On the Control of Asynchronous Automata*†

Hugo Gimbert

LaBRI, CNRS, Université de Bordeaux, France
hugo.gimbert@cnrs.fr

Abstract

The decidability of the distributed version of the Ramadge and Wonham controller synthesis problem [11], where both the plant and the controllers are modeled as asynchronous automata [12, 1] and the controllers have causal memory is a challenging open problem [9, 7]. There exist three classes of plants for which the existence of a correct controller with causal memory has been shown decidable: when the dependency graph of actions is series-parallel, when the processes are connectedly communicating and when the dependency graph of processes is a tree. We design a class of plants, called decomposable games, with a decidable controller synthesis problem. This provides a unified proof of the three existing decidability results as well as new examples of decidable plants.

1998 ACM Subject Classification B.1.2 Automatic synthesis, H.3.4 Distributed systems

Keywords and phrases Asynchronous automata, Controller synthesis

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.30

1 Introduction

The decidability of the distributed version of the Ramadge and Wonham control problem [11], where both the plant and the controllers are modeled as asynchronous automata [12, 1] and the controllers have causal memory is a challenging open problem. Very good introductions to this problem are given in [9, 7].

In this setting a controllable plant is distributed on several finite-state processes which interact asynchronously using shared actions. On every process, the local controller can choose to block some of the actions, called *controllable* actions, but it cannot block the *uncontrollable* actions from the environment. The choices of the local controllers are based on two sources of information.

- First the controller monitors the sequence of states and actions of the local process. This information is called the *local view* of the controller.
- Second when a shared action is played by several processes then all the controllers of these processes can exchange as much information as they want. In particular together they can compute their mutual view of the global execution: their *causal past*.

A controller is correct if it guarantees that every possible execution of the plant satisfies some specification. The controller synthesis problem is a decision problem which, given a plant as input, asks whether the system admits a correct controller. In case such a controller exists, the algorithm should compute one as well.

The difficulty of controller synthesis depends on several factors, e.g.:

- the size and architecture (pipeline, ring, ...) of the system,

* A full version of this paper, including proofs, is available as a technical report [5].
† We acknowledge support from ANR-13-BS02-0011 "Stoch-MC" and UMI Relax.

37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017).
Editors: Satya Lokam and R. Ramanujam; Article No. 30; pp. 30:1–30:15

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

- the information available to the controllers,
- the specification.

Assuming that processes can exchange information upon synchronization and use their causal past to take decisions is one of the key aspects to get decidable synthesis problems [3]. In early work on distributed controller synthesis, for example in the setting of [10], the only source of information available to the controllers is their local view. In this setting, distributed synthesis is not decidable in general, except for very particular architectures like the pipeline architecture. The paper [2] proposes information forks as a uniform notion explaining the (un)decidability results in distributed synthesis. The idea of using causal past as a second source of information appeared in [3].

We adopt a modern terminology and call the plant a *distributed game* and the controllers are *distributed strategies* in this game. A distributed strategy is a function that maps the causal past of processes to a subset of controllable actions. In the present paper we focus on the *termination condition*, which is satisfied when each process is guaranteed to terminate its computation in finite time, in a final state. A distributed strategy is winning if it guarantees the termination condition, whatever uncontrollable actions are chosen by the environment.

We are interested in the following problem, whose decidability is an open question.

DISTRIBUTED SYNTHESIS PROBLEM: given a distributed game decide whether there exists a winning strategy.

There exist three classes of plants for which the DISTRIBUTED SYNTHESIS PROBLEM has been shown decidable:

1. when the dependency graph of actions is series-parallel [3],
2. when the processes are connectedly communicating [6],
3. and when the dependency graph of processes is a tree [4, 8].

A series-parallel game is a game such that the dependency graph of the alphabet A is a co-graph. Series-parallel games were proved decidable in [3], for a different setup than ours: in the present paper we focus on process-based control while [3] was focusing on action-based control. Actually action-based control is more general than process-based control, see [9] for more details. The results of the present paper could probably be extended to action-based control however we prefer to stick to process-based control in order to keep the model intuitive. To our knowledge, the result of [3] was the first discovery of a class of asynchronous distributed system with causal memory for which the DISTRIBUTED SYNTHESIS PROBLEM is decidable.

Connectedly communicating games have been introduced [6]. A game is connectedly communicating if there is a bound k such that if a process p executes k steps in parallel to another process q then all further actions of p will be parallel to q. The event structure of a connectedly communicating games has a decidable MSO theory [6] which implies that the DISTRIBUTED SYNTHESIS PROBLEM is decidable for these games.

An acyclic game is a game where processes are arranged as a tree and actions are either local or synchronize a father and its son. Even in this simple setting the DISTRIBUTED SYNTHESIS PROBLEM is non-elementary hard [4].

Our contribution

We develop a new proof technique to address the DISTRIBUTED SYNTHESIS PROBLEM, and provide a unified proof of decidability for series-paralell, connectedly communicating and acyclic games. We design a class of games, called *decomposable games*, for which the

DISTRIBUTED SYNTHESIS PROBLEM is decidable. This leads to new examples of decidable architectures for controller synthesis.

The winning condition of the present paper is the termination of all processes in a final state. Richer specifications can be expressed by parity conditions. In the present paper we stick to termination conditions for two reasons. First, the long-term goal of this research is to establish the decidability or undecidability of the distributed controller synthesis problem. A possible first step is to prove decidability for games with termination conditions. Second, it seems that the results of the present paper can be lifted to parity games, using the same concepts but at the cost of some extra technical details needed to reason about infinite plays.

Our proof technique consists in simplifying a winning strategy by looking for useless parts to be removed in order to get a smaller winning strategy. These parts are called *useless repetitions*. Whenever a useless repetition exists, we remove it using an operation called a *shortcut* in order to get a simpler strategy. Intuitively, a shortcut is a kind of cut-and-paste operation which makes the strategy smaller. By taking shortcuts again and again, we make the strategy smaller and smaller, until it does not have any useless repetition anymore.

If a winning strategy exists, there exists one with no useless repetition. In decomposable games, there is a computable upper bound on the size of strategies with no useless repetition, which leads to decidability of the controller synthesis problem.

Performing cut-and-paste in a distributed game is not as easy as doing it in a single-process game. In a single-process game, strategies are trees and one can cut a subtree from a node A and paste it to any other node B, and the operation makes sense as long as the state of the process is the same in both nodes. In the case of a general distributed strategy, designing cut-and-paste operations is more challenging. Such operations on the strategy tree should be consistent with the level of information of each process, in order to preserve the fundamental property of distributed strategies: the decisions taken by a process should depend only on its causal view, not on parallel events.

The decidability of series-parallel games established in [3] relies also on some simplification of the winning strategies, in order to get *uniform* strategies. The series-parallel assumption is used to guarantee that the result of the replacement of a part of a strategy by a uniform strategy is still a strategy, as long as the states of all processes coincide. Here we work without the series-parallel assumption, and matching the states is not sufficient for a cut-and-paste operation to be correct.

This is the reason for introducing the notion of *lock*. A lock is a part of a strategy where information is guaranteed to spread in a team of processes before any of these processes synchronize with a process outside the team. When two locks A and B are similar, in some sense made precise in the paper, the lock B can be cut and paste on lock A. Upon arrival on A, a process of the team initiates a change of strategy, which progressively spreads across the team. All processes of the team should eventually play as if the play from A to B had already taken place, although it actually did not.

