uparrow}(w)$ and $\mathrm{TDiam}(T_i) = \sigma_{T_i}(w)$.

Let $w_0 = \mathrm{view}_{p_i}^{\uparrow}(w)$, $w_i = \mathrm{view}_{T_i}(w)$, and $w_{i-1} = \mathrm{view}_{T_{i-1}'}(w)$. By definition, w_0 is a prefix of both w_i and w_{i-1}. Thus, let $w_i = w_0 \cdot w_i'$ and $w_{i-1} = w_0 \cdot w_{i-1}'$. As w_0 contains the last communication involving p_i and its parent p, it is easy to see that w_i' and w_{i-1}' are independent, otherwise w_0 would not be their longest common prefix. Note also that $\mathrm{view}_{T_i'}(w) = w_0 v$, where v is an interleaving of w_i' and w_{i-1}'. Also, all actions of w_i' are in $C_{p_i}^{\downarrow}$, otherwise they would involve p. Using the definition of TDiam, we get $\mathrm{TDiam}(T_i') = \mathrm{Diam}(s_i, \sigma_{T_{i-1}'}(w), \sigma_{T_i}(w), C_i^{\downarrow})$ with $s_i = \sigma_{p_i}^{\uparrow}(w) = \Delta(s_0, w_0)$, $\sigma_{T_{i-1}'}(w) = \Delta(s_i, w_{i-1}')$ and $\sigma_{T_i}(w) = \Delta(s_i, w_i')$. Thus, by Lemma 1, $\mathrm{TDiam}(T_i') = \sigma_{T_i'}(w)$.

We have shown that (P_i) holds for all i, and therefore for $i = k$, and as $T' = T_k'$ we have $\mathrm{TDiam}(T') = \sigma_{T'}(w)$. $\square$

Note that this lemma implies that if T' is the full tree T and is correctly labeled, then $\mathrm{TDiam}(T) = \sigma_{\mathbb{P}}(w) = \Delta(s_0, w)$. We will use this property to prove the correctness of our construction.

Let us define invariants that should be satisfied throughout a run. Let $w \in \Sigma^*$. Our first invariant states that a run ρ' of $\mathcal{B}$ on w exists if and only if a run ρ of $\mathcal{A}$ on the same word w exists.

$$(\mathtt{I}_{\mathrm{DEF}}[w]) : \text{The run } \rho \text{ on } w \text{ is defined} \Leftrightarrow \text{The run } \rho' \text{ on } w \text{ is defined.}$$

Now if both runs are undefined, then w is trivially rejected by both $\mathcal{A}$ and $\mathcal{B}$ and there is nothing left to show. So for the remaining invariants, assume that both ρ and ρ' are defined and end in states s and $\mathbf{s} = ((s_1, t_1), \dots, (s_n, t_n))$ respectively. Then we establish the invariants relating the actual state of ρ with the pair of states that each process keeps in its local state, which corresponds to the intuition given in the previous section.

$$(\mathtt{I}_S^{\uparrow}[w]) : \quad \forall p \in \mathbb{P}.\, s_p = \sigma_p^{\uparrow}(w) \qquad\qquad (\mathtt{I}_S[w]) : \quad \forall p \in \mathbb{P}.\, t_p = \sigma_{\{p\}}(w)$$

Those two invariants, used with Lemma 2, are key to prove that the languages of $\mathcal{A}$ and $\mathcal{B}$ are equivalent. Let us now prove that those invariants hold inductively.

Initialization of the invariants. All invariants hold on the empty word $w = \varepsilon$. $(\mathtt{I}_{\text{DEF}}[\varepsilon])$ is trivial. $(\mathtt{I}_S^{\uparrow}[\varepsilon]), (\mathtt{I}_S[\varepsilon])$ are easily derived from the definition of the initial state $\mathbf{s}_0$ of $\mathcal{B}$.

Invariants after a transition. Now consider a word of the form $w' = w \cdot a$, with both runs ρ and ρ' defined up to w ending in states s and $\mathbf{s}$ respectively. Assume that all invariants hold in w. We show that they still hold in w'.

For $(\mathtt{I}_{\text{DEF}}[w'])$, by using $(\mathtt{I}_S^{\uparrow}[w])$, $(\mathtt{I}_S[w])$, and Lemma 2, we have that $s' :=$ $\mathrm{TDiam}(T_a) = \sigma_{\mathbb{C}(a)}(w)$. Therefore, $\Delta(s, a)$ is defined if and only if $\Delta(s', a)$ is defined. By definition of $\mathcal{B}$, there is an a transition from $\mathbf{s}$ iff $\Delta(s', a)$ is defined iff $\Delta(s, a)$ is defined iff there is an a transition in $\mathcal{A}$. Thus $(\mathtt{I}_{\text{DEF}}[w'])$ holds.

Now assume both runs are defined for w' and let $s \xrightarrow{a} s'$ in $\mathcal{A}$ and $\mathbf{s} \xrightarrow{a} \mathbf{s}'$ in $\mathcal{B}$. Let $p \in \mathbb{P}$, its state in $\mathbf{s}$ be (s_p, t_p), and its state in $\mathbf{s}'$ be (s'_p, t'_p). If p was not part of the communication on a, then its state is unchanged and by definition $\mathrm{view}_p^{\uparrow}(w') = \mathrm{view}_p^{\uparrow}(w)$ and $\mathrm{view}_{\{p\}}(w') = \mathrm{view}_{\{p\}}(w)$, so $(\mathtt{I}_S^{\uparrow}[w'])$ and $(\mathtt{I}_S[w'])$ are trivially obtained from $(\mathtt{I}_S^{\uparrow}[w])$ and $(\mathtt{I}_S[w])$ respectively.

