Fair Synthesis for Asynchronous Distributed Systems

PAUL GASTIN, LSV, ENS Cachan, INRIA, CNRS
NATHALIE SZNAJDER, LIP6/MoVe, Université Pierre & Marie Curie, CNRS

We study the synthesis problem in an asynchronous distributed setting: a finite set of processes interact locally with an uncontrollable environment and communicate with each other by sending signals – actions controlled by a sender process and that are immediately received by the target process. The fair synthesis problem is to come up with a local strategy for each process such that the resulting fair behaviors of the system meet a given specification. We consider external specifications satisfying some natural closure properties related to the architecture. We present this new setting for studying the fair synthesis problem for distributed systems, and give decidability results for the subclass of networks where communications happen through a strongly connected graph. We claim that this framework for distributed synthesis is natural, convenient and avoids most of the usual sources of undecidability for the synthesis problem. Hence, it may open the way to a decidable theory of distributed synthesis.

Categories and Subject Descriptors: B.6.3 [**Logic Design**]: Design Aids—*Automatic Synthesis*; D.2.4 [**Software Engineering**]: Software/Program Verification—*Formal Methods*; B.4.3 [**Input/Output and Data Communication**]: Interconnections (Subsystems—*Asynchronous / Synchronous Operations*; F.1.2 [**Computation by Abstract Devices**]: Modes of Computation—*Parallelism and Concurrency*; F.4.1 [**Mathematical Logic and Formal Languages**]: Mathematical Logic—*Computability Theory, temporal logic*

General Terms: Algorithms,Theory, Verification

Additional Key Words and Phrases: Church's problem, Distributed Systems, Synthesis, Semi-commutations

ACM Reference Format:
Gastin, P. and Sznajder, N. 2012. Fair Synthesis for Asynchronous Distributed Systems. ACM Trans. Comput. Logic V, N, Article A (January YYYY), 31 pages.
DOI = 10.1145/0000000.0000000 http://doi.acm.org/10.1145/0000000.0000000

1. INTRODUCTION

Synthesis (also known as Church's problem) is an essential problem in computer science, especially in the context of formal methods for the design of systems. It consists in automatically deriving a system from its specification, hence allowing to produce a program that is certified to be correct, without any debugging process. Moreover, when the synthesis algorithm answers that there is no program for this specification, it is known at an early stage of the development process that the specification is unrealizable, and thus likely to be erroneous.

The initial problem considered by Church [1963] concerned reactive centralized systems interacting with an uncontrollable environment whose behaviors were described

Author's addresses: P. Gastin, LSV, ENS Cachan, INRIA, CNRS, Cachan, France `Paul.Gastin@lsv.ens-cachan.fr`; N. Sznajder, LIP6/MoVe, Université Pierre & Marie Curie, CNRS, Paris, France `Nathalie.Sznajder@lip6.fr`

Work supported by FP7 Quasimodo, ANR-06-SETI-003 DOTS, ARCUS Île de France-Inde.
Part of this research was conducted while the second author was at LSV, ENS de Cachan.

by a specification in monadic second order logic (MSO). Its decidability was established by Büchi and Landweber in [1969] with a game theoretic approach. A simpler solution has been brought in [Rabin 1972], where it is advocated that, even if the specification is linear-time, the need to consider all possible behaviors of an uncontrollable environment yields a branching-time behavior, which is best described by a tree and thus amenable to *tree* automata. This problem is now quite well understood, and current work aims at defining efficient algorithms towards implementations (see the tools Lily [Jobstmann and Bloem 2006], Acacia [Filiot et al. 2009] and Unbeast [Ehlers 2011] for synthesis from LTL specifications, which build on the efficient technique presented in [Kupferman and Vardi 2005]).

Different extensions of the problem have been studied. Among them, an important line of research is the synthesis for distributed systems, i.e., systems consisting of several communicating processes cooperating against an uncontrollable environment in order to satisfy the specification. This case is much more involved, and undecidable in general [Pnueli and Rosner 1990]. An important difficulty inherent to the distributed setting is the fact that each process has only a *partial* view of the global system and decisions about its behavior must be taken based solely on this local knowledge.

When dealing with distributed systems, an important parameter to take into account is whether the semantics is *synchronous* (a global clock governs the whole system, and at each tick of this clock, all the processes make an action simultaneously), or *asynchronous* (each component evolves at its own speed, with synchronization mechanisms for communication). In the synchronous semantics, undecidability is quickly reached, even if a few decidable subclasses have been identified. A bit surprisingly, more hope is permitted in the asynchronous case, in which the problem stays decidable for wider classes of architectures (see the paragraph on the related work for more details).

Contributions

We consider the synthesis problem for asynchronous distributed systems, and define a model yielding decidability of this problem when the communication graph of the system is strongly connected. It opens the way to a general decidable theory of synthesis for asynchronous distributed systems.

More precisely, we study systems which communicate via signals. In fact, it is often considered in distributed asynchronous systems that the different processes synchronize on common actions (rendezvous mechanism). However, this means that two processes must agree to execute a shared action; for synthesis, it implies that the processes must take the same decision, whereas they may have a completely different view of the global state of the system. Signals are a more convenient and realistic synchronization mechanism: it can be seen as an asymmetric rendezvous, where only one of the processes is able to trigger the signal, the other process receives it without executing any action.

Second, we restrict the specifications to *external* ones: we say that a specification is external if it only relates inputs from and outputs to the environment, without any constraint on the internal communication actions. Indeed, as we have already advocated in [Gastin et al. 2009], we believe that external specifications are more natural when dealing with distributed systems: the communication between the processes can be restricted by the architecture (the communication graph), but not by the specification. When describing expected behaviors of such a system, one should only be concerned with the visible external behaviors, and let the processes communicate freely in order to achieve the specification. Moreover, total specifications can be used to break communication links that exist in the architecture, and thus quickly yields undecidability results. For these two reasons, we address the synthesis problem with external specifications.

While writing a specification for a distributed system, one must be cautious and comply with the nature of the architecture. In asynchronous distributed systems, executions can be seen as *partial orders of actions*, where two actions occurring on different processes are ordered if there is some causality relation between them. Hence, the specification can only impose an order between two events on different processes if there is a causality relating them. To rule out unrealistic constraints, we forbid some unrealizable causalities between processes. This leads us to express specifications as *semi-traces*.

Finally, we introduce fairness conditions in our model and require that the specifications be met only by the fair executions of the system.

Hence, the inputs of the problem addressed in this article are: an architecture represented by a communication graph, and a specification given by a semi-trace-closed language relating input and output actions. The synthesis problem then asks for a local program for *each process* such that any fair run of the global system following these programs belongs to the specification language, or the answer that no such programs exist. We obtain decidability of the fair synthesis problem for the whole class of systems whose communication graph is strongly connected. This is a major improvement, in particular with respect to the synchronous case where the problem is in general undecidable for such architectures. We believe that this model will yield decidability for many more classes of architectures.

As an additional yet orthogonal contribution, we explore some useful properties on languages closed by semi-commutations: we prove that it can be decided in polynomial space whether a given ω-regular language is closed under semi-commutation, and we define a natural logic (a fragment of the logic MSO) in which closure under semi-commutation is guaranteed. Similar closure problems have been studied for partial commutation relations such as Mazurkiewicz traces, see [Muscholl 1996; Diekert et al. 1995; Peled et al. 1998].

The article is organized as follows: Section 2 defines some notions that will be used throughout the article. Section 3 presents in details the model we consider and Section 4 describes the specifications that will be used. Finally, the decidability results are presented in Section 5.

A preliminary version of this work appeared in [Chatain et al. 2009].

Related work

We relate our contribution to existing work by presenting the different results obtained in two categories: results obtained for synchronous systems, and then results obtained for asynchronous systems.