The complexity of our algorithm is really bad, so probably this work has no immediate practical applications. This is not surprising since the problem is non-elementary even for the class of acyclic games [4]. Nevertheless we think this paper sheds new light on the difficult open problem of distributed synthesis.

Organization of the paper

Section 2 introduces the DISTRIBUTED SYNTHESIS PROBLEM. Section 3 provides several examples. In section 4 we show how to simplify strategies which contain useless repetitions, and prove that if a winning strategy exists, there exists one without any useless repetition.

Finally, section 5 introduces the class of decomposable games and show their controller synthesis problem is decidable. A full version of this paper, including proofs, is available as a technical report [5].

2 The distributed synthesis problem

The theory of Mazurkiewicz traces is very rich, for a thorough presentation see [1]. Here we only fix notations and recall the notions of traces, views, prime traces and parallel traces.

We fix an alphabet A and a symmetric and reflexive dependency relation $D \subseteq A \times A$ and the corresponding independency relation $\mathbb{I} \subseteq A \times A$ defined as $\forall a, b \in A, (a \mathbb{I} b) \iff (a, b) \notin D$. A *Mazurkiewicz trace* or, more simply, a *trace*, is an equivalence class for the smallest equivalence relation $\equiv$ on A^* which commutes independent letters i.e. for every letters a, b and every words w_1, w_2,

$$a \mathbb{I} b \implies w_1abw_2 \equiv w_1baw_2 \ .$$

The words in the equivalence class are the *linearizations* of the trace. The trace whose only linearization is the empty word is denoted ϵ. All linearizations of a trace u have the same set of letters and length, denoted respectively $\mathrm{Alph}(u)$ and $|u|$. Given $B \subseteq A$, the set of traces such that $\mathrm{Alph}(u) \subseteq B$ is denoted $B^*_{\equiv}$ in particular the set of all traces is $A^*_{\equiv}$.

The concatenation on words naturally extends to traces. Given two traces $u, v \in A^*_{\equiv}$, the trace uv is the equivalence class of any word in uv. The prefix relation $\sqsubseteq$ is defined by $(u \sqsubseteq v \iff \exists w \in A^*_{\equiv}, uw = v)$. and the suffix relation is defined similarly.

Maxima, prime traces and parallel traces

A letter $a \in A$ is a *maximum* of a trace u if it is the last letter of one of the linearizations of u. A trace $u \in A^*_{\equiv}$ is *prime* if it has a unique maximum, denoted $\mathrm{last}(u)$ and called the last letter of u. Two prime traces u and v are said to be *parallel* if

- neither u is a prefix of v nor v is a prefix of u; and
- there is a trace w such that both u and v are prefixes of w. These notions are illustrated on Fig. 1.

Processes and automata

Asynchronous automata are to traces what finite automata are to finite words, as witnessed by Zielonka's theorem [12]. An asynchronous automaton is a collection of automata on finite words, whose transition tables do synchronize on certain actions.

▶ **Definition 1.** An asynchronous automaton on alphabet A with processes $\mathbb{P}$ is a tuple $\mathcal{A} = ((A_p)_{p\in\mathbb{P}}, (Q_p)_{p\in\mathbb{P}}, (i_p)_{p\in\mathbb{P}}, (F_p)_{p\in\mathbb{P}}, \Delta)$ where:

- every process $p \in \mathbb{P}$ has a set of actions A_p, a set of states Q_p and $i_p \in Q_p$ is the initial state of p and $F_p \subseteq Q_p$ its set of final states.
- $A = \bigcup_{p\in\mathbb{P}} A_p$. For every letter $a \in A$, the domain of a is $\mathrm{dom}(a) = \{p \in \mathbb{P} \mid a \in A_p\}$.
- Δ is a set of transitions of the form $(a, (q_p, q'_p)_{p\in\mathrm{dom}(a)})$ where $a \in A$ and $q_p, q'_p \in Q_p$. Transitions are *deterministic*: for every $a \in A$, if $\delta = (a, (q_p, q'_p)_{p\in\mathrm{dom}(a)}) \in \Delta$ and $\delta' = (a, (q_p, q''_p)_{p\in\mathrm{dom}(a)}) \in \Delta$ then $\delta = \delta'$ (hence $\forall p \in \mathrm{dom}(a), q'_p = q''_p$).

Such an automaton works asynchronously: each time a letter a is processed, the states of the processes in $\mathrm{dom}(a)$ are updated according to the corresponding transition, while the

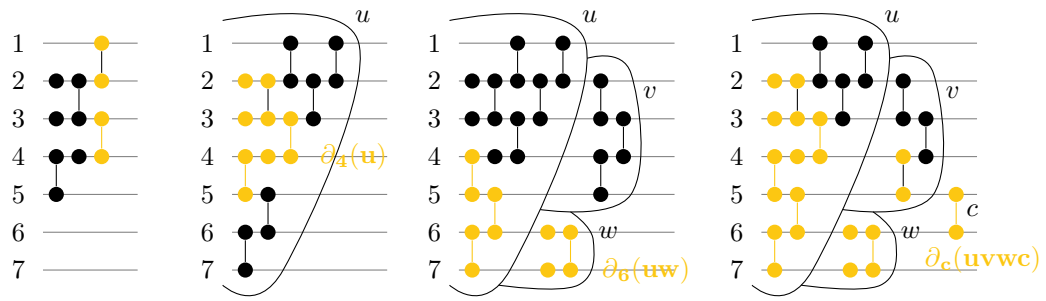

Figure 1 The set processes is $\{1 \dots 7\}$. A letter is identified with its domain. Here the domains are either singletons, represented by a single dot, or pairs of contigous processes, represented by two dots connected with a vertical segment. The trace $\{2\}\{3\}\{4,5\}\{2,3\}\{4\}\{1,2\}\{3,4\} = \{4,5\}\{4\}\{2\}\{3\}\{2,3\}\{3,4\}\{1,2\}$ is represented on the left-handside. It has two maximal letters $\{1,2\}$ and $\{3,4\}$ thus is not prime. Center left: process 4 sees only its causal view $\partial_4(u)$ (in yellow). Center right: $uvw = uwv$ since $\mathrm{dom}(v) \cap \mathrm{dom}(w) = \emptyset$. Both uv and $\partial_6(uw)$ (in yellow) are prime prefixes of uvw and they are parallel. Right: uv and $\partial_c(uvwc)$ (in yellow) are parallel.

states of other processes do not change. This induces a natural commutation relation $\mathbb{I}$ on A: two letters commute iff they have no process in common i.e.

$$(a \mathbb{I} b) \iff (\mathrm{dom}(a) \cap \mathrm{dom}(b) = \emptyset) \ .$$

The set of *plays* of the automaton $\mathcal{A}$ is a set of traces denoted plays($\mathcal{A}$) and defined inductively, along with a mapping state : plays($\mathcal{A}$) $\to \Pi_{p\in\mathbb{P}} Q_p$.