Assume now that p was part of this communication. Let us first show $(\mathtt{I}_S[w'])$. By definition, $\mathrm{view}_{\{p\}}(w') = \mathrm{view}_{\mathbb{C}(a)}(w) \cdot a$. Again, by combining $(\mathtt{I}_S[w])$, $(\mathtt{I}_S^{\uparrow}[w])$, and Lemma 2, we get that $\sigma_{\mathbb{C}(a)}(w) = \mathrm{TDiam}(T_a) = s'$. Thus $\sigma_{\{p\}}(w') =$ $\Delta(s', a) = t'_p$ and $(\mathtt{I}_S[w'])$ holds. For $(\mathtt{I}_S^{\uparrow}[w'])$, there are two cases. First, assume that the parent of p in T is not part of the communication on a (either because p is the root and has no parent, or because its parent does not listen to a). In that case, $\mathrm{view}_p^{\uparrow}(w') = \mathrm{view}_p^{\uparrow}(w)$ by definition and $\sigma_p^{\uparrow}(w) = s_p$ by $(\mathtt{I}_S^{\uparrow}[w])$. Moreover, p must be the root of T_a. Therefore $s'_p = \textsc{is-root}_a^p ? s_p{:}\Delta(s_a, a) = s_p$ and $(\mathtt{I}_S^{\uparrow}[w'])$ is satisfied. Second, assume now that p's parent, say q, is part of the communication on a. Then by definition $\mathrm{view}_p^{\uparrow}(w') = \mathrm{view}_{\{p\}}(w') =$ $\mathrm{view}_{\mathbb{C}(a)}(w) \cdot a$. Thus, $\sigma_p^{\uparrow}(w') = \Delta(s', a)$ by the same proof as for $(\mathtt{I}_S[w'])$. Thus $s'_p = \textsc{is-root}_a^p ? s_p{:}\Delta(s', a) = \Delta(s', a)$ and we have $(\mathtt{I}_S^{\uparrow}[w'])$.

Conclusion of the proof. We have shown that the invariants hold for any word w. We use them to show that $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$. Let $w \in \Sigma^*$ be a word, then w is accepted by $\mathcal{A}$ iff there is a run of $\mathcal{A}$ on w ending in a state $s \in F$. By $(\mathtt{I}_{\text{DEF}}[w])$, the run of $\mathcal{A}$ on w is defined iff the run of $\mathcal{B}$ on w is defined. If both runs are undefined, then w is neither in $\mathcal{L}(\mathcal{A})$ nor $\mathcal{L}(\mathcal{B})$. Otherwise, by $(\mathtt{I}_S^{\uparrow}[w])$, $(\mathtt{I}_S[w])$, and Lemma 2, we have that $\mathrm{TDiam}(T) = \mathrm{view}_{\mathbb{P}}(w) = s$. Then $\mathcal{B}$ accepts w iff $\mathrm{TDiam}(T) = s \in F$ iff $\mathcal{A}$ accepts w. Therefore $\mathcal{L}(\mathcal{A}) = \mathcal{L}(\mathcal{B})$, which concludes the proof of Theorem 2.

3.2 Triangulated dependence alphabets

The complexity of Zielonka's general distribution construction stems from the need to store information regarding knowledge about other processes and a time-stamping mechanism that allows to put together these pieces of knowledge [27]. Diekert and Muscholl have shown that in the case that the dependence relation between the letters in Σ is a *triangulated graph*, this complex structure of information can be avoided [6,7].

We observe that tree-like architectures give a fresh view on Diekert and Muscholl's construction, and an AA construction that is exponentially better.

Consider a graph $G = (V, E)$. A sequence of vertices $v_1, \ldots, v_n$ is a cycle if for every i we have $(v_i, v_{i+1}) \in E$ and $(v_n, v_1) \in E$. A graph $G = (V, E)$ is *triangulated* if for every cycle $v_1, \ldots, v_n$ such that $n > 3$ there exists a pair $(v_i, v_j) \in E$ such that $1 < |j - i| < n - 1$. That is, the cycle must have some chord.

Consider an architecture $\mathbb{C}$ and let $D = D(\mathbb{C})$. It turns out that an architecture is tree-like if and only if the dependency graph $G = (\Sigma, D)$ is triangulated and connected. It follows that if a dependency graph is triangulated but not connected then its architecture is forest-like.

Lemma 3 ([10]). *An architecture $\mathbb{C}$ is tree-like iff $G_D = (\Sigma, D)$ is triangulated and connected.*

In Gavril's terms, given a tree, every graph of subtrees of the tree, where edges correspond to non-empty intersection of the subtrees, is triangulated. In the other direction, every triangulated graph can be represented as the graph of intersections of subtrees of an appropriate tree. The part about connectivity is simple to add.

Note first that the construction of AA for non-connected dependency graphs easily reduces to the construction for connected dependency graphs. To see this assume that Σ is the disjoint union of two pairwise independent alphabets Σ_1, Σ_2, so $(a, b) \notin D$ for every $a \in \Sigma_1, b \in \Sigma_2$. Consider some I-diamond DFA $\mathcal{A}$ over Σ and an input word $w \in \Sigma^*$. We can use the diamond property (Lemma 1) to infer from the states reached by $\mathcal{A}$ on the projection w_1 respectively w_2 of the input w on Σ_1 and Σ_2, resp., the state reached by $\mathcal{A}$ on $w \sim_I w_1 w_2$. More formally, assume that we constructed an AA $\mathcal{B}_1$ over alphabet Σ_1 and an AA $\mathcal{B}_2$ over alphabet Σ_2. For each of $\mathcal{B}_1$, $\mathcal{B}_2$ we can suppose that the global state reached on the respective input determines the state of $\mathcal{A}$ reached on that word. We obtain an AA $\mathcal{B}$ by composing $\mathcal{B}_1$, $\mathcal{B}_2$ in parallel (because they share no letter). The global state reached by $\mathcal{B}$ on $w \sim_I w_1 w_2$ determines the two states $s_1 = \Delta(s_0, w_1), s_2 = \Delta(s_0, w_2)$ of $\mathcal{A}$, and $\mathrm{Diam}(s_0, s_1, s_2, \Sigma_2)$ says which global states are accepting.