Synthesis of distributed synchronous systems has been initiated by Pnueli and Rosner [1990] who proved that the problem was undecidable for LTL specifications. At the same time, they have identified the subclass of pipeline architectures – architectures in which processes communicate in a chain – for which the problem is decidable for LTL external specifications. The following contributions on the topic considered several variants for the specification. In addition to the specification language itself (given by a formula in linear or branching time logic, with different expressive power), another concern is the set of channels (input, output or dedicated to internal communication ones) the specification is allowed to talk about. Three types of specifications have thus been given attention to: total, external and local.

— Total specifications may refer to any channel. They are the most general.

— External specifications are only allowed to relate input and output values and let the internal communication channels unconstrained.
— Local specifications are Boolean combinations of specifications relating only input and output values of one process.

In [Kupferman and Vardi 2001], an automata-theoretic approach to solving synthesis for pipe-line architectures (and some variations) is given, and they show that this case stays decidable for total specifications given as CTL^* formulas. To fill the gap between the undecidable architecture of [Pnueli and Rosner 1990] and the specific decidable case where the input architecture is a variant of the pipe-line, a uniform (un)decidability criterion is given in [Finkbeiner and Schewe 2005]: they introduce the notion of information fork in the architecture as a necessary and sufficient condition for undecidability of the problem. In [Gastin et al. 2009] it is shown that this criterion is only valid when the specification given is total: if we restrict to external specifications, then the cases where the problem is decidable can be expanded: for instance, the synthesis problem becomes decidable for so-called uniformly well-connected architectures. Madhusudan and Thiagarajan [2001] have considered an extension of the synthesis problem (the controller synthesis problem) with local specifications. The problem remains undecidable in most cases: it is decidable if and only if it is a sub-architecture of a pipe-line with inputs at both endpoints.

Early work on synthesis of asynchronous systems concerned centralized systems [Pnueli and Rosner 1989], and fairness conditions were included in [Anuchitankul and Manna 1994; Vardi 1995]. In [Madhusudan and Thiagarajan 2002], the (controller) synthesis problem has been studied for distributed asynchronous systems, with synchronization on common actions. Communications with the environment are local to each process, while communication between processes are shared actions (rendezvous) between two processes. As in our setting, runs can naturally be represented by partial orders of actions, more specifically by Mazurkiewicz traces [Mazurkiewicz 1977]. When the specification is trace-closed, (i.e., does not discriminate between two linearizations of the same trace) the problem is decidable, but only if the processes have a very restricted type of local memory [Madhusudan and Thiagarajan 2002]: strategies of processes depend only on the *number* of inputs received and the value of the last input. Considering external specifications, as we do in this work, break the undecidability arguments used there.

The synthesis of distributed systems in the general case of μ-calculus specifications was studied in [Finkbeiner and Schewe 2006], where the processes are allowed to communicate through shared variables. Decidability is obtained only when there is a single process to control. Indeed, contrary to our setting, they did not impose any closure property on the specification language which may accept some linearizations of a (partially ordered) behavior and reject other linearizations of the same behavior.

In [Gastin et al. 2004; Madhusudan et al. 2005; Muscholl et al. 2009], it is assumed that, each time they synchronize on a common action, two processes exchange all the information they have accumulated on the current behavior. This is different from our setting (and the other ones discussed above) where processes are assumed to have only a local view of the behaviors.

Synthesis of asynchronous distributed systems have also been studied in [Stefănescu et al. 2003; Baudru 2009] in the restricted case of *closed systems*, following the line of [Emerson and Clarke 1982; Manna and Wolper 1984].

Since dealing with distributed systems implies dealing with imperfect information, all these works are related to the synthesis of centralized systems with imperfect information as studied in the branching time setting in[Kupferman and Vardi 1999; Kupferman and Vardi 2000; Arnold et al. 2003].

Moreover, for all the variants of the synthesis problem for open systems, the game setting is as useful as in the centralized case. A specialized version of multiplayer games named distributed games has been defined in [Mohalik and Walukiewicz 2003] and allows to reason in a uniform way about synthesis of distributed systems in all these different frameworks.

2. PRELIMINARIES

An alphabet Σ is a finite set of symbols. A sequence of elements of Σ is a *word*. If $w = w_0 w_1 \ldots w_{n-1}$ is a word, n is the *length* of w, noted $|w| = n$. If $w = w_0 w_1 w_2 \cdots$ is infinite, $|w| = \omega$. The word of length 0 is the *empty word* noted ε. We denote the set of finite words over Σ by Σ^*, the set of non empty finite words by Σ^+ and the set of infinite words by Σ^ω. We use $\Sigma^\infty = \Sigma^* \uplus \Sigma^\omega$ to denote the set of finite and infinite words over Σ. For $w = w_0 w_1 \cdots w_{n-1} \in \Sigma^*$, $w' = w'_0 w'_1 \cdots \in \Sigma^\infty$, the *concatenation* of w and w' noted $w \cdot w'$ (or simply ww') is the word $w_0 w_1 \cdots w_{n-1} w'_0 w'_1 \cdots \in \Sigma^\infty$. The set of finite prefixes of w is defined by $\mathrm{Pref}(w) = \{u \in \Sigma^* \mid \exists v, uv = w\}$. The *prefix* relation over Σ^∞ is a partial order relation that is denoted $\leq$: $w' \leq w$ if $w' \in \mathrm{Pref}(w)$.

The prefix of length $i \leq |w|$ of a word $w \in \Sigma^\infty$ is noted $w[i]$, with the convention that $w[i] = \varepsilon$ if $i \leq 0$ and $w[i] = w$ if $i \geq |w|$. If u is a prefix of v, the word $u^{-1}v$ is such that $uu^{-1}v = v$ (i.e., it is the word obtained from v by deleting the prefix u).

For $w \in \Sigma^\infty$, we denote by $\mathrm{alphinf}(w)$ the set of letters from Σ occurring infinitely often in w and by $\mathrm{alph}(w)$ the set of letters from Σ occurring at least once in w.

3. MODEL

An architecture defines how a set of processes may communicate with each other and with an (uncontrollable) external environment. An important parameter of the problem is the type of communications allowed between processes. We are interested in asynchronous distributed systems, hence it would be natural to use unbounded fifo channels. However, this leads to infinite state systems, making decidability results more uncertain to obtain.

A finite model can be obtained by using shared variables: processes can write on variables that can be read by other processes. But in an asynchronous system, communication is difficult to achieve with shared variables. Assume that process p wants to transmit to process q a sequence $m_1, m_2, \ldots$ of messages. First, p writes m_1 to some shared variable x. But since processes evolve asynchronously, p does not know when m_1 will be read by q. Hence, some acknowledgement is required from q to p before p may write m_2 to x. Depending on the architecture, this may not be possible. In any cases, it makes synthesis of distributed programs satisfying a given specification harder.

Hence, we will use point to point communication by signals in the vein of [Lynch and Tuttle 1989]. Sending a signal is an action but receiving a signal is not. Instead, all signals sent to some process q are automatically added to its local history, *without requiring actions from* q. The system is still asynchronous, meaning that processes evolve at different speeds. We are interested in synthesizing *local* programs, also called strategies. By *local* we mean that to decide which action it should execute next, a process q only knows its current local history, which automatically includes all signals sent to q in addition to the signals sent by q.