- ϵ is a play and $\mathrm{state}(\epsilon) = (i_p)_{p\in\mathbb{P}}$,
- for every play u such that $(\mathrm{state}_p(u))_{p\in\mathbb{P}}$ is defined and $\big(a, (\mathrm{state}_p(u), q_p)_{p\in\mathrm{dom}(a)}\big)$ is a transition then ua is a play and $\forall p \in \mathbb{P}, \mathrm{state}_p(ua) = \begin{cases} \mathrm{state}_p(u) & \text{if } p \notin \mathrm{dom}(a), \\ q_p & \text{otherwise.} \end{cases}$

For every play u, state(u) is called the *global state* of u. The inductive definition of state(u) is correct because it is invariant by commutation of independent letters of u.

Counting actions of a process

For every trace u we can count how many times a process p has played an action in u, which we denote $|u|_p$. Formally, $|u|_p$ is first defined for words, as the length of the projection of u on A_p, which is invariant by commuting letters. The domain of a trace is defined as

$$\mathrm{dom}(u) = \{p \in \mathbb{P} \mid |u|_p \neq 0\} \ .$$

Views, strategies and games

Given an automaton $\mathcal{A}$, we want the processes to choose actions which guarantee that every play eventually terminates in a final state.

To take into account the fact that some actions are controllable by processes while some other actions are not, we assume that A is partitioned in

$$A = A_c \sqcup A_e$$

where A_c is the set of controllable actions and A_e the set of (uncontrollable) environment actions. Intuitively, processes cannot prevent their environment to play actions in A_e, while they can decide whether to block or allow any action in A_c.

We adopt a modern terminology and call the automaton $\mathcal{A}$ together with the partition $A = A_c \sqcup A_e$ a *distributed game*, or even more simply a *game*. In this game the processes play distributed strategies, which are individual plans of action for each process. The choice of actions by a process p is dynamic: at every step, p chooses a new set of controllable actions, depending on its information about the way the play is going on. This information is limited since processes cannot communicate together unless they synchronize on a common action. In that case however they exchange as much information about the play as they want. Finally, the information missing to a process is the set of actions which happened in parallel of its own actions. The information which remains is called the p-view of the play, it is illustrated on Fig. 1 and defined formally as follows.

▶ **Definition 2** (Views)**.** For every set of processes $\mathbb{Q} \subseteq \mathbb{P}$ and trace u, the $\mathbb{Q}$-*view* of u, denoted $\partial_{\mathbb{Q}}(u)$, is the unique trace such that u factorizes as $u = \partial_{\mathbb{Q}}(u) \cdot v$ and v is the longest suffix of u such that $\mathbb{Q} \cap \mathrm{dom}(v) = \emptyset$. In case $\mathbb{Q}$ is a singleton $\{p\}$ the view is denoted $\partial_p(u)$ and is either empty or prime. For every letter $a \in A$ we denote $\partial_a(u) = \partial_{\mathrm{dom}(a)}(u)$.

Some useful properties of the $\mathbb{Q}$-view are:

$$\partial_{\mathbb{Q}}(uv) = \partial_{\mathbb{Q}'}(u)\,\partial_{\mathbb{Q}}(v) \text{ where } \mathbb{Q}' = \mathbb{Q} \cup \mathrm{dom}(\partial_{\mathbb{Q}}(v)) \tag{1}$$

$$(\mathbb{Q} \subseteq \mathbb{Q}') \implies (\partial_{\mathbb{Q}}(u) \sqsubseteq \partial_{\mathbb{Q}'}(u)) \ . \tag{2}$$

We can now define what is a distributed strategy.

▶ **Definition 3** (Distributed strategies, consistent and maximal plays)**.** Let $G = (\mathcal{A}, A_c, A_e)$ be a distributed game. A *strategy for process* p in G is a mapping which associates with every play u a set of actions $\sigma_p(u)$ such that:

- environment actions are allowed: $A_e \subseteq \sigma_p(u)$,
- the decision depends only on the view of the process: $\sigma_p(u) = \sigma_p(\partial_p(u))$.

A *distributed strategy* is a tuple $\sigma = (\sigma_p)_{p \in \mathbb{P}}$ where each σ_p is a strategy of process p. A play $u = a_1 \cdots a_{|u|} \in \mathrm{plays}(\mathcal{A})$ is *consistent with* σ, or equivalently is a σ-*play* if:

$$\forall i \in 1 \ldots |u|, \forall p \in \mathrm{dom}(a_i), a_i \in \sigma_p(a_1 \cdots a_{i-1}) \ .$$

A σ-play is *maximal* if it is not the strict prefix of another σ-play.

Note that a strategy is forced to allow every environment action to be executed at every moment. This may seem to be a huge strategic advantage for the environment. However depending on the current state, not every action can be effectively used in a transition because the transition function is not assumed to be total. So in general not every environment actions can actually occur in a play. In particular it may happen that a process enters a final state with no outgoing transition, where no uncontrollable action can happen.

Winning games

Our goal is to synthesize strategies which ensure that the game terminates and all processes are in a final state.

▶ **Definition 4** (Winning strategy)**.** A strategy σ is winning if the set of σ-plays is finite and in every maximal σ-play u, every process is in a final state i.e. $\forall p \in \mathbb{P}, \mathrm{state}_p(u) \in F_p$.

We are interested in the following problem, whose decidability is an open question.

DISTRIBUTED SYNTHESIS PROBLEM: given a distributed game decide whether there exists a winning strategy.

If the answer is positive, the algorithm should compute a winning strategy as well.

3 Three decidable classes

Series-parallel games. A game is *series-parallel* if its dependency alphabet (A, D) is a co-graph i.e. belongs to the smallest class of graphs containing singletons and closed under parallel product and complementation. In this case A has a *decomposition tree*, this is a binary tree whose nodes are subsets of A, its leaves are the singletons $(\{a\})_{a\in A}$, its root is A. Moreover every node B with two children B_0 and B_1 is the disjoint union of B_0 and B_1 and either $B_0 \times B_1 \subseteq D$ (serial product) or $(B_0 \times B_1) \cap D = \emptyset$ (parallel product).

The synthesis problem is decidable for series-parallel games [3].

Connectedly communicating games. A game is *k-connectedly communicating* if for every pair p, q of processes, if process p plays k times in parallel of process q then all further actions of q will be parallel to p. Formally, for every prime play uvw, $(q \notin \mathrm{dom}(v)$ and $|v|_p \geq k) \implies q \notin \mathrm{dom}(w)$.

The MSO theory of the event structure of a k-connectedly communicating game is decidable [6], which implies that controller synthesis is decidable for theses games.

Acyclic games. An acyclic game is a game where processes $\mathbb{P}$ are the nodes of a tree $T_{\mathbb{P}}$ and the domain of every action is a connected set of nodes of $T_{\mathbb{P}}$. The synthesis problem is known to be decidable for acyclic games such that the domain of each action has size 1 or 2 [4].

4 Simplifying strategies

In this section we present an elementary operation called a *shortcut*, which can be used to simplify and reduce the duration of a winning strategy.

To create a shortcut, one selects a σ-play xy and modifies the strategy σ so that as soon as any of the processes sees the play x in its view, this process assumes that not only x but also xy has actually occurred. In other words, a shortcut is a kind of *cut-and-paste* in the strategy: we glue on node x the sub-strategy rooted at node xy.

The choice of x and y should be carefully performed so that the result of the shortcut is still a strategy. We provide a sufficient condition for that: (x, y) should be a *useless repetition*.