The result we improve on in Theorem 2 is stated in the next theorem. The exponential size comes from the fact that the construction relies on transition monoids.

Theorem 3 ([5,6]). *Given an I-diamond DFA $\mathcal{A}$ over a triangulated dependency graph $G = (\Sigma, D)$, an AA $\mathcal{S}$ that recognizes the same language can be constructed with $|\mathcal{S}| = n!$, where $n = |\mathcal{A}|$.*

4 Distributed Synthesis

We extend now the result of decidability of the control problem for asynchronous automata from binary-tree architectures [23] to tree-like architectures. We first recall the definitions and adapt them to our notations.

4.1 Controllers and Control Problem

Let us fix some distributed alphabet $(\Sigma, \mathbb{C})$. As in the setting of [25], Σ is partitioned into System (*controllable*) actions Σ_{sys} and Environment (*uncontrollable*) actions Σ_{env}. Remember that Σ_p denotes the set of p-local actions, then let $\Sigma_p^{sys} = \Sigma_p \cap \Sigma_{sys}$ and similarly for Σ_p^{env}. As in [23] we require that all communication actions, i.e., actions shared by at least 2 processes, are uncontrollable, in other words all controllable actions are local: $\Sigma_{sys} \subseteq \cup_{p \in \mathbb{P}} \Sigma_p$. This is not a restriction, as the general setting where communication actions can be controllable reduces to this one, as shown in Proposition 6 of [23]. Furthermore, there should be at least one controllable (thus local) action from each state. This is of course not a restriction because one can always add local, controllable self-loops when needed. This assumption will make the construction a bit simpler.

For control it is common to use local acceptance conditions, instead of global ones, because more systems are controllable with local conditions. That is, we replace the Acc part of the definition of AA by a set $\{Acc_p\}_{p \in \mathbb{P}}$. Each Acc_p is of the form (F_p, Ω_p) with $F_p \subseteq S_p$ and $\Omega_p : S_p \to \mathbb{N}$ a local parity function. The idea is that a finite run is accepted by p if it ends in a state in F_p, and an infinite run is accepted by p if it satisfies the (max) parity condition for p. Then a run is accepted by $\mathcal{A}$ if it is accepted by all processes p.

We say that a run over w is *maximal* if there is no way to decompose w into $w = u \cdot v$ and no $a \in \Sigma$ with $\mathbb{C}(a) \cap \mathbb{C}(v) = \emptyset$ such that the run over $u \cdot a \cdot v$ is defined. In plain words, it means we cannot extend the run on w by some action involving processes that perform only a finite number of actions in w.

Now we define the notion of controllers. For the rest of this section, let us fix some AA $\mathcal{A} = ((S_p)_{p \in \mathbb{P}}, (s_p^0)_{p \in \mathbb{P}}, (\delta_a)_{a \in \Sigma}, (Acc_p)_{p \in \mathbb{P}})$. For simplicity we will not use the general definition of controllers but go directly to what is actually called covering controllers for $\mathcal{A}$ in [23], and simply call them controllers. It is shown in Lemma 10 of [23] that the Control Problem for covering controllers is equivalent to the one for general controllers, and so we focus only on the former.

The intuition for a controller is that it is a machine (described by an AA without acceptance condition) guiding the System in choosing which controllable action(s) to follow in order to get only runs accepted by $\mathcal{A}$. On the other hand, as Environment actions are uncontrollable, the controller should not be able to restrict them from happening. To do that, a controller can enrich states of $\mathcal{A}$ by adding extra information and use this to choose a subset of controllable actions that are possible from this state, and enable only this subset of actions.

Formally, a *controller* for $\mathcal{A}$ is itself an AA $\mathcal{C} = ((C_p)_{p\in\mathbb{P}}, (c_p^0)_{p\in\mathbb{P}}, (\Delta_a)_{a\in\Sigma})$ together with a projection π that maps each state $c_p \in C_p$ of $\mathcal{C}$ to a state $s_p \in S_p$ of $\mathcal{A}$ and that satisfies the following requirements:

- for all $a \in \Sigma$ with $\mathbb{C}(a) = \{p_1, \dots, p_k\}$, and $\mathbf{c}, \mathbf{c}' \in C_{p_1} \times \cdots \times C_{p_k}$, we have that $\Delta_a(\mathbf{c}) = \mathbf{c}'$ implies $\delta_a(\pi(\mathbf{c}))) = \pi(\mathbf{c}')$ (with π applied component-wise),
- for all $a \in \Sigma_{env}$, if $\delta_a(\pi(\mathbf{c}))$ is defined then so is $\Delta_a(\mathbf{c})$,
- for all $p \in \mathbb{P}$, $\pi(c_p^0) = s_p^0$.

Thanks to the projection π, a controller $\mathcal{C}$ inherits the acceptance condition of $\mathcal{A}$. A controller $\mathcal{C}$ is *winning* for $\mathcal{A}$ if every maximal run of $\mathcal{C}$ is accepting.

The *Control Problem* asks, for a given input $\mathcal{A}$, whether there exists a winning controller $\mathcal{C}$ for $\mathcal{A}$ (and if so to build it explicitly).