Architectures and runs. Formally, an architecture is defined by a tuple $\mathcal{A} = (\mathrm{Proc}, E, (\mathrm{In}_p)_{p \in \mathrm{Proc}}, (\mathrm{Out}_p)_{p \in \mathrm{Proc}})$ where (Proc, E) is the directed communication graph whose nodes are processes and there is an edge $(p, q) \in E$ if process p may send signals to process q. See for example the architecture represented in Figure 1, where $\mathrm{Proc} = \{1, 2, 3\}$ and where process 1 can send signals to process 2 and process 2 to process 3. For each process $p \in \mathrm{Proc}$, the sets In_p and Out_p define input and output signals that p may receive from or send to the environment. We assume that all these sets are pairwise disjoint. We let $\mathrm{In} = \bigcup_{p\in\mathrm{Proc}} \mathrm{In}_p$ and $\mathrm{Out} = \bigcup_{p\in\mathrm{Proc}} \mathrm{Out}_p$ be the sets of input and output signals of the whole system. Let also $\Gamma = \mathrm{In} \cup \mathrm{Out}$. In order to implement a specification, the processes may choose for each communication link $(p, q) \in E$ a (finite or infinite) set $\Sigma_{p,q}$ of signals that p could send to q. Again, we assume that these sets are pairwise disjoint and disjoint from Γ. The complete alphabet (of signals) is then $\Sigma = \Gamma \cup \bigcup_{(p,q)\in E} \Sigma_{p,q}$. The actions in Γ are called *external* signals whereas the actions in $\Sigma \setminus \Gamma$ are called *internal* signals. For each $a \in \Sigma$ we let $\mathrm{process}(a)$ be the set of processes taking part in the execution of a: $\mathrm{process}(a) = \{p\}$ if $a \in \mathrm{In}_p \cup \mathrm{Out}_p$ and $\mathrm{process}(a) = \{p, q\}$ if $a \in \Sigma_{p,q}$. For $p \in \mathrm{Proc}$, we denote by $\Sigma_p = \{a \in \Sigma \mid p \in \mathrm{process}(a)\}$ the set of actions visible to process p and by $\Sigma_{p,C} = \mathrm{Out}_p \cup \bigcup_{q|(p,q)\in E} \Sigma_{p,q}$ the set of actions *controlled* by process p.

A (concrete) run $w \in \Sigma^\infty$ of $\mathcal{A}$ is then a (finite or infinite) word over Σ.

Strategies. We aim at synthesizing distributed strategies, with *local* memory. A strategy is a program that controls the behavior of the system, in interaction with uncontrollable inputs from an environment. It either proposes the next value to output, or decides to wait until a new event occurs. A *local strategy* for process p only depends on its visible actions, i.e, actions in Σ_p. Formally, let $\pi_p : \Sigma^* \to \Sigma_p^*$ be the projection on Σ_p. We define

Definition 3.1. A *local strategy* for process p is a partial function $f_p : (\Sigma_p)^* \to \Sigma_{p,C}$. We extend it to words over Σ in the natural way: for $w \in \Sigma^*$, $\hat{f}_p(w) = f_p(\pi_p(w))$.

By a slight abuse of notation, in this article we will simply write f_p for $\hat{f}_p$.

A *distributed strategy* for the system, is a tuple $F = (f_p)_{p\in\mathrm{Proc}}$ such that $f_p : \Sigma^* \to \Sigma_{p,C}$ is a local strategy for process p. By abuse of notations, we also use F to denote the induced mapping, defined by $F : \Sigma^* \to 2^{\Sigma_C}$ such that $F(w) = \{f_p(w) \mid p \in \mathrm{Proc}, f_p(w) \text{ is defined}\}$, for $w \in \Sigma^*$.

Runs compatible with a strategy. Let us fix a distributed strategy $F = (f_p)$. We say that a run $w = w_0w_1 \cdots \in \Sigma^\infty$ is an *F-run* (or is compatible with strategy F, or is F-compatible) if all controllable events occur according to F, i.e., for all $p \in \mathrm{Proc}$, for all index $0 \le i < |w|$, if $w_i \in \Sigma_{p,C}$, we have $w_i = f_p(w[i])$.

A finite run $w \in \Sigma^*$ is *F-maximal* if $F(w) = \emptyset$.

Example 3.2. Consider the architecture of Figure 1, restricted to the two leftmost processes, where the set of external signals is $\mathrm{In}_1 = \{\mathrm{req}_1\}$, $\mathrm{In}_2 = \{\mathrm{req}_2\}$, $\mathrm{Out}_1 = \{\mathrm{grant}_1\}$ and $\mathrm{Out}_2 = \{\mathrm{grant}_2\}$. We define the strategies $f_1(w) = \mathrm{grant}_1$ and $f_2(w) = \mathrm{grant}_2$ for any $w \in \Sigma^*$. Then, the runs $w \in \{\mathrm{req}_1, \mathrm{req}_2\}^\omega$ are indeed F-runs, but in which the system has no opportunity to grant the requests sent continuously by the environment. Moreover, runs $w \in \{\mathrm{req}_1, \mathrm{req}_2, \mathrm{grant}_1\}^\omega$ are also F-runs, in which only Process 1 could emit signals, while Process 2 was never scheduled. Without fairness assumptions, it is impossible to find a strategy for the system such that all compati-

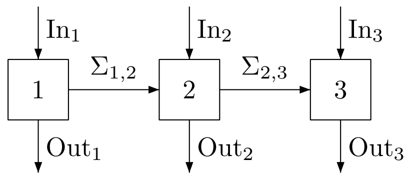


Fig. 1. An architecture

ble runs satisfy a simple request-response specification for each process. However, if we assume "fair" schedulers for our systems, we can circumvent these pathological behaviors.

Fairness. Thus, we will restrict our attention to *fair* runs of the system, which is defined below with respect to some partition of the set of all controllable actions.

Definition 3.3 (*Fair run*). Given a partition $\mathcal{P}$ of Σ_C and a distributed strategy F, a run $w \in \Sigma^\infty$ is $(\mathcal{P}, F)$-*fair* if, for all $C \in \mathcal{P}$, for all prefix v of w, if $F(v') \cap C \neq \emptyset$ for all $v \leq v' \leq w$, then some output from C will be emitted, i.e., $\mathrm{alph}(v^{-1}w) \cap C \neq \emptyset$.

This definition is equivalent to saying that, in a fair run, if $F(v') \cap \Sigma' \neq \emptyset$ for all $v \leq v' \leq w$, then an infinite number of outputs from C will be emitted, i.e., $\mathrm{alphinf}(w') \cap C \neq \emptyset$.

Remark 3.4. The coarsest partition of Σ_C will then ensure that, if at some point the system as a whole is continuously willing to output signals, it will eventually do so. However, Example 3.2 has shown that this may not be sufficient : with this definition, it may happen that a run is fair, where one of the processes is continuously enabled, but never scheduled, if other processes are allowed to output infinitely many signals. Then, the coarsest partition of Σ_C we will consider is $\{\Sigma_{p,C} \mid p \in \mathrm{Proc}\}$, i.e., a "per process" fairness notion.

Remark 3.5. A *finite* run is $(\mathcal{P}, F)$-fair if and only if it is F-maximal: let $w \in \Sigma^*$ be a finite F-compatible run. If $F(w) \neq \emptyset$ (hence w is not F-maximal), then one can find C element of the partition such that $F(w) \cap C \neq \emptyset$. However, since w is finite, w cannot be the prefix of a word $w'a$ with $a \in C$, and w is not $(\mathcal{P}, F)$-fair. Conversely, if w is F-maximal, then $F(w) = \emptyset$ and for all $C \in \mathcal{P}$, $F(w) \cap C = \emptyset$. The run is then $(\mathcal{P}, F)$-fair.

Specifications. The specifications we consider only constrain *external* actions from Γ, i.e., actions that reflect communications with the environment. We want the processes to collaborate freely in order to achieve the specification, hence we do not constrain internal signals. Specifications will describe *observable* runs, defined as follows.

Concrete and observable runs. For a (concrete) run $w \in \Sigma^\infty$, we define its observable part by $\pi_\Gamma(w)$ where all events from $\Sigma \setminus \Gamma$ have been removed.