The interest of taking shortcuts is the following: if the original strategy is winning, then the strategy obtained by taking the shortcut is winning as well, and strictly smaller than the original one. In the remainder of this section, we formalize these concepts.

4.1 Locks

We need to limit the communication between a set of processes, called a team, and processes outside the team. This leads to the notion of a $\mathbb{Q}$-lock: this is a prime play u such that there is no synchronization between $\mathbb{Q}$ and $\mathbb{P} \setminus \mathbb{Q}$ in parallel of u.

▶ **Definition 5.** Let $\mathbb{Q} \subseteq \mathbb{P}$. An action b is $\mathbb{Q}$-safe if ($\mathrm{dom}(b) \subseteq \mathbb{Q}$ or $\mathrm{dom}(b) \cap \mathbb{Q} = \emptyset$). A play u is a $\mathbb{Q}$-*lock* if it is prime and the last action of every prime play parallel to u is $\mathbb{Q}$-safe.

The notion of lock is illustrated on the right handside of Fig. 1. Set $\mathbb{Q} = \{1, 2, 3, 4, 5\}$. Then uv is not a $\mathbb{Q}$-lock because $\partial_c(uvwc)$ is parallel to uv but c is not $\mathbb{Q}$-safe. Locks occur in a variety of situations, including the three decidable classes.

▶ **Lemma 6** (Sufficient conditions for $\mathbb{Q}$-locks)**.** *Let u be a prime play of a game G and $\mathbb{Q} \subseteq \mathbb{P}$. Each of the following conditions is sufficient for u to be a $\mathbb{Q}$-lock:*

(i) $\mathbb{Q} = \mathbb{P}$.
(ii) *u is a $(\mathbb{P} \setminus \mathbb{Q})$-lock.*
(iii) $\mathbb{Q} \subseteq \mathrm{dom}(\mathrm{last}(u))$.
(iv) *The game is series-parallel and $\mathbb{Q} = \mathrm{dom}(B)$ where B is the smallest node of the decomposition tree of A which contains* $\mathrm{Alph}(u)$.
(v) *The game is connectedly communicating game with bound k, $\mathbb{Q} = \mathrm{dom}(u)$ and $\forall p \in dom(u), |u|_p \geq k$.*
(vi) *The game is acyclic with respect to a tree $T_{\mathbb{P}}$ and $\mathbb{Q}$ is the set of descendants in $T_{\mathbb{P}}$ of the processes in* $\mathrm{dom}(\mathrm{last}(u))$.
(vii) *There are two traces x and z such that $u = xz$ and z is a $\mathbb{Q}$-lock in the game G_x identical to G except the initial state is changed to* $\mathrm{state}(x)$.

4.2 Taking shortcuts

In this section we present a basic operation used to simplify a strategy, called a *shortcut*, which consists in modifying certain parts of a strategy, called *useless repetitions*. These notions rely on the notion of *strategic state* as well as two operations on strategies called *shifting* and *projection*.

▶ **Definition 7** (Residual)**.** Let σ be a strategy, u a σ-play and $\mathbb{Q} \subseteq \mathbb{P}$. The $\mathbb{Q}$-residual of σ after u is the set:

$$\pi(\sigma, u, \mathbb{Q}) = \{(v, \sigma(uv)) \mid v \in A^*_{\equiv}, \mathrm{dom}(v) \subseteq \mathbb{Q} \text{ and } uv \text{ is a } \sigma\text{-play.}\} \ .$$

A winning strategy may take unnecessarily complicated detours in order to ensure termination. Such detours are called *useless repetitions*.

▶ **Definition 8** (Strategic state)**.** Let $\mathbb{Q} \subseteq \mathbb{P}$ be a set of processes, σ a strategy and u a prime σ-play with maximal letter b. The strategic $\mathbb{Q}$-state of σ after u is the tuple

$$\mathrm{strate}_{\sigma,\mathbb{Q}}(u) = (b, \mathrm{state}(u), \pi\left(\sigma, u, \mathbb{Q} \setminus \mathrm{dom}(b)\right)) \ .$$

▶ **Definition 9** (Useless repetition)**.** A useless $\mathbb{Q}$-repetition in a strategy σ is a pair of traces (x, y) such that y is not empty, xy is a σ-play, $\mathrm{dom}(y) \subseteq \mathbb{Q}$, both x and xy are $\mathbb{Q}$-locks and $\mathrm{strate}_{\sigma,\mathbb{Q}}(x) = \mathrm{strate}_{\sigma,\mathbb{Q}}(xy)$.

The following theorem is the key to our decidability results.

▶ **Theorem 10.** *If there exists a winning strategy then there exists a winning strategy without any useless repetition.*

The proof of this theorem relies on the notion of shortcuts, an operation which turns a winning strategy into another strategy with strictly shorter duration.

▶ **Definition 11** (Duration of a strategy). The duration of a strategy σ is

$$\mathrm{dur}(\sigma) = \sum_{u \text{ maximal } \sigma\text{-play}} |u| \ .$$

The duration of a strategy σ may in general be infinite but is finite if σ is winning.

▶ **Lemma 12.** *Let (x, y) be a useless $\mathbb{Q}$-repetition in a strategy σ. Let $\Phi : A^*_{\equiv} \to A^*_{\equiv}$ and τ defined by* $\Phi(u) = \begin{cases} u \text{ if } x \not\sqsubseteq u \\ xyu' \text{ if } x \sqsubseteq u \text{ and } u = xu' \end{cases}$ *and*

$\forall p \in \mathbb{P}, \tau_p(u) = \sigma_p(\Phi(\partial_p(u)))$.

1. *Then τ is a strategy called the (x, y)*-shortcut of σ. *Moreover for every trace u,*

$$(u \text{ is a } \tau\text{-play}) \iff (\Phi(u) \text{ is a } \sigma\text{-play}) \ . \tag{3}$$

2. *If σ is a winning strategy then τ is winning as well and has a strictly smaller duration.*

The strategy τ can be summarized as asking to every process "(†) whenever play x has occurred, replace it by xy and apply σ". Claim 1 in Lemma 12 would not hold in general if (x, y) would not be a useless repetition. For example, assume $u = x$ in the definition above. In general, right after x has occurred, a process p which is not part of the domain of the maximal action of x is playing in parallel of x and sees a strict prefix $\partial_p(x)$ of x. Thus p does not know whether the play x has actually occurred. Asking this process to apply (†) is "cheating" because by definition of strategies, the decision of p after play x should be based only on $\partial_p(x)$. However, if (x, y) is a useless repetition then τ is a distributed strategy, which relies on the equality $\sigma_p(\Phi(\partial_p(u))) = \sigma_p(\partial_p(\Phi(u)))$ for every play u.

Claim 2 relies on $\mathrm{dur}(\tau) < \mathrm{dur}(\sigma)$ which follows immediately from (3). And according to (3) again, the set of global states of the maximal plays is the same for σ and τ thus if σ is winning then τ is winning as well.