Example 2. Consider a server-client setup where one process, the server, communicates with several processes, the clients. The server broadcasts to all clients that it wants a task to be solved. Each client starts working independently on the task requested, and communicates back to the server when it has finished. When the server gets two answers back from its children, it then sends another broadcast telling all clients to stop working.

Formally, we take $\mathbb{P} = \{p, c_1, \dots, c_n\}$ with p the server and $c_1, \dots, c_n$ the clients. We have two actions t_1, t_2 representing requests for two different *tasks*, $\mathbb{C}(t_1) = \mathbb{C}(t_2) = \mathbb{P}$. Each process c_i has two local actions p_1^i, p_2^i to *progress* those tasks, $\mathbb{C}(p_1^i) = \mathbb{C}(p_2^i) = \{c_i\}$. Then there is a shared action e_i to communicate back to the server that the task has *ended*, $\mathbb{C}(e_i) = \{c_i, p\}$. Finally, there is an action r that can broadcast to all clients to *reset* their state when the task has been successfully completed, $\mathbb{C}(r) = \mathbb{P}$. The architecture is tree-like with the server p at its root, and all clients c_i are the children of p. The automata of the server and the clients are illustrated in Figure 2. For the acceptance condition, we require that the server visits its initial state infinitely often, and that each client performs action e_i infinitely often. For simplicity of presentation, finite runs are not accepted (clients do not crash).

Now for the Control Problem, we say that all local actions (i.e. actions p_k^i) are controllable, while all non-local ones are uncontrollable by definition. The naive controller that only enables the correct progressing action on every process, e.g. p_1^i after receiving a t_1 broadcast, is not winning. Indeed, since Environment controls which processes get to perform their e_i, it could choose the same two processes for every task. Then those not chosen never get their turn at performing e_i and thus will not be accepting. However we can still build a winning controller in a round-robin manner. To do this, we simply add to the state of each c_i a count of how many tasks have been requested since the beginning modulo n, so we have $C_{c_i} = S_{c_i} \times \{0, \dots, n-1\}$ and the projection π is simply the projection on the first component. If the count is $k-1$, then the controllers for processes c_k and c_{k+1} are set to enable only the correct progress action, while other processes enable nothing. After that, Environment has no choice but to perform the corresponding e_k and e_{k+1} to keep the run going. Then the server sends a reset broadcast, the next task is requested, k is incremented, and each process gets to perform its e_i action in turn, thus this controller is winning.

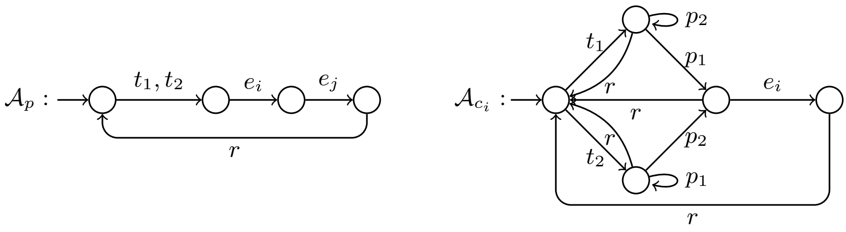

Fig. 2. A server-client protocol for distributing and executing tasks.

4.2 Leaf Process Simulated by Parent

In [23], it is shown that in an architecture that forms a tree where all channels are binary, a leaf process can be simulated by its parent for the purpose of control w.r.t. local parity conditions. This is done by abstracting away all local actions of the leaf into a finite summary, and any synchronization between the leaf and the parent can then be simulated locally by the parent. This simulation allows to deduce an upper complexity bound for the Control Problem that is a tower of exponentials in the depth of the tree.

We show here that the proof of [23] extends to tree-like architectures. The key observation is that, in TCA, by the continuity condition, any communication involving a leaf and other processes necessarily includes the parent of the leaf as well. Therefore, if some process relies on the state of the to-be-removed leaf for some transition, it will still be able to access that information after the removal because it will instead be provided by the parent that is simulating the leaf.

For the rest of this subsection we consider an AA $\mathcal{A}$ and a TCA where $\mathbb{P}$ is the set of processes, ℓ is a leaf process, and p is the parent of ℓ. The first step is to ensure that processes communicate "frequently". This will provide a finite description of what happens locally on leaf ℓ between two consecutive actions shared by ℓ with other processes.

Definition 2. *An AA $\mathcal{A}$ is ℓ-short if there is some bound $B \in \mathbb{N}$ such that in every local ℓ-run of $\mathcal{A}$, the number of local actions that ℓ can do in a row is at most B.*

Lemma 4 (Th.10, [23]). *For every AA $\mathcal{A}$ we can construct an ℓ-short automaton $\mathcal{A}^{\circledS}$ such that $\mathcal{A}$ is controllable iff $\mathcal{A}^{\circledS}$ is controllable. The state space of any $q \neq \ell$ does not change from $\mathcal{A}$ to $\mathcal{A}^{\circledS}$; the states of ℓ in $\mathcal{A}_\ell^{\circledS}$ are simple, local paths of $\mathcal{A}_\ell$.*

Although the proof of Lemma 4 is quite involved, it does not rely on the communication architecture being a tree. The main idea behind is that every parity game is equivalent to a finite cycle game.

It follows that to show that a leaf process can be simulated by its parent (for the purposes of control) it is enough to work with $\mathcal{A}^{\circledS}$, so in particular, with an ℓ-short AA. From Lemma 4 we know that the bound B can be set to the number of ℓ-states in $\mathcal{A}$, so $B = |S_\ell|$ with S_ℓ the set of states of ℓ in $\mathcal{A}$.