We now state precisely the synthesis problem we address in this article.

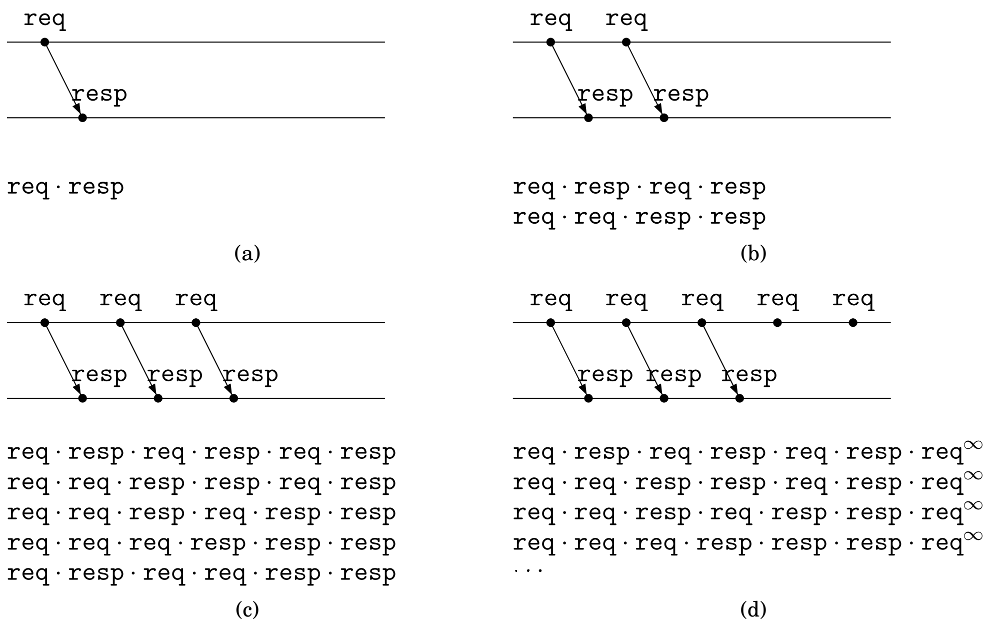
Fair synthesis of asynchronous distributed systems:. Given an architecture $\mathcal{A} = (\mathrm{Proc}, E, (\mathrm{In}_p)_{p\in\mathrm{Proc}}, (\mathrm{Out}_p)_{p\in\mathrm{Proc}})$ and a specification $L \subseteq \Gamma^\infty$, consider the partition $\mathcal{P} = \{\Sigma_{p,C} \mid p \in \mathrm{Proc}\}$, and decide whether there exist internal signal sets $(\Sigma_{p,q})_{(p,q)\in E}$ and a distributed strategy F such that, for *every* $(\mathcal{P}, F)$-fair concrete F-run w, we have $\pi_\Gamma(w) \in L$. We then say that $((\Sigma_{p,q})_{(p,q)\in E}, F)$ is winning for $(\mathcal{A}, L)$.

4. SPECIFICATIONS FOR DISTRIBUTED SYNTHESIS

We explain now that not all specifications are *acceptable* in our framework, and describe the properties of the specifications we will restrict to.

4.1. Motivations

We consider here *asynchronous* systems, hence each process of the system evolves at its own speed. Then, when a specification requests an order between two events, it may have different meanings according to their relative positions. When describing the events occurring on a same process, the order requested by the specification may really mean *sequentiality* of events, as in classical temporal specifications on centralized systems. However, if an order between events on different processes is required, then it can only come from a *causality* relation, as no global clock can order these events.



$\texttt{req} \cdot \texttt{resp}$

(a)

$\texttt{req} \cdot \texttt{resp} \cdot \texttt{req} \cdot \texttt{resp}$
$\texttt{req} \cdot \texttt{req} \cdot \texttt{resp} \cdot \texttt{resp}$

(b)

$\texttt{req} \cdot \texttt{resp} \cdot \texttt{req} \cdot \texttt{resp} \cdot \texttt{req} \cdot \texttt{resp}$
$\texttt{req} \cdot \texttt{req} \cdot \texttt{resp} \cdot \texttt{resp} \cdot \texttt{req} \cdot \texttt{resp}$
$\texttt{req} \cdot \texttt{req} \cdot \texttt{resp} \cdot \texttt{req} \cdot \texttt{resp} \cdot \texttt{resp}$
$\texttt{req} \cdot \texttt{req} \cdot \texttt{req} \cdot \texttt{resp} \cdot \texttt{resp} \cdot \texttt{resp}$
$\texttt{req} \cdot \texttt{resp} \cdot \texttt{req} \cdot \texttt{req} \cdot \texttt{resp} \cdot \texttt{resp}$

(c)

$\texttt{req} \cdot \texttt{resp} \cdot \texttt{req} \cdot \texttt{resp} \cdot \texttt{req} \cdot \texttt{resp} \cdot \texttt{req}^\infty$
$\texttt{req} \cdot \texttt{req} \cdot \texttt{resp} \cdot \texttt{resp} \cdot \texttt{req} \cdot \texttt{resp} \cdot \texttt{req}^\infty$
$\texttt{req} \cdot \texttt{req} \cdot \texttt{resp} \cdot \texttt{req} \cdot \texttt{resp} \cdot \texttt{resp} \cdot \texttt{req}^\infty$
$\texttt{req} \cdot \texttt{req} \cdot \texttt{req} \cdot \texttt{resp} \cdot \texttt{resp} \cdot \texttt{resp} \cdot \texttt{req}^\infty$
$\dots$

(d)

Fig. 2. Example of specification

Hence, specifications can be seen as *partial orders* of events labeled by Γ, where events occurring on a same process are totally ordered, and events occurring on two different processes are only ordered when a causality relation exists between them.

Remark 4.1. Concrete runs of our systems can be seen as *Mazurkiewicz traces* [Mazurkiewicz 1977], i.e., they can be partitioned in equivalence classes according to a partial commutation relation. In that case, the synthesis problem is undecidable if we allow specification languages that are not trace-closed [Madhusudan and Thiagarajan 2002]. Here, specifications are *external* and refer only to *observable* runs (projections of concrete runs), which are not Mazurkiewicz traces, hence the above result does not apply. However, as we will see below, it is still relevant, and in fact crucial, to consider specification languages that are closed under some semi-commutation relation induced by the architecture.

Example 4.2. Consider an architecture with two processes, one receiving service requests from a client and the other sending a response: $\mathrm{In}_c = \{\texttt{request}\}$ and $\mathrm{Out}_s = \{\texttt{response}\}$. Figure 2 shows the different partial orders (and their associated linearizations) corresponding to a specification requesting the server to answer to up to three requests and then ignore following requests. Then, when several requests occur, we require that the same number of responses (up to three) is issued, and the order imposed means that a request triggers a response, but no response can occur spontaneously. Then, the specification accepts the word $\texttt{req} \cdot \texttt{resp} \cdot \texttt{req} \cdot \texttt{resp}$, and the word $\texttt{req} \cdot \texttt{req} \cdot \texttt{resp} \cdot \texttt{resp}$, in which a second request has been sent before the first response could be emitted.

Moreover, it would be meaningless for a specification to request a causality relation between any event on some process p and an *input* event on a different process q. Indeed, input events are uncontrollable, it is then unrealistic to expect them to occur *as a consequence* of any other event. Therefore, our specifications will not require a causal relationship between an event and an input on any another process.

Since runs of the systems are words, the specifications we consider will describe *linearizations* of the desired partial orders, with the restriction that the original partial orders contain no causality between an event and an *input* on another process. Formally, our specifications will be languages $L \subseteq \Gamma^\infty$ closed under semi-commutations [Clerbout and Latteux 1987]: if $uabv \in L$ with $b \in \mathrm{In}_q$ and $a \notin \Sigma_q$, then there is no causality from a to b, and $ubav \in L$.