Proof of Theorem 10. As long as there exists a useless repetition, take the corresponding shortcut. According to Lemma 12, this creates a sequence $\sigma_0, \sigma_1, \ldots$ of winning strategies whose duration strictly decreases. Thus the sequence is finite and its last element is a winning strategy without useless repetition. ◀

5 Decomposable games

In this section we introduce *decomposable* games, for which the DISTRIBUTED SYNTHESIS PROBLEM is decidable (Theorem 21). There are actually three notions of decomposability: structural decomposability, process decomposability and action decomposability. These three notions form a hierarchy: structural decomposability implies process decomposability which itself implies action decomposability (Lemma 19). Known decidable classes are decomposable: acyclic games are structurally decomposable (Lemma 14), connectedly-communicating games are process decomposable (Lemma 16) and series-parallel games are action decomposable (Lemma 18). Structural decomposability is stable under some operations between games which leads to new examples of decidable games (Lemma 26).

5.1 Decomposability

The notions of decomposability rely on *preorders* defined on $2^{\mathbb{P}}$ or 2^A. A preorder $\preceq$ is a reflexive and transitive relation. We denote $\prec$ the relation $(x \prec y) \iff (x \preceq y \wedge y \not\preceq x)$.

Structural decomposability. This notion of decomposability relies on a preorder $\preceq$ on $2^{\mathbb{P}}$ which is monotonic with respect to inclusion, i.e. $\forall \mathbb{Q}, \mathbb{Q}' \subseteq \mathbb{P}$, $(\mathbb{Q} \subseteq \mathbb{Q}' \implies \mathbb{Q} \preceq \mathbb{Q}')$.

▶ **Definition 13** (Structural decomposability)**.** A game is $\preceq$-*structurally decomposable* if for every non-empty prime trace $y \in A^*_{\equiv}$ there exists $\mathbb{Q} \supseteq \mathrm{dom}(y)$ and $b \in \mathrm{Alph}(y)$ such that:

$$(\mathbb{Q} \setminus \mathrm{dom}(b)) \prec \mathbb{Q}$$
$$\forall a \in A, (a \mathbb{I} b \implies a \text{ is } \mathbb{Q}\text{-safe}) \ .$$

We say a game is structurally decomposable if it is $\preceq$-structurally decomposable for some preorder $\preceq$. We have already seen one example of such games.

▶ **Lemma 14.** *Acyclic games are structurally decomposable.*

Proof. Assume the game is acyclic with process tree $T_{\mathbb{P}}$. Set $\mathbb{Q} \preceq \mathbb{Q}'$ iff every process in $\mathbb{Q}$ has a $T_{\mathbb{P}}$-ancestor in $\mathbb{Q}'$, which is monotonic with respect to inclusion. Let y be a prime trace, $p \in \mathbb{P}$ the least common ancestor in $T_{\mathbb{P}}$ of processes in $\mathrm{dom}(y)$ and $\mathbb{Q}$ the set of descendants of p. Then $\mathrm{dom}(y) \subseteq \mathbb{Q}$. Moreover, since y is prime and since the domain of every action is a connected subset of $T_{\mathbb{P}}$ then $\mathrm{dom}(y)$ is connected as well thus $p \in \mathrm{dom}(y)$ and there exists a letter $b \in \mathrm{Alph}(y)$ such that $p \in \mathrm{dom}(b)$. We show that b satisfies the conditions in the definition of structural decomposability. First, $(\mathbb{Q} \setminus \mathrm{dom}(b)) \preceq \mathbb{Q}$ and the inequality is strict because the only ancestor of p in $\mathbb{Q}$ is p itself and $p \in \mathrm{dom}(b)$. Second, let $a \in A$ such that $a \mathbb{I} b$. Then $p \notin \mathrm{dom}(a)$ and since $\mathrm{dom}(a)$ is connected in $T_{\mathbb{P}}$, then either none of the processes in $\mathrm{dom}(a)$ or all of them are descendants of p in $T_{\mathbb{P}}$, i.e. a is $\mathbb{Q}$-safe. ◀

Process decomposability. The definition of process decomposable games relies on a parameter $k \in \mathbb{N}$ and a preorder $\preceq$ on $2^{\mathbb{P}}$ which is monotonic with respect to inclusion.

▶ **Definition 15** (Process decomposable games)**.** Fix an integer k. A trace y is k-*repeating* if

$$y \text{ is not empty and } \forall p \in \mathrm{dom}(y), |y|_p \geq k \ .$$

A game is $(\preceq, k)$-*process decomposable* if for every prime play xy, if y is k-repeating then there exists $\mathbb{Q} \supseteq \mathrm{dom}(y)$ and a prime prefix $z \sqsubseteq y$ such that $\partial_{\mathrm{last}(z)}(xz)$ is a $\mathbb{Q}$-lock and

$$(\mathbb{Q} \setminus \mathrm{dom}(\mathrm{last}(z))) \prec \mathrm{dom}(y) \ . \tag{4}$$

We have already seen one example of process decomposable games.

▶ **Lemma 16.** *Connectedly communicating games are process decomposable.*

Action decomposability. Action decomposability is defined with respect to a parameter $k \in \mathbb{N}$ and a preorder $\preceq$ on 2^A which is monotonic with respect to inclusion.

▶ **Definition 17** (Action decomposable games)**.** Let k be an integer. A game is $(\preceq, k)$ *action decomposable* if for every prime play xy such that y is k-repeating, there exists $\mathbb{Q} \supseteq \mathrm{dom}(y)$ and a prime prefix $z \sqsubseteq y$ such that $\partial_{\mathrm{last}(z)}(xz)$ is a $\mathbb{Q}$-lock and

$$\{a \in A \mid \mathrm{dom}(a) \subseteq (\mathbb{Q} \setminus \mathrm{dom}(\mathrm{last}(z))\} \prec \mathrm{Alph}(y) \ .$$

We have already seen one example of action decomposable games.

▶ **Lemma 18.** *Series-parallel games are action decomposable.*

Finally we show that these notions form a hierarchy.

▶ **Lemma 19.** *Every structurally decomposable game is process decomposable and every process decomposable game is action decomposable.*

Thus *action decomposability* is the most general notion of decomposability. In the sequel for the sake of conciseness, it is simply called *decomposability*.

5.2 Decidability

In this section we show that decomposability is a decidable property and decomposable games have a decidable controller synthesis problem.

▶ **Lemma 20** (Decomposability is decidable). *Whether a game is decomposable is decidable. There exists a computable function* decomp *from games to integers such that whenever a game G is $(\preceq, k)$ decomposable for some k, it is $(\preceq, \mathrm{decomp}(G))$ decomposable.*

▶ **Theorem 21.** *The distributed synthesis problem is decidable for decomposable games.*

Proof of Theorem 21. We show that there exists a computable function f from games to integers such that in every decomposable distributed game G every strategy with no useless repetition has duration $\leq f(G)$.

Let $\preceq$ be a preorder on 2^A compatible with inclusion, k' an integer and G a $(\preceq, k')$ action decomposable distributed game. Assume $k' = \mathrm{decomp}(G)$ w.l.o.g. (cf. Lemma 20).

For every set of actions $B \subseteq A$, denote G_B the game with actions B and the same processes, initial state and final states than G. The transitions of G_B are all transitions of G whose action is in B. An action $a \in B$ is controllable in G_B iff it is controllable in G.