We are now ready to show that given $\mathcal{A}$ we can construct an AA $\mathcal{A}^{\backslash\ell}$ whose set of processes is $\mathbb{P} \setminus \{\ell\}$ and such that $\mathcal{A}$ is controllable iff $\mathcal{A}^{\backslash\ell}$ is controllable. Moreover, for every $p' \neq p$ we have $\mathcal{A}_{p'}^{\backslash\ell} = \mathcal{A}_{p'}$ and the size of $\mathcal{A}_p^{\backslash\ell}$ is proportional to $|\mathcal{A}_p| \cdot |\Sigma_p^{sys}| \cdot exp(|\mathcal{A}_\ell|)$.

Let $\mathcal{A} = ((S_p)_{p\in\mathbb{P}}, (s_p^0)_{p\in\mathbb{P}}, (\delta_a)_{a\in\Sigma}, (Acc_p)_{p\in\mathbb{P}})$, $\ell \in \mathbb{P}$ a leaf and $p \in \mathbb{P}$ its parent, and assume that $\mathcal{A}$ is ℓ-short thanks to Lemma 4. With $\mathbb{P}^{\backslash\ell} = \mathbb{P} \setminus \{\ell\}$, we build $\mathcal{A}^{\backslash\ell} = ((S_p^{\backslash\ell})_{p\in\mathbb{P}^{\backslash\ell}}, (s_p^{0\backslash\ell})_{p\in\mathbb{P}^{\backslash\ell}}, (\delta_a^{\backslash\ell})_{a\in\Sigma^{\backslash\ell}}, (Acc_p^{\backslash\ell})_{p\in\mathbb{P}^{\backslash\ell}})$. The new alphabet $\Sigma^{\backslash\ell}$ will be defined shortly after. For all processes $q \neq p$, we have $S_q^{\backslash\ell} = S_q$, $s_q^{0\backslash\ell} = s_q^0$, and $Acc_q^{\backslash\ell} = Acc_q$. For all actions $a \in \Sigma$ such that $\mathbb{C}(a) \cap \{\ell, p\} = \emptyset$, we have $\delta_a^{\backslash\ell} = \delta_a$. That is, all components that are not related to either p or ℓ remain unchanged.

Let us define $S_p^{\backslash\ell}$. To that end, we define ℓ-local strategies first. Remember that $\mathcal{A}$ is ℓ-short, so let B be the corresponding bound. Recall also that Σ_ℓ is the set of ℓ-local actions. For presentation purposes, we write $s \xrightarrow{a_1\ldots a_n} s'$ as a stand in for $s' = \delta_{a_n}(\ldots(\delta_{a_1}(s)))$ when $a_1, \ldots, a_n \in \Sigma_\ell$ and s, s' are states of ℓ. We write $s \xrightarrow{a_1\cdots a_n}$ when there exists a state s' such that $s \xrightarrow{a_1\ldots a_n} s'$. An *$\ell$-local strategy* from a given state $s_\ell \in S_\ell$ is a partial function $f : (\Sigma_\ell)^{\leq B} \to \Sigma_\ell^{sys}$ such that whenever $f(w) = a$ then $s_\ell \xrightarrow{w\cdot a}$ is defined in $\mathcal{A}_\ell$. If $s_\ell \xrightarrow{a} s'_\ell$ and f is an ℓ-local strategy from s_ℓ, then $f_{|a}$ is the ℓ-local strategy from s'_ℓ defined as $f_{|a}(w) = f(a \cdot w)$. With this defined, a state in $S_p^{\backslash\ell}$ of the parent p of ℓ is of one of 3 types:

$$(s_p, s_\ell), (s_p, s_\ell, f), (s_p, a, s_\ell, f)$$

where $s_p \in S_p$, $s_\ell \in S_\ell$ are the simulated states of p and ℓ respectively, f is an ℓ-local strategy from s_ℓ, and $a \in \Sigma_p^{sys}$ is a controllable local action of the parent p. Since $\mathcal{A}$ is ℓ-short, ℓ-local strategies can be seen simply as paths of $\mathcal{A}_\ell$ without repetitions, and thus there are only $exp(|\mathcal{A}_\ell|)$ many of them. Finally, we let $s_p^{0\backslash\ell} = (s_p^0, s_\ell^0)$ be the initial state of p.

We now precisely define the new alphabet $\Sigma^{\backslash\ell}$ before defining the new transitions. All actions that were previously ℓ-local are changed into uncontrollable p-local actions, regardless of whether they were controllable or not in Σ. Similarly, all actions that were p-local are now uncontrollable in $\Sigma^{\backslash\ell}$, but for each

p-local action $a \in \Sigma_p^{sys}$ that was controllable we add a new action $ch(a)$ in $\Sigma^{\setminus \ell}$ that is also p-local and controllable. We also add new actions $ch(f)$ that are p-local and controllable for each ℓ-local strategy f. Any non-local action that included ℓ in Σ remains in $\Sigma^{\setminus \ell}$ but with ℓ removed from its domain. All remaining actions in Σ, i.e. those that are local to some process not in $\{p, \ell\}$ or those that are non-local and do not intersect with ℓ, remain unchanged.