4.2. Semi-commutations and SC-closed specifications

In the following, we will often identify a word $w = w_0 w_1 \cdots \in \Sigma^\infty$ with the *labeled total order* $(\mathrm{Pos}(w), \leq, w)$ where $\mathrm{Pos}(w) = \{i \in \mathbb{N} \mid 0 \leq i < |w|\}$, $\leq$ is the natural order over $\mathbb{N}$, and $w : \mathrm{Pos}(w) \to \Sigma$ is such that $w(i) = w_i$.

Given an alphabet Σ, a *semi-commutation* relation is an irreflexive binary relation $\mathrm{SC} \subseteq \Sigma \times \Sigma \setminus \{(a,a) \mid a \in \Sigma\}$. We denote by SD the semi-dependence relation given by the semi-commutation relation SC, i.e., $\mathrm{SD} = \Sigma \times \Sigma \setminus \mathrm{SC}$. We associate with SC a *rewriting relation* $\to_{\mathrm{SC}}$ that is defined by $uabv \to_{\mathrm{SC}} ubav$ if $u \in \Sigma^*$, $(a,b) \in \mathrm{SC}$, and $v \in \Sigma^\infty$. When w is finite, its semi-commutation closure (i.e., the set of words that can be derived from w by applying rewriting permutations) is given by the reflexive and transitive closure $\to_{\mathrm{SC}}^*$ of $\to_{\mathrm{SC}}$. The set $[w] = \{w' \in \Sigma^* \mid w \to_{\mathrm{SC}}^* w'\}$ is called a *semi-trace* [Hung and Knuth 1989; Ochmanski 1989]. However, when w is infinite, this is not enough. For instance, suppose that $(a,b) \in \mathrm{SC}$, and consider $(ab)^\omega$. Intuitively, we would like $(ba)^\omega$ to be in the semi-commutation closure of $(ab)^\omega$, but this cannot be obtained within a finite number of commutations with $\to_{\mathrm{SC}}^*$. On the other hand, allowing infinitely many commutations should be "controlled" in order to avoid obtaining b^ω as a limit of $(ab)^\omega \to_{\mathrm{SC}}^* b^n a^n (ab)^\omega$. To define the semi-commutation closure of an infinite word, we rely upon the *semi-dependence graph* associated with a word $w \in \Sigma^*$ as presented in [Diekert 1994], and extended to words in Σ^∞:

Definition 4.3. Let $w = (\mathrm{Pos}(w), \leq, w)$ be a (finite or infinite) word over Σ and $\mathrm{SD} \subseteq \Sigma^2$ a semi-dependence relation. The *semi-dependence graph* associated with w is the labeled acyclic graph

$$\mathrm{G}(w) = (\mathrm{Pos}(w), E_w, w)$$

where $E_w = \{(i,j) \in \mathrm{Pos}(w)^2 \mid i < j \text{ and } (w(i), w(j)) \in \mathrm{SD}\}$.

The semi-dependence graph associated with w only keeps the order between two events if this order won't change in the rewriting process. We can now define the semi-trace of w, as the set of *linearizations* of $\mathrm{G}(w)$. Formally, we give the following definition.

Definition 4.4. Let $w = (\mathrm{Pos}(w), \leq, w)$ and $w' = (\mathrm{Pos}(w'), \leq, w')$ be two words. Then $w \Rightarrow_{\mathrm{SC}} w'$ if there exists a bijection $\sigma : \mathrm{Pos}(w) \to \mathrm{Pos}(w')$ such that for all $i, j \in \mathrm{Pos}(w)$,

— $w'(\sigma(i)) = w(i)$,
— and $(i,j) \in E_w$ implies $\sigma(i) < \sigma(j)$.

We say that $w \Rightarrow_{\mathrm{SC}} w'$ *by* σ. We let $[w]_{\mathrm{SC}} = \{w' \in \Sigma^\infty \mid w \Rightarrow_{\mathrm{SC}} w'\}$.

Note that, the relation $\Rightarrow_{\mathrm{SC}}$ is reflexive ($w \Rightarrow_{\mathrm{SC}} w$ by the identity) and is transitive: if $w \Rightarrow_{\mathrm{SC}} w'$ by σ and $w' \Rightarrow_{\mathrm{SC}} w''$ by σ' then $w \Rightarrow_{\mathrm{SC}} w''$ by $\sigma' \circ \sigma$.

Remark 4.5. If $w \Rightarrow_{\mathrm{SC}} w'$, then $|w| = |w'|$. Moreover, the bijection σ is unique. Indeed, let $i \in \mathrm{Pos}(w)$ such that $w(i) = a$. We claim that if i is the position of the k-th occurrence of a in w, then $\sigma(i)$ is the position of the k-th occurrence of a in w'. Indeed, if $i < j$ and $w(i) = w(j)$, then $(i,j) \in E_w$, which implies that $\sigma(i) < \sigma(j)$. The claim follows.

Remark 4.6. Definition 4.4 is in fact equivalent to the *limit extension* of a relation given in [Peled et al. 1998, Definition 1]: $w \Rightarrow_{\mathrm{SC}} w'$ if and only if for all $u \leq w$, there exist $v, v' \in \Sigma^*$ such that $v' \leq w'$ and $uv \to_{\mathrm{SC}}^* v'$ and, for all $u' \leq w'$, there exist $v, v' \in \Sigma^*$ such that $v \leq w$ and $v \to_{\mathrm{SC}}^* u'v'$. Since this characterization is not used in this article, its proof is omitted.

We denote by $[L]_{\mathrm{SC}} = \bigcup_{w \in L} [w]_{\mathrm{SC}}$ the semi-commutation closure of a language L with respect to SC. We say that $L \subseteq \Sigma^\infty$ is SC-*closed* if $L = [L]_{\mathrm{SC}}$. When SC is clear from the context, we drop the subscript and simply write $[L]$.

We are now ready to define formally the specifications we restrict to for the distributed synthesis problem. We associate with the architecture $\mathcal{A} = (\mathrm{Proc}, E, (\mathrm{In}_p)_{p\in\mathrm{Proc}}, (\mathrm{Out}_p)_{p\in\mathrm{Proc}})$ the semi-commutation relation:

$$\mathrm{SC}_{\mathcal{A}} = \{(a,b) \mid b \in \mathrm{In}_p \text{ and } a \notin \Sigma_p\} = \bigcup_{p\in\mathrm{Proc}} (\Sigma \setminus \Sigma_p) \times \mathrm{In}_p.$$

The *semi-dependence* relation is defined by $\mathrm{SD}_{\mathcal{A}} = (\Sigma \times \Sigma) \setminus \mathrm{SC}_{\mathcal{A}}$. Observe that $\Sigma_p \times \Sigma_p \subseteq \mathrm{SD}_{\mathcal{A}}$ for all $p \in \mathrm{Proc}$, i.e., the set of actions relative to a process are pairwise dependent.

Definition 4.7. Let $\mathcal{A} = (\mathrm{Proc}, E, (\mathrm{In}_p)_{p\in\mathrm{Proc}}, (\mathrm{Out}_p)_{p\in\mathrm{Proc}})$ be an architecture. *SC-closed specifications* for $\mathcal{A}$ are languages $L \subseteq \Gamma^\infty$ closed under $\mathrm{SC}_{\mathcal{A}}$.

In the next two subsections, we elaborate on semi-commutations and SC-closed specifications. Since these results are independent from the rest of the article, the reader can safely go directly to Section 5 where the decidability results are given.