We show that for every $B \subseteq A$ the game G_B is $(\preceq_B, k')$ decomposable, where $\preceq_B$ denotes the restriction of $\preceq$ to 2^B. Let xy be a prime play of G_B such that y is k'-repeating. Since G is $(\preceq, k')$ decomposable, there exists $\mathbb{Q} \supseteq \mathrm{dom}(y)$ and a prime prefix $z \sqsubseteq y$ such that $\partial_{\mathrm{last}(z)}(xz)$ is a $\mathbb{Q}$-lock in G and $C \prec \mathrm{Alph}(y)$ where $C = \{a \in A \mid \mathrm{dom}(a) \subseteq \mathbb{Q} \text{ and } a \mathbb{I} \mathrm{last}(z)\}$. Since $\preceq$ is monotonic with respect to inclusion then $\{b \in B \mid \mathrm{dom}(b) \subseteq \mathbb{Q} \text{ and } b \mathbb{I} \mathrm{last}(z)\} = (C \cap B) \preceq C \prec \mathrm{Alph}(y)$ thus $(C \cap B) \prec_B \mathrm{Alph}(y)$. Since xy is a play in G_B then $\partial_{\mathrm{last}(z)}(xz) \sqsubseteq xy$ is a play in G_B as well. And since every play in G_B is a play in G, $\partial_{\mathrm{last}(z)}(xz)$ is a $\mathbb{Q}$-lock not only in G but also in G_B. All conditions of action decomposability are met : G_B is $(\preceq_B, k')$ decomposable.

Denote $R_B(m)$ the largest size of a complete undirected graph whose edges are labelled with 2^B and which contains no monochromatic clique of size $\geq m$. According to Ramsey's theorem, $R_B(m)$ is finite and computable. For every $B \subseteq A$, defined inductively $f(G_B)$ as :

$$f(G_B) = R_B\left((k' + |\mathbb{P}|) \cdot |B| \cdot |Q|^{|\mathbb{P}|} \cdot 2^{2^{|A||\mathbb{P}| \cdot \max\{f(G_{B'}), B' \prec B\}}}\right) \ ,$$

with the convention $\max \emptyset = 0$.

Fix a strategy σ with no useless repetition. We prove that for every σ-play zu,

$$|u| \leq f\left(G_{\mathrm{Alph}(u)}\right) \ . \tag{5}$$

The proof is by induction on $\mathrm{Alph}(u)$ with respect to $\preceq$. The base case when $\mathrm{Alph}(u) = \emptyset$ is easy, in this case $|u| = 0$.

Now let zu be a σ-play consistent with σ. Assume the induction hypothesis holds: for every σ-play $z'u'$, if $\mathrm{Alph}(u') \prec \mathrm{Alph}(u)$ then $|u'| \leq f\left(G_{\mathrm{Alph}(u')}\right)$.

We start with computing, for every non-empty set of letters $B \prec \mathrm{Alph}(u)$ an upper bound on the length of every factorization $u = u_0u_1 \cdots u_Nu_{N+1}$ such that

$$B = \mathrm{Alph}(u_1) = \mathrm{Alph}(u_2) = \ldots = \mathrm{Alph}(u_N) \ . \tag{6}$$

For a start, we consider the case where B is connected in the sense where the dependency graph $D_B = (B, D \cap B \times B)$ is connected. Set $k = k' + |\mathbb{P}|$. For $0 \leq \ell < \frac{N}{k}$, denote w_ℓ the concatenation $w_\ell = u_{1+\ell k} \cdot u_{2+\ell k} \cdots u_{k+\ell k}$ and $h_\ell = zu_0w_1 \ldots w_{\ell-1}$. Let $\mathbb{R}_B = \mathrm{dom}(B)$ and fix some $c \in B$.

Let $0 \leq \ell < \frac{N}{k}$. We show that $\partial_c(w_\ell)$ is k'-repeating and $\partial_c(h_\ell w_\ell) = \partial_{\mathbb{R}_B}(h_\ell)\, \partial_c(w_\ell)$. Since $w_\ell = u_{1+\ell k} \cdot u_{2+\ell k} \cdots u_{k+\ell k}$, according to property (1) of views there exists a sequence $\mathbb{P} \supseteq \mathbb{R}_1 \supseteq \ldots \supseteq \mathbb{R}_k$ such that

$$\partial_c(w_\ell) = \partial_{\mathbb{R}_1}(u_{1+\ell k})\, \partial_{\mathbb{R}_2}(u_{2+\ell k}) \cdots \partial_{\mathbb{R}_k}(u_{k+\ell k}) \tag{7}$$

where $\mathbb{R}_k = \{c\}$ and for every $1 \leq i \leq k-1$, $\mathbb{R}_i = \mathbb{R}_{i+1} \cup \mathrm{dom}(\partial_{\mathbb{R}_{i+1}}(u_{i+1+\ell k}))$. Since the sequence $(\mathbb{R}_i)_{1 \leq i \leq k'+|\mathbb{P}|}$ is monotonic, there exists $i \in k' \ldots k' + |\mathbb{P}|$ such that $\mathbb{R}_i = \mathbb{R}_{i+1}$. Denote $\mathbb{R} = \mathbb{R}_i = \mathbb{R}_{i+1}$ and $B' = \{b \in B, \mathrm{dom}(b) \cap \mathbb{R} \neq \emptyset\}$ and $B'' = \{b \in B, \mathrm{dom}(b) \subseteq \mathbb{R}\}$. By definition of views, and according to (6), $B' \subseteq \mathrm{Alph}(\partial_{\mathbb{R}}(u_{i+1+\ell k}))$. Since $\mathbb{R} = \mathbb{R}_i = \mathbb{R}_{i+1}$ and $\mathbb{R}_i = \mathbb{R}_{i+1} \cup \mathrm{dom}(\partial_{\mathbb{R}_{i+1}}(u_{i+1+\ell k}))$ then $\mathrm{dom}(\partial_{\mathbb{R}}(u_{i+1+\ell k})) \subseteq \mathbb{R}$ thus $\mathrm{Alph}(\partial_{\mathbb{R}}(u_{i+1+\ell k})) \subseteq B''$. Since $B'' \subseteq B'$ then finally $B' = \mathrm{Alph}(\partial_{\mathbb{R}}(u_{i+1+\ell k})) = B''$. Thus the set B'' is a connected component of the graph $D_B = (B, D \cap B \times B)$: by definition of B' and B'', all edges with source B'' have target in $B' = B''$. However by hypothesis D_B is connected thus $B = B' = B''$ and $\mathbb{R} = \mathbb{R}_B$. Finally $\mathbb{R}_B \subseteq \mathbb{R}_i \subseteq \mathbb{R}_1$ and since $\mathbb{R}_1 \subseteq \mathrm{dom}(\partial_c(w_\ell)) \subseteq \mathbb{R}_B$, the sequence $(\mathbb{R}_i)_{1 \leq i' \leq i}$ is constant equal to $\mathbb{R}_B$. Thus, according to (6) and the definition of $\mathbb{R}_B$, for every $1 \leq i' \leq i$, $\partial_{\mathbb{R}_{i'}}(u_{i'+\ell k}) = u_{i'+\ell k}$. Thus, according to (6) and (7) and since $k' \leq i'$, every letter of B occurs at least k' times in $\partial_c(w_\ell)$ thus $\partial_c(w_\ell)$ is k'-repeating and $\partial_c(h_\ell w_\ell) = \partial_{\mathbb{R}_B}(h_\ell)\, \partial_c(w_\ell)$.