Let us now define transitions over actions involving p. First, from a state of p of the form (s_p, s_ℓ), there is only one kind of transition available

$$(s_p, s_\ell) \xrightarrow{ch(f)} (s_p, s_\ell, f) \text{ for all } \ell\text{-local strategies } f \text{ from } s_\ell$$

where System fixes a local strategy for ℓ. Then from these states there is again only one kind of transition possible

$$(s_p, s_\ell, f) \xrightarrow{ch(a)} (s_p, a, s_\ell, f) \text{ for all } a \in \Sigma_p^{sys} \text{ enabled from } s_p$$

where System chooses one p-local controllable (in Σ_p) action[4], provided there is a transition from s_p with this action. Then there are multiple choices from here. The two actions

$$(s_p, a, s_\ell, f) \xrightarrow{b_p} (s'_p, s_\ell, f) \text{ if } (s_p, s'_p) \in \delta_{b_p} \text{ and } b_p = a \text{ or } b_p \in \Sigma_p^{env}$$
$$(s_p, a, s_\ell, f) \xrightarrow{b_\ell} (s_p, a, s'_\ell, f_{|b_\ell}) \text{ if } (s_\ell, s'_\ell) \in \delta_{b_\ell} \text{ and } b_\ell \text{ is enabled by } f$$

are p-local actions (all uncontrollable) which either progress p by its pre-selected (previously) controllable action a, progress p by any local uncontrollable action b_p, or progress ℓ by any action enabled by the pre-selected strategy f, i.e. any uncontrollable ℓ-local action or the one controllable action output by f on ε. Then there are also non-local actions where p communicates with other processes.

$$\begin{aligned}
&((s_p, a, s_\ell, f), s_{q_1}, \ldots, s_{q_k}) \xrightarrow{b_{p+}} ((s'_p, s_\ell, f), s'_{q_1}, \ldots, s'_{q_k}) \text{ if} \\
&\quad p \in \mathbb{C}(b_{p+}), \ell \notin \mathbb{C}(b_{p+}), ((s_p, s_{q_1}, \ldots, s_{q_k}), (s'_p, s'_{q_1}, \ldots, s'_{q_k})) \in \delta_{b_{p+}} \\
&((s_p, a, s_\ell, f), s_{q_1}, \ldots, s_{q_k}) \xrightarrow{b_{p\ell+}} ((s'_p, s'_\ell), s'_{q_1}, \ldots, s'_{q_k}) \text{ if} \\
&\quad \ell, p \in \mathbb{C}(b_{p\ell+}), ((s_p, s_\ell, s_{q_1}, \ldots, s_{q_k}), (s'_p, s'_\ell, s'_{q_1}, \ldots, s'_{q_k})) \in \delta_{b_{p\ell+}}
\end{aligned}$$

The first kind is for when ℓ is excluded from the communication, so while s_p gets updated to a new state and the choice of a is reset, s_ℓ and f remain unchanged. The second kind occurs when ℓ is part of the communication, so both s_p and s_ℓ are read and updated, and the choices of a and f are reset. Note that $k = 0$ covers the case where a communication involving only p and ℓ occurs. Also, since we assumed that the architecture is tree-like, it is not possible for a communication to involve ℓ and other processes without p being included, which is why this case is not covered. Note that this is the only major difference with the original

[4] The assumption that in each state there is at least one enabled controllable action is used here.

construction from [23], and it is the reason that the proofs of correctness are mostly similar.

The last part of $\mathcal{A}^{\setminus\ell}$ is the acceptance conditions $Acc_p^{\setminus\ell} = (F_p^{\setminus\ell}, \Omega_p^{\setminus\ell})$. For $F_p^{\setminus\ell}$ it is simple: any state of the form $(s_p, s_\ell) \in F_p \times F_\ell$, and any $(s_p, s_\ell, f), (s_p, a, s_\ell, f)$ where $s_p \in F_p$ and any possible maximal continuation respecting f from s_ℓ eventually reaches a state in F_ℓ, are in this set. Defining $\Omega_p^{\setminus\ell}$ is trickier. First, an infinite run of p must satisfy the parity condition of Ω_p when restricted to the s_p component of the state. Then there are two cases. If the projection on the s_ℓ component is infinite and satisfies Ω_ℓ then the run must be accepted, which amounts to a conjunction of two parity conditions. This can be translated into a single parity condition, and we will discuss the complexity in Section 4.3. Otherwise, we require that for the last f and s_ℓ reached (uniquely defined as the only components seen infinitely often in the run), any possible continuation respecting f from s_ℓ eventually reaches a state in F_ℓ. This amounts to a single parity condition, the one on p.

Note that we do not define what happens when the run on ℓ is infinite but the run on p is finite, because this is not compatible with the ℓ-short assumption.

Now that $\mathcal{A}^{\setminus\ell}$ is defined, we are ready to prove the following theorem (proof in appendix).

Theorem 4. *For every ℓ-short AA $\mathcal{A}$ with local acceptance condition: there is a winning controller for $\mathcal{A}$ iff there is a winning controller for $\mathcal{A}^{\setminus\ell}$.*

4.3 Solving the Control Problem on Tree-like Architectures

Theorem 4 generalizes the leaf-elimination procedure of [23] from binary channels to tree-like architectures. Iterating this step reduces the distributed control problem to one involving a single process, so to a usual parity game. It is argued in [23] that the size of the resulting parity game is $\mathrm{Tower}_d(n)$, where n is the size of the AA and $d \geq 0$ the depth of the process tree. A matching lower bound for the control problem is provided in [12]. For the upper bound we provide below a detailed argument based on the following lemma:

Lemma 5. *Given $n+1$ parity conditions p_0 and $p_1, \ldots, p_n$, there is a deterministic parity automaton with $O\left((p_0+p)\cdot \frac{p!\cdot(p_0+1)^p}{(\prod_{i>0}(p_i!))}\right)$ states and p_0+p priorities, where $p = \sum_{i>1} p_i$, reading tuples of $n+1$ priorities and accepting iff all $n+1$ parity conditions are accepting.*