4.3. More on semi-commutations

An important question is to decide whether a given language is *closed* under SC. This closure problem for ω-regular languages has been studied for partial commutation relations such as Mazurkiewicz traces, see [Muscholl 1996; Diekert et al. 1995; Peled et al. 1998]. Other types of closure problems are studied in [Bouajjani et al. 2001; Cécé et al. 2008; Cano et al. 2011]. These works investigate classes $\mathcal{C}$ of regular languages (of finite words) such that for each language $L \in \mathcal{C}$ the closure $[L]$ is still in $\mathcal{C}$. In [Cano et al. 2011], the authors also give conditions on the semi-commutation (resp. partial commutation) relation and on the class $\mathcal{C}$ of regular languages ensuring that the closure of any language $L \in \mathcal{C}$ stays regular.

Here we show that one can decide whether an ω-regular language is closed under a given semi-commutation relation.

THEOREM 4.8. *Given an ω-regular language $L \subseteq \Sigma^\infty$ and a semi-commutation relation* SC*, we can decide whether L is* SC*-closed, i.e., whether $L = [L]_{\mathrm{SC}}$.*

The proof of this theorem follows the ideas presented in [Peled et al. 1998] about the closure-problem for partial commutations and other equivalence relations such as stuttering. We show that some of their results can be obtained even if the relation considered is not symmetric. Our proof relies on a characterization of $\Rightarrow_{\mathrm{SC}}$ using piecewise extension of $\to_{\mathrm{SC}}^*$: for $u, v \in \Sigma^\infty$, we write $u \to_{\mathrm{SC}}^\omega v$ if there are infinite factorizations $u = u_0u_1u_2\cdots$, $v = v_0v_1v_2\cdots$ with $u_i, v_i \in \Sigma^*$ and $u_i \to_{\mathrm{SC}}^* v_i$ for all $i \geq 0$. Note that, if $u, v \in \Sigma^*$ then $u \to_{\mathrm{SC}}^\omega v$ if and only if $u \to_{\mathrm{SC}}^* v$. Hence, this definition is really useful for infinite words in which case we may assume proper factorizations, i.e., $u_i, v_i \in \Sigma^+$ for all $i \geq 0$.

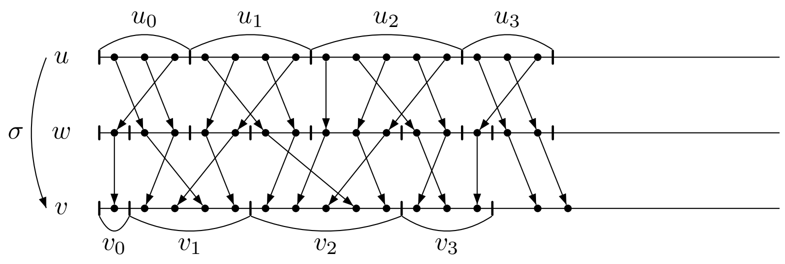


Fig. 3. Double Factorization

It is easy to see that $u \to^{\omega}_{\mathrm{SC}} v$ implies $u \Rightarrow_{\mathrm{SC}} v$. The converse is not true in general since the relation $\to^{\omega}_{\mathrm{SC}}$ is not necessarily transitive. Observe for instance that $abaabaaab\ldots \Rightarrow_{\mathrm{SC}} babababa\ldots$ while $abaabaaab\cdots \not\to^{\omega}_{\mathrm{SC}} babababa\ldots$. Still, it yields the following characterization.

LEMMA 4.9 (DOUBLE FACTORIZATION). *For all $u, v \in \Sigma^{\infty}$, we have $u \Rightarrow_{\mathrm{SC}} v$ if and only if there exists $w \in \Sigma^{\infty}$ such that $u \to^{\omega}_{\mathrm{SC}} w \to^{\omega}_{\mathrm{SC}} v$.*

PROOF. Since $\Rightarrow_{\mathrm{SC}}$ is transitive and $\to^{\omega}_{\mathrm{SC}}$ is contained in $\Rightarrow_{\mathrm{SC}}$, one direction is trivial. Conversely, assume that $u \Rightarrow_{\mathrm{SC}} v$ by σ. If $u, v \in \Sigma^*$ are finite then $u \to^*_{\mathrm{SC}} v$ and the result is trivial. Hence we assume in the following that $u, v \in \Sigma^{\omega}$.

Write $u = a_0a_1a_2\cdots$ and $v = b_0b_1b_2\cdots$ with $a_i, b_i \in \Sigma$ for all $i \geq 0$. We construct now inductively infinite factorizations of u and v, see Figure 3.

— Let $m_0 = 0$, $v_0 = b_0$ and $n_0 = \sigma^{-1}(0)$, $u_0 = a_0\cdots a_{n_0}$.
— For $i \geq 0$, let $m_{i+1} = 1 + \max(\sigma(\{0, \ldots, n_i\}))$, $v_{i+1} = b_{1+m_i}\cdots b_{m_{i+1}}$ and $n_{i+1} = \max(\sigma^{-1}(\{0, \ldots, m_{i+1}\}))$, $u_{i+1} = a_{1+n_i}\cdots a_{n_{i+1}}$.

Clearly, $m_{i+1} > m_i$ and $n_{i+1} > n_i$ for all $i \geq 0$. Hence we have defined infinite factorizations $u = u_0u_1u_2\cdots$ and $v = v_0v_1v_2\cdots$ with $u_i, v_i \in \Sigma^+$ for all $i \geq 0$.

Now, we define the word w and its double factorization, see Figure 3.

— Let $U_0 = \{0, \ldots, n_0\}$, $X_0 = \{n_0\}$ and $Y_0 = \{0, \ldots, n_0 - 1\}$.
— For $i \geq 0$, let $U_{i+1} = \{1 + n_i, \ldots, n_{i+1}\}$, $X_{i+1} = U_{i+1} \cap \sigma^{-1}(\{0, \ldots, m_{i+1}\}) = U_{i+1} \cap \sigma^{-1}(\{1 + m_i, \ldots, m_{i+1}\})$ and $Y_{i+1} = U_{i+1} \setminus X_{i+1}$.

Now, for all $i \geq 0$, we define x_i and y_i as the subwords of u corresponding to positions in X_i and Y_i respectively. Finally, we let $w = x_0y_0x_1y_1x_2y_2\cdots$.

CLAIM 4.10. *For all $i \geq 0$ we have $u_i \to^*_{\mathrm{SC}} x_iy_i$ and $y_ix_{i+1} \to^*_{\mathrm{SC}} v_{i+1}$.*

Indeed, u_i is the subword (factor) of u corresponding to positions in $U_i = X_i \uplus Y_i$. By definition, we have $\max(\sigma(X_i)) \leq m_i < \min(\sigma(Y_i))$. Since $u \Rightarrow_{\mathrm{SC}} v$ by σ, we deduce that for all $j \in X_i$ and $k \in Y_i$, either $j < k$ or $(a_k, a_j) \in \mathrm{SC}$. Therefore $u_i \to^*_{\mathrm{SC}} x_iy_i$.

Next, for all $i \geq 0$, the subword of u corresponding to positions in $Y_i \uplus X_{i+1}$ is y_ix_{i+1}. We can check that $V_{i+1} = \{1 + m_i, \ldots, m_{i+1}\} = \sigma(Y_i \uplus X_{i+1})$. Hence v_{i+1} is the subword of v corresponding to positions in $\sigma(Y_i \uplus X_{i+1})$. Since $u \Rightarrow_{\mathrm{SC}} v$ by σ, we deduce that $y_ix_{i+1} \Rightarrow_{\mathrm{SC}} v_{i+1}$ (essentially by σ). Therefore, $y_ix_{i+1} \to^*_{\mathrm{SC}} v_{i+1}$ since these two words are finite. This concludes the proof of the claim.