Since the game is $(\preceq, k')$ decomposable and $\partial_c(w_\ell)$ is k'-repeating, and $\partial_c(h_\ell w_\ell) = \partial_{\mathbb{R}_B}(h_\ell)\, \partial_c(w_\ell)$, there exists a superset $\mathbb{T}^{(\ell)}$ of $\mathbb{R}_B$, an action b_ℓ, and a prime prefix $w'_\ell b_\ell \sqsubseteq \partial_c(w_\ell)$ such that the play $z_\ell = \partial_{b_\ell}(\partial_{\mathbb{R}_B}(h_\ell) w'_\ell b_\ell)$ is a $\mathbb{T}^{(\ell)}$-lock and $B_\ell \prec B$ where $B_\ell = \{a \in A \mid \mathrm{dom}(a) \subseteq (\mathbb{T}^{(\ell)} \setminus \mathrm{dom}(b_\ell))\}$.

For every $0 \leq \ell < \frac{N}{k}$, denote $\mathrm{strate}_\ell = \left(b_\ell, (s_{\ell,p})_{p \in \mathbb{P}}, \sigma^{(\ell)}\right)$ the $\mathbb{T}^{(\ell)}$ strategic state of σ after z_ℓ. We show two properties of $(\mathrm{strate}_\ell)_{0 \leq \ell < \frac{N}{k}}$.

- First, all elements of $(\mathrm{strate}_\ell)_{0 \leq \ell < \frac{N}{k}}$ are distinct. For the sake of contradiction, assume $\mathrm{strate}_\ell = \mathrm{strate}_{\ell'}$ for some $0 \leq \ell < \ell' < \frac{N}{k}$. We show that $z_\ell \sqsubset z_{\ell'}$. Since $\mathrm{strate}_\ell = \mathrm{strate}_{\ell'}$ then $b_\ell = b_{\ell'}$, denote this letter b. Then

$$\begin{aligned} z_\ell = \partial_b(\partial_{\mathbb{R}_B}(h_\ell) w'_\ell b) \sqsubseteq \partial_b(\partial_{\mathbb{R}_B}(h_\ell)\, \partial_c(w_\ell)) = \partial_b(\partial_c(h_\ell w_\ell)) \\ \sqsubseteq \partial_b(\partial_c(h_{\ell'})) \sqsubseteq \partial_b(\partial_{\mathbb{R}_B}(h_{\ell'})) \sqsubset \partial_b(\partial_{\mathbb{R}_B}(h_{\ell'}) w'_{\ell'} b) = z_{\ell'} \ , \end{aligned}$$

  where the second inequality holds because $h_\ell w_\ell \sqsubseteq h_{\ell'}$ since $\ell \leq \ell' - 1$, and the third inequality holds because $c \in B$ thus $\mathrm{dom}(c) \subseteq \mathbb{R}_B$ hence property (2) applies. Moreover the last inequality is strict because there is at least one more b in $\partial_b(\partial_{\mathbb{R}_B}(h_{\ell'}) w'_{\ell'} b)$ than in $\partial_b(\partial_{\mathbb{R}_B}(h_{\ell'}))$. We get a contradiction because by hypothesis there is no useless repetition in σ, however, denoting $x = z_\ell$ and y such that $xy = z_{\ell'}$, the pair (x, y) is a useless $\mathbb{T}^{(\ell)}$-repetition in σ: by hypothesis the strategic $\mathbb{T}^{(\ell)}$-states of z_ℓ and $z_{\ell'}$ are equal and both x and xy are $\mathbb{T}^{(\ell)}$-locks, moreover y is not empty because $z_\ell \sqsubset z_{\ell'}$ and finally $\mathrm{dom}(y) \subseteq \mathrm{dom}(u_{1+\ell k} \cdots u_{k+\ell' k}) \subseteq \mathbb{R}_B \subseteq \mathbb{T}^{(\ell)}$. Thus (x, y) is a useless repetition in σ.

- Second, for every $0 \leq \ell < \frac{N}{k}$, all plays in $\sigma^{(\ell)} = \pi(\sigma, z_\ell, \mathbb{T}^{(\ell)} \setminus \mathrm{dom}(b_\ell))$ have length $\leq m = \max_{B' \prec B} f(G_{B'})$. Let $z_\ell u'$ be a σ-play such that $\mathrm{dom}(u') \subseteq (\mathbb{T}^{(\ell)} \setminus \mathrm{dom}(b_\ell))$. Then $\mathrm{Alph}(u') \subseteq B_\ell$. Since $\preceq$ is monotonic with respect to inclusion, $\mathrm{Alph}(u') \preceq B_\ell \prec B \preceq \mathrm{Alph}(u)$. Thus by induction hypothesis, $|u'| \leq f\left(G_{\mathrm{Alph}(u')}\right) \leq m$.

According to the second property, there are at most $2^{2^{m|A||\mathbb{P}|}}$ different residuals appearing in the sequence $(\sigma^{(\ell)})_{0 \leq \ell < \frac{N}{k}}$. Thus the sequence $(\mathrm{strate}_\ell)_{0 \leq \ell < \frac{N}{k}}$ takes at most $K = |B| \cdot |Q|^{|\mathbb{P}|} \cdot 2^{2^{m|A||\mathbb{P}|}}$ different values. And according to the first property, all these states are different thus $N \leq k \cdot K$.

The inequality $N \leq k \cdot K$ has been established under the assumption that D_B is connected. The general case reduces to this case: let C be a connected component of D_B and for $1 \leq i \leq N$ let v_i be the projection of u_i on C. Then $\forall 1 \leq i \leq N, \mathrm{Alph}(v_i) = C$ and there exists u'_0 such that $u = u'_0 v_1 v_2 \dots v_N u_{N+1}$ thus $N \leq k \cdot K$.

Let us reformulate the inequality $N \leq k \cdot K$ as a property of an undirected complete graph with edges colored by 2^A. Let $u = a_1 a_2 \cdots a_{|u|}$ the factorization of u into its letters. Let J_u be the complete graph with vertices $1, \dots, |u|$ and the label of the edge $\{i, j\}$ with $i < j$ is the set of letters $\{a_i, \dots, a_j\}$. Then every monochromatic clique of J_u has size $\leq k \cdot K$. Thus, according to Ramsey theorem, $|u| \leq R_{\mathbb{T}}(k \cdot K) = R_{\mathbb{T}}((k' + |\mathbb{P}|) \cdot K)$, which completes the inductive step.

As a consequence, winning strategies in G can be looked for in the finite family of strategies all of whose plays have length $\leq f(G)$ with $f(G)$ computable. As a consequence, the synthesis problem can be solved by enumerating all these strategies and testing whether any of them is winning. For testing whether a strategy of finite duration is winning the algorithm simply checks that the global state of all the maximal plays is final. ◀

5.3 New examples of decidable games

The three classes of games whose decidability is already known are decomposable (cf Lemmas 14, 16 and 18). In this section we give some new examples of decidable games.

▶ **Lemma 22.** *Four players games are structurally decomposable.*

Although our techniques do not seem to provide an algorithm for solving games with five processes, they can address a subclass.