Proof. We use a variant of the index appearance record construction [26,20] to convert $n+1$ parity conditions to one condition. We start by using the index appearance record to convert the conditions $p_1, \ldots, p_n$ to one parity condition with p priorities. This is a straightforward application of the index appearance record, noticing that the order within each parity condition is always maintained. Thus, we keep a permutation of p indices, but where all the indices belonging to a single parity condition are always ordered in decreasing order. Thus, there are $\frac{p!}{(\prod_{i>0}(p_i!))}$ such permutations. It remains to keep the respective order of the elements in p_0 among the sequence of p indices. As the elements of p_0 themselves are ordered in decreasing order, it is sufficient to keep track of how many of them appear before every element in the sequence of p indices. Thus, we multiply the number of options by $(p_0+1)^p$. Thus, $\frac{p!\cdot(p_0+1)^p}{(\prod_{i>0}(p_i!))}$ possible values represent all possible sequences of p_0+p indices, where the indices of each parity condition itself are ordered in decreasing order. Every transition of the created parity automaton needs to signal a priority between p_0+p and 1. For that, we keep a record of what was the left-most index that is visited in a transition from one p_0+p sequence to its successor. This brings the total number of states to the stated in the Lemma. We note that p_0 does not appear in the exponent. $\square$

We note that the complexity of solving games where the winning condition is a conjunction of parity conditions is known to be co-NP-complete [4]. Thus, a much simpler conversion to parity games than the one described in the previous lemma is not possible.

We also note that the upper bound of the previous construction [23] was assuming that the acceptance condition is not part of the input.

Corollary 1. *The Control Problem for asynchronous automata over tree-like architectures is decidable in d-EXPTIME where d is the depth of the tree.*

Proof. We apply Theorem 4 iteratively. Choose a parent p with leaves $\ell_1,\ldots,\ell_n$. Apply the theorem to each one of the leaves $\ell_1,\ldots,\ell_n$. We end up with an equi-realizable control problem, where $\ell_1,\ldots,\ell_n$ are missing and p is replaced by $p^{\setminus \ell_1,\ldots,\ell_n}$. The number of states of $p^{\setminus \ell_1,\ldots,\ell_n}$ is proportional to $|S_p|\cdot|\Sigma|^n\cdot\prod_{i>0} exp(|S_{\ell_i}|)$. Furthermore, the infinitary condition of $p^{\setminus \ell_1,\ldots,\ell_n}$ is a conjunction of $n+1$ parity conditions. By Lemma 5, if the number of parities of p and $\ell_1,\ldots,\ell_n$ are $p_0,\ldots,p_n$, respectively, then by multiplying the number of states of $p^{\setminus \ell_1,\ldots,\ell_n}$ by $O\left((p_0+p)\cdot\frac{p!\cdot(p_0+1)^p}{(\prod_{i>0}(p_i!))}\right)$ we get a parity condition with $\sum_{i\geq 0} p_i$ priorities.

It follows that by eliminating leaves level by level, we get an asynchronous automaton whose size is a tower of exponentials of height d, where d is the depth of the eliminated tree, and whose parity index is the sum of all parity indices of all processes in the tree.

When the entire tree is reduced to a single process, we get a control problem for one process and the corollary follows. $\square$

The d-EXPTIME complexity bound stated in Corollary 1 is matched by a result from [12] showing that how to reduce acceptance of a d-EXPSPACE bounded Turing machine to a Control Problem on a *tree architecture* of depth $O(d)$ (although not exactly d, so some gap remains).

5 Conclusion

We revisited simpler constructions of Zielonka asynchronous automata, and the control problem, in the case of tree-like process architectures. We showed that the quadratic construction of Zielonka automata generalizes from trees to tree-like architectures, and also showed how this improves on a previous, exponential construction of asynchronous automata for triangulated dependence alphabets. Finally we showed that distributed controller synthesis generalizes smoothly to tree-like architectures, too.

While causal synchronisation like in the Zielonka model may appear far from real systems, it represents a first step to handle communication. Communication has different flavor than rendez-vous mechanisms like in the Zielonka model, e.g. it lacks the symmetry of Zielonka synchronisations since receivers are better informed than senders. At the same time, causal dependencies, like those arising in the Zielonka model, are important and can be approximated by practical forms of communication (e.g., local broadcast). In such a context, we believe that tree-like process architectures may provide more potential for future applications involving communication, than tree architectures. Indeed, tree-like architectures allow for real multi-party communication as channels are not restricted to binary connections. It adds the requirement to inform all the “region” affected by a communication, suggesting possible uses where interaction is connected to physical location. In our future work, we are working on an implementation of a communication infrastructure that supports a relaxed version of Zielonka type synchronization. Our framework ensures that actions happen in the same partial order as in the theoretical (Zielonka) model. The communication framework currently enacts asynchronous automata that are modelled by hand. However, if the communication architecture is tree-like we could use an automated distribution of centrally crafted plans based on similar ideas as in this paper. At a theoretical level, we plan to investigate which central plans can be distributed into a tree-like communication framework with bounded communication channels.