The claim implies $u \to^{\omega}_{\mathrm{SC}} w$ and since $x_0 = v_0$, we also get $w \to^{\omega}_{\mathrm{SC}} v$. □

COROLLARY 4.11. *A language $L \subseteq \Sigma^{\infty}$ is* SC*-closed if and only if it is closed under $\to^{\omega}_{\mathrm{SC}}$: if $u \in L$ and $u \to^{\omega}_{\mathrm{SC}} v$ then $v \in L$.*

To prove Theorem 4.8, we provide an even simpler characterization of SC-closure *for regular languages*. We let $\Rrightarrow_{\text{SC}}$ be the reflexive closure of $\to_{\text{SC}}$, and write $u \Rrightarrow^{\omega}_{\text{SC}} v$ if there are infinite factorizations $u = u_0u_1\dots$ and $v = v_0v_1\dots$ such that $u_i \Rrightarrow_{\text{SC}} v_i$ for all $i \geq 0$. The following proposition is reminiscent of [Peled et al. 1998, Theorem 15].

PROPOSITION 4.12. *An ω-regular language $L \subseteq \Sigma^\infty$ is* SC*-closed if and only if it is closed under $\Rrightarrow^{\omega}_{\text{SC}}$.*

To prove Proposition 4.12, we use the algebraic definition of regular languages. We recall first some useful definitions and properties. We refer to [Perrin and Pin 2004; Carton et al. 2008] for more details. Let $h : \Sigma^* \to M$ be a morphism to some finite monoid M. Below, we only consider morphisms satisfying $h^{-1}(1_M) = \{\varepsilon\}$. Two words $u, v \in \Sigma^\infty$ are h-similar, denoted $u \sim_h v$, if we can find infinite factorizations $u = u_0u_1u_2\cdots$, $v = v_0v_1v_2\cdots$ with $u_i, v_i \in \Sigma^*$ and $h(u_i) = h(v_i)$ for all $i \geq 0$. From the hypothesis $h^{-1}(1_M) = \{\varepsilon\}$, we deduce that if $u \sim_h v$ then either $u, v \in \Sigma^*$ are both finite or $u, v \in \Sigma^\omega$ are both infinite. Moreover, if $u, v \in \Sigma^*$ are finite then $u \sim_h v$ if and only if $h(u) = h(v)$. Note that $\sim_h$ is not necessarily transitive on infinite words.

A language $L \subseteq \Sigma^\infty$ is recognized (saturated) by h if $u \in L$ and $u \sim_h v$ implies $v \in L$ for all $u, v \in \Sigma^\infty$. A *linked pair* of the monoid M is a pair $(s, e) \in M^2$ such that $se = s$ and $e^2 = e$. We recall now the following classical lemma, as presented in [Carton et al. 2008].

LEMMA 4.13 (RAMSEYAN FACTORIZATION). *Let M be a finite monoid and $h : \Sigma^* \to M$ be a morphism. Let $u_0u_1u_2\cdots \in \Sigma^\infty$ be an infinite factorization with $u_i \in \Sigma^*$ for all $i \geq 0$. There exists a linked pair $(s, e) \in M^2$, and there exists $0 < i_1 < i_2 < \cdots$ such that $h(u_0\cdots u_{i_1}) = s$ and $h(u_{1+i_j}\cdots u_{i_{j+1}}) = e$ for all $j > 0$.*

Proof of Proposition 4.12. We fix a finite monoid M and a morphism $h : \Sigma^* \to M$ recognizing L.

Since $\Rrightarrow_{\text{SC}} \subseteq \to^*_{\text{SC}}$, if L is SC-closed then L is closed under $\to^{\omega}_{\text{SC}}$ by Corollary 4.11, hence it is also closed under $\Rrightarrow^{\omega}_{\text{SC}}$.

Conversely, assume that L is not SC-closed, hence not closed under $\to^{\omega}_{\text{SC}}$ by Corollary 4.11. Consider two words $u, v \in \Sigma^\infty$ such that $u \in L$, $v \notin L$ and $u \to^{\omega}_{\text{SC}} v$. Then we have factorizations $u = u_0u_1\cdots$ and $v = v_0v_1\cdots$ with $u_i, v_i \in \Sigma^*$ and $u_i \to^*_{\text{SC}} v_i$ for all $i \geq 0$. Consider now a Ramseyan h-factorization of $u_0u_1u_2\cdots$ given by the sequence $0 < i_1 < i_2\cdots$. Let $u'_0 = u_0\cdots u_{i_1}$ and $u'_j = u_{1+i_j}\cdots u_{i_{j+1}}$ for $j > 0$ so that $h(u'_0) = s$, $h(u'_j) = e$ for all $j > 0$, $se = s$, $e^2 = e$. Let also $v'_0 = v_0\cdots v_{i_1}$ and $v'_j = v_{1+i_j}\cdots v_{i_{j+1}}$ for $j > 0$. Clearly we have $u'_j \to^*_{\text{SC}} v'_j$ for all $j \geq 0$.

Similarly, considering now a Ramseyan h-factorization for $v'_0v'_1v'_2\cdots$ we obtain new factorizations $u = u''_0u''_1u''_2\cdots$ and $v = v''_0v''_1v''_2\cdots$ such that $u''_i \to^*_{\text{SC}} v''_i$ for all $i \geq 0$, $h(u''_0) = s$, $h(v''_0) = t$, $h(u''_i) = e$ and $h(v''_i) = f$ for all $i > 0$, (s, e) and (t, f) linked pairs.

Now, since $u \in L$ and $u \sim_h u''_0(u''_1)^\omega$ we get $u''_0(u''_1)^\omega \in L$. Also, $v \notin L$ and $v \sim_h v''_0(v''_1)^\omega$ implies $v''_0(v''_1)^\omega \notin L$. However, $u''_0 \to^*_{\text{SC}} v''_0$ and $u''_1 \to^*_{\text{SC}} v''_1$, hence there exists $n \geq 0$ such that both rewritings use at most n steps: $u''_0 \Rrightarrow^{n}_{\text{SC}} v''_0$ and $u''_1 \Rrightarrow^{n}_{\text{SC}} v''_1$, where $\Rrightarrow^{n}_{\text{SC}}$ is the n-th iteration of the relation $\Rrightarrow_{\text{SC}}$. We deduce that $u''_0(u''_1)^\omega\ (\Rrightarrow^{\omega}_{\text{SC}})^n\ v''_0(v''_1)^\omega$. Therefore, L is not closed under $\Rrightarrow^{\omega}_{\text{SC}}$. □

THEOREM 4.14. *Given an ω-regular language $L \subseteq \Sigma^\infty$ described by a Büchi automaton, and a semi-commutation relation* SC*, we can decide whether L is* SC*-closed in* PSPACE *with respect to the size of the automaton.*

PROOF. We assume that L is given by a Büchi automaton A. We can compute a Büchi automaton B for the complement $\overline{L}$. As in [Peled et al. 1998], it is easy to build a Büchi automaton C over the alphabet $\Sigma \times \Sigma$ that recognizes $\{(u, v) \in \Sigma^\infty \times \Sigma^\infty \mid$

$u \Rrightarrow^{\omega}_{\mathrm{SC}} v\}$. Synchronizing A with the first track of C and B with the second track of C, we obtain a Büchi automaton, denoted $A \times C \times B$, recognizing

$$(L \times \Sigma^{\infty}) \cap \mathcal{L}(C) \cap (\Sigma^{\infty} \times \overline{L}) = \{(u, v) \in L \times \overline{L} \mid u \Rrightarrow^{\omega}_{\mathrm{SC}} v\}.$$

Then, L is closed under $\Rrightarrow^{\omega}_{\mathrm{SC}}$ if and only if $\mathcal{L}(A \times C \times B)$ is empty. Classical constructions for the complement yield an automaton B of exponential size. However, in order to check $A \times C \times B$ for emptiness, it is not necessary to construct B first. Since emptiness reduces to repeated reachability, we can solve the problem in polynomial space with a non-deterministic procedure. We conclude since $\mathrm{PSPACE} = \mathrm{NPSPACE}$. $\square$

4.4. MSO for semi-traces.

We introduce a *syntactic restriction* of the Monadic Second Order Logic (MSO) over words so that the semantics of any sentence defines a language closed under semi-commutations. Given an alphabet Γ, two finite sets of variables Var and SetVar, a semi-commutation relation $\mathrm{SC} \subseteq \Gamma \times \Gamma$ and the corresponding semi-dependence relation SD, we let $\mathsf{MSO}_{\mathsf{acc}}(\mathrm{SD})$ be the set of formulas defined by:

$$\begin{aligned} \varphi \ ::= &\ P_a(x) \mid \neg P_a(x) \mid x \in X \mid x \notin X \mid x = y \mid x \neq y \mid x\ E_{\mathrm{SD}}\ y \\ &\mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists x.\varphi \mid \forall x.\varphi \mid \exists X.\varphi \mid \forall X.\varphi \end{aligned}$$

for $x \in \mathrm{Var}$, $X \in \mathrm{SetVar}$, and $a \in \Gamma$.

The semantics defines when $w, \nu \models \varphi$, where $w = (\mathrm{Pos}(w), \leq, w)$ is a (finite or infinite) word, and $\nu : \mathrm{Var} \cup \mathrm{SetVar} \to \mathrm{Pos}(w) \cup 2^{\mathrm{Pos}(w)}$ is a valuation such that $\nu(x) \in \mathrm{Pos}(w)$ for all $x \in \mathrm{Var}$ and $\nu(X) \in 2^{\mathrm{Pos}(w)}$ for all $X \in \mathrm{SetVar}$.

— $w, \nu \models P_a(x)$ if $w(\nu(x)) = a$
— $w, \nu \models \neg P_a(x)$ if $w(\nu(x)) \neq a$
— $w, \nu \models x \in X$ if $\nu(x) \in \nu(X)$
— $w, \nu \models x \notin X$ if $\nu(x) \notin \nu(X)$
— $w, \nu \models x = y$ iff $\nu(x) = \nu(y)$
— $w, \nu \models x \neq y$ iff $\nu(x) \neq \nu(y)$
— $w, \nu \models x\ E_{\mathrm{SD}}\ y$ if $\nu(x) < \nu(y)$ and $(w(\nu(x)), w(\nu(y))) \in \mathrm{SD}$
— $w, \nu \models \varphi_1 \wedge \varphi_2$ iff $w, \nu \models \varphi_1$ and $w, \nu \models \varphi_2$
— $w, \nu \models \varphi_1 \vee \varphi_2$ iff $w, \nu \models \varphi_1$ or $w, \nu \models \varphi_2$
— $w, \nu \models \exists x.\varphi$ if there is $i \in \mathrm{Pos}(w)$ such that $w, \nu[x \mapsto i] \models \varphi$
— $w, \nu \models \forall x.\varphi$ if for all $i \in \mathrm{Pos}(w)$, $w, \nu[x \mapsto i] \models \varphi$
— $w, \nu \models \exists X.\varphi$ if there is $S \subseteq \mathrm{Pos}(w)$ such that $w, \nu[X \mapsto S] \models \varphi$
— $w, \nu \models \forall X.\varphi$ if for all $S \subseteq V$, $w, \nu[X \mapsto S] \models \varphi$

Remark 4.15. It is crucial to forbid the negation of a general formula to keep the fragment closed under semi-commutation. Indeed, let $(a, b) \in \mathrm{SD}$ and $(b, a) \notin \mathrm{SD}$, then $L = \{ab\}$ is closed under semi-commutation, but $\overline{L}$ is not, because $ba \to_{\mathrm{SC}} ab$ and $ba \in \overline{L}$.
Also, some atomic propositions are redundant since we can express $\neg P_a(x)$ by $\bigvee_{b \neq a} P_b(x)$ and $x \neq y$ by $x\ E_{\mathrm{SD}}\ y \vee y\ E_{\mathrm{SD}}\ x \vee \bigvee_{a \in \Gamma}(P_a(x) \wedge \neg P_a(y))$.

Remark 4.16. The reflexive and transitive closure of E_{SD} is given by $x \leq_{\mathrm{SD}} y = (x = y) \vee \bigvee_{0 < k < |\Gamma|} \exists z_1 \ldots \exists z_{k-1}.x\ E_{\mathrm{SD}}\ z_1\ E_{\mathrm{SD}} \ldots E_{\mathrm{SD}}\ z_{k-1}\ E_{\mathrm{SD}}\ y$. Indeed, suppose that $w, \nu \models z_0\ E_{\mathrm{SD}}\ z_1\ E_{\mathrm{SD}} \ldots E_{\mathrm{SD}}\ z_{|\Gamma|}$. Necessarily, by the pigeonhole principle, there are two different variables labeled with the same letter of Γ: $0 \leq i < j \leq |\Gamma|$ such that $w(\nu(z_i)) = w(\nu(z_j))$. Then we obtain that $w, \nu \models z_0\ E_{\mathrm{SD}} \ldots E_{\mathrm{SD}}\ z_{i-1}\ E_{\mathrm{SD}}\ z_j\ E_{\mathrm{SD}}\ z_{j+1}\ E_{\mathrm{SD}} \ldots E_{\mathrm{SD}}\ z_{|\Gamma|}$. The remark follows.

If $\Gamma' \subseteq \Gamma$ is a semi-dependence *clique* (i.e., for all $a, b \in \Gamma'$, we have $(a,b),(b,a) \in \mathrm{SD}$), then, *when restricted to positions labeled by* Γ', we have $\neg(x\ E_{\mathrm{SD}}\ y) = (x = y) \vee (y\ E_{\mathrm{SD}}\ x)$. Thus, when restricted to Γ', $\mathsf{MSO}_{\mathsf{acc}}(\mathrm{SD})$ has the same expressive power as $\mathsf{MSO}(<)$.

To show that the models of a sentence of $\mathsf{MSO}_{\mathsf{acc}}(\mathrm{SD})$ form a language closed under SC, we build on the following lemma.

LEMMA 4.17. *For all $w, w' \in \Sigma^\infty$ such that $w \Rightarrow_{\mathrm{SC}} w'$ by σ, all $\varphi \in \mathsf{MSO}_{\mathsf{acc}}(\mathrm{SD})$ and all valuation $\nu : \mathrm{Var} \cup \mathrm{SetVar} \to \mathrm{Pos(w)} \cup 2^{\mathrm{Pos}}(\mathrm{w})$, if $w, \nu \models \varphi$, then $w', \sigma \circ \nu \models \varphi$.*

PROOF. We show it by induction on the structure of the formula φ. Let $w = (\mathrm{Pos}(w), \leq, w)$, $w' = (\mathrm{Pos}(w'), \leq, w')$ and $\sigma : \mathrm{Pos}(w) \to \mathrm{Pos}(w')$ bijection such that $w \Rightarrow_{\mathrm{SC}} w'$ by σ.

— Let $a \in \Gamma$. Then, $w, \nu \models P_a(x)$ iff $w(\nu(x)) = a$ iff $w'(\sigma \circ \nu(x)) = a$ (by Definition 4.4) iff $w', \sigma \circ \nu \models P_a(x)$.
— $w, \nu \models x \in X$ iff $\nu(x) \in \nu(X)$ iff $\sigma(\nu(x)) \in \sigma(\nu(X))$ iff $w', \sigma \