▶ **Lemma 23.** *Let G be a distributed game with five processes. Assume that the number of actions that a process can successively play in a row without synchronizing simultaneously with two other processes is bounded. Then G is process decomposable.*

Another decidable example is the class of *majority games*:

▶ **Lemma 24** (Majority games)**.** *Assume that every non-local action synchronizes a majority of the processes i.e. for every action a, $|\mathrm{dom}(a)| = 1$ or $|\mathrm{dom}(a)| \geq |\mathbb{P} \setminus \mathrm{dom}(a)|$. Then the game is structurally decomposable.*

The class of structurally decomposable games is stable under projection and merge.

▶ **Definition 25** (Projecting games)**.** Let G be a game with processes $\mathbb{P}$ and alphabet $(A_p)_{p \in \mathbb{P}}$. Let $\mathbb{P}' \subseteq \mathbb{P}$ a subset of the processes. The projection of G on $\mathbb{P}'$ is the game G' with processes $\mathbb{P}'$ and alphabet $A' = \{a \in A \mid \mathrm{dom}(a) \cap \mathbb{P}' \neq \emptyset\}$ partitioned in $(A' \cap A_p)_{p \in \mathbb{P}'}$. The states of a process $p \in \mathbb{P}'$ are the same in G and G', every transition $\delta \in \{a\} \times \Pi_{p \in \mathrm{dom}(a)} Q_p \times Q_p$ of G on a letter $a \in A'$ is projected to $\{a\} \times \Pi_{p \in \mathrm{dom}(a) \cap \mathbb{P}'} Q_p \times Q_p$, and every transition on a letter $a \notin A'$ is simply deleted.

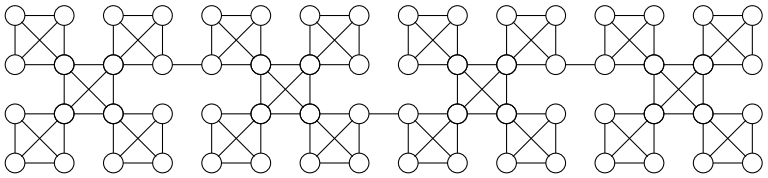

Figure 2 A decidable process architecture.

The following result combines two structurally decomposable games into one.

▶ **Lemma 26** (Merging games). *Let G be a game, and $\mathbb{P}_0, \mathbb{P}_1 \subseteq \mathbb{P}$ two set of processes such that $\mathbb{P} = \mathbb{P}_0 \cup \mathbb{P}_1$ and $\mathbb{P}_0 \cap \mathbb{P}_1 \neq \emptyset$ and for every action $a \in A$,*

$$(\mathrm{dom}(a) \cap \mathbb{P}_0 \neq \emptyset) \wedge (\mathrm{dom}(a) \cap \mathbb{P}_1 \neq \emptyset) \implies (\mathbb{P}_0 \cap \mathbb{P}_1 \subseteq \mathrm{dom}(a)) \ .$$

If both projections of G on $(\mathbb{P}_0 \setminus \mathbb{P}_1)$ and $(\mathbb{P}_1 \setminus \mathbb{P}_0)$ are structurally decomposable then G is structurally decomposable.

The merge operation can combine two structurally decomposable games in order to create a new one. For example all acyclic games can be obtained this way, since 3-player games are structurally decomposable and every tree with more than three nodes can be obtained by merging two strictly smaller subtrees. This technique can go beyond acyclic games, by merging together 4-player games and majority games. The graph of processes is an undirected graph with nodes $\mathbb{P}$ and there is an edge between p and q whenever both p and q both belong to the domain of one of the actions. Then all the games whose graph of processes is contained in the one depicted on Fig. 2 are structurally decomposable.

6 Conclusion

We considered the DISTRIBUTED SYNTHESIS PROBLEM, which aims at controlling asynchronous automata using automatically synthesized controllers with causal memory. We presented a theorem that unifies several known decidability results and provide new ones.

The decidability of this problem is, to the best of our knowledge, stil opened, even in the simple case where the graph of processes is a ring of five processes where each process can interact only with both its neighbors.

Another intriguing open problem is the case of *weakly k-connectedly communicating* plants. In such a plant, whenever two processes play both k times in a row without hearing from each other, they will never hear from each other anymore. It is not known whether the MSO theory of the corresponding event structures is decidable or not [6], neither do we know how to use techniques of this paper to solve this class of games.

Acknowledgements. We thank Blaise Genest, Anca Muscholl, Igor Walukiewicz, Paul Gastin and Marc Zeitoun for interesting discussions on the topic. Moreover we thank one of the reviewers of a previous version, who spotted several mistakes and did provide very useful comments which led to several improvements in the presentation of the results. We also thank Engel Lefaucheux for spotting a missing hypothesis in Lemma 26.

References

1 Volker Diekert and Grzegorz Rozenberg. *The Book of Traces*. World Scientific, 1995. URL: `https://books.google.co.uk/books?id=vNFLOE2pjuAC`.

2 Bernd Finkbeiner and Sven Schewe. Uniform distributed synthesis. In *Logic in Computer Science, 2005. LICS 2005. Proceedings. 20th Annual IEEE Symposium on*, pages 321–330. IEEE, 2005.

3 Paul Gastin, Benjamin Lerman, and Marc Zeitoun. Distributed games with causal memory are decidable for series-parallel systems. In Kamal Lodaya and Meena Mahajan, editors, *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science, 24th International Conference, Chennai, India, December 16-18, 2004, Proceedings*, volume 3328 of *Lecture Notes in Computer Science*, pages 275–286. Springer, 2004. doi:10.1007/978-3-540-30538-5_23.

4 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.

5 Hugo Gimbert. A class of zielonka automata with a decidable controller synthesis problem. *CoRR*, abs/1601.05176, 2016. arXiv:1601.05176.

6 P. Madhusudan, P. S. Thiagarajan, and Shaofa Yang. The MSO theory of connectedly communicating processes. In Ramaswamy Ramanujam and Sandeep Sen, editors, *FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science, 25th International Conference, Hyderabad, India, December 15-18, 2005, Proceedings*, volume 3821 of *Lecture Notes in Computer Science*, pages 201–212. Springer, 2005. doi:10.1007/11590156_16.

7 Anca Muscholl. Automated synthesis of distributed controllers. In Magnús M. Halldórsson, Kazuo Iwama, Naoki Kobayashi, and Bettina Speckmann, editors, *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, volume 9135 of *Lecture Notes in Computer Science*, pages 11–27. Springer, 2015. doi:10.1007/978-3-662-47666-6_2.

8 Anca Muscholl and Igor Walukiewicz. Distributed synthesis for acyclic architectures. In Venkatesh Raman and S. P. Suresh, editors, *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, volume 29 of *LIPIcs*, pages 639–651. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.639.

9 Anca Muscholl, Igor Walukiewicz, and Marc Zeitoun. A look at the control of asynchronous automata. In M. Mukund K. Lodaya and eds. N. Kumar, editors, *Perspectives in Concurrency Theory*. Universities Press, CRC Press, 2009.

10 Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *Foundations of Computer Science, 1990. Proceedings., 31st Annual Symposium on*, pages 746–757. IEEE, 1990.

11 Peter JG Ramadge and W Murray Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

12 Wieslaw Zielonka. Notes on finite asynchronous automata. *ITA*, 21(2):99–135, 1987.