References

1. Bharat Adsul, Paul Gastin, Shantanu Kulkarni, and Pascal Weil. An expressively complete local past propositional dynamic logic over Mazurkiewicz traces and its applications. In Pawel Sobocinski, Ugo Dal Lago, and Javier Esparza, editors, *Proceedings of the 39th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2024, Tallinn, Estonia, July 8-11, 2024*, pages 2:1–2:13. ACM, 2024. doi:10.1145/3661814.3662110.
2. S. Akshay, Ionut Dinca, Blaise Genest, and Alin Stefanescu. Implementing realistic asynchronous automata. In *FSTTCS'13*, LIPIcs, pages 213–224. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2013.
3. Raven Beutner, Bernd Finkbeiner, and Jesko Hecking-Harbusch. Translating asynchronous games for distributed synthesis. In *International Conference on Concurrency Theory (CONCUR'19)*, volume 140 of *LIPIcs*, pages 26:1–26:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.
4. Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. Generalized parity games. In Helmut Seidl, editor, *Foundations of Software Science and Computational Structures, 10th International Conference, FOSSACS 2007, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2007, Braga, Portugal, March 24-April 1, 2007, Proceedings*, volume 4423 of *Lecture Notes in Computer Science*, pages 153–167. Springer, 2007. doi:10.1007/978-3-540-71389-0_12.
5. Robert Cori, Yves Métivier, and Wieslaw Zielonka. Asynchronous mappings and asynchronous cellular automata. *Inf. Comput.*, 106(2):159–202, 1993. URL: https://doi.org/10.1006/inco.1993.1052, doi:10.1006/INCO.1993.1052.
6. Volker Diekert and Anca Muscholl. A note on Métivier's construction of asynchronous automata for triangulated graphs. *Fundam. Informaticae*, 25(3):241–246, 1996. doi:10.3233/FI-1996-253402.
7. Volker Diekert and Grzegorz Rozenberg, editors. *The Book of Traces.* World Scientific, 1995. doi:10.1142/2563.
8. Bernd Finkbeiner and Sven Schewe. Uniform distributed synthesis. In *20th Annual IEEE Symposium on Logic in Computer Science (LICS' 05)*, pages 321–330, 2005. doi:10.1109/LICS.2005.53.
9. Paul Gastin, Nathalie Sznajder, and Marc Zeitoun. Distributed synthesis for well-connected architectures. *Formal Methods Syst. Des.*, 34(3):215–237, 2009. URL: https://doi.org/10.1007/s10703-008-0064-7, doi:10.1007/S10703-008-0064-7.
10. Fănică Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16(1):47–56, 1974. URL: https://www.sciencedirect.com/science/article/pii/009589567490094X, doi:https://doi.org/10.1016/0095-8956(74)90094-X.
11. Blaise Genest, Hugo Gimbert, Anca Muscholl, and Igor Walukiewicz. Optimal Zielonka-type construction of deterministic asynchronous automata. In *Automata, Languages and Programming, ICALP 2010*, volume 6199 of *Lecture Notes in Computer Science*, pages 52–63. Springer, 2010. doi:10.1007/978-3-642-14162-1_5.
12. Blaise Genest, Hugo Gimbert, Anca Muscholl, and Igor Walukiewicz. Asynchronous games over tree architectures. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*, volume 7966 of *Lecture Notes in Computer Science*, pages 275–286. Springer, 2013. doi:10.1007/978-3-642-39212-2_26.

13. Blaise Genest and Anca Muscholl. Constructing exponential-size deterministic Zielonka automata. In *Automata, Languages and Programming, ICALP 2006*, pages 565–576. Springer, 2006.
14. Hugo Gimbert. Distributed asynchronous games with causal memory are undecidable. *Logical Methods in Computer Science*, Volume 18, Issue 3, 09 2022. doi:10.46298/lmcs-18(3:30)2022.
15. Daniel Hausmann, Mathieu Lehaut, and Nir Piterman. Distribution of reconfiguration languages maintaining tree-like communication topology. In *Automated Technology for Verification and Analysis - 22nd International Symposium, ATVA 2024, Kyoto, Japan, October 21-24, 2024, Proceedings*, Lecture Notes in Computer Science. Springer, 2024.
16. Siddharth Krishna and Anca Muscholl. A quadratic construction for Zielonka automata with acyclic communication structure. *Theor. Comput. Sci.*, 503:109–114, 2013. doi:10.1016/j.tcs.2013.07.015.
17. Orna Kupferman and Moshe Y. Vardi. Synthesizing distributed systems. In *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science*, LICS '01, page 389, USA, 2001. IEEE Computer Society.
18. Stephane Lafortune, Karen Rudie, and Stavros Tripakis. Thirty years of the Ramadge-Wonham theory of supervisory control: A retrospective and future perspectives [conference reports]. *IEEE Control Systems Magazine*, 38(4):111–112, 2018. doi:10.1109/MCS.2018.2830083.
19. Mathieu Lehaut, Anca Muscholl, and Nir Piterman. From trees to tree-like: Distribution and synthesis for asynchronous automata, 2026. URL: https://arxiv.org/abs/2601.14078, arXiv:2601.14078.
20. Christoph Löding. Methods for the transformation of automata: Complexity and connection to second order logic. Master's thesis, Christian-Albrechts-University of Kiel, Kiel, Germany, 1999. Available at https://www.lics.rwth-aachen.de/global/show_document.asp?id=aaaaaaaaabcqdty.
21. Antoni Mazurkiewicz. Concurrent program schemes and their interpretations. *DAIMI Report Series*, (78), 1977.
22. Madhavan Mukund and Milind A. Sohoni. Keeping track of the latest gossip in a distributed system. *Distributed Comput.*, 10(3):137–148, 1997. doi:10.1007/S004460050031.
23. Anca Muscholl and Igor Walukiewicz. Distributed synthesis for acyclic architectures. *arXiv preprint arXiv:1402.3314*, 2014. Conference version FST&TCS 2014.
24. A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *Proceedings [1990] 31st Annual Symposium on Foundations of Computer Science*, pages 746–757 vol.2, 1990. doi:10.1109/FSCS.1990.89597.
25. Peter Ramadge and Walter Wonham. Supervisory control of a class of discrete event systems. *SIAM Journal on Control and Optimization*, 25:206–230, 01 1987. doi:10.1137/0325013.
26. Shmuel Safra. Exponential determinization for omega-automata with strong-fairness acceptance condition (extended abstract). In S. Rao Kosaraju, Mike Fellows, Avi Wigderson, and John A. Ellis, editors, *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, pages 275–282. ACM, 1992. doi:10.1145/129712.129739.
27. Wieslaw Zielonka. Notes on finite asynchronous automata. *RAIRO Theor. Informatics Appl.*, 21(2):99–135, 1987. URL: https://doi.org/10.1051/ita/1987210200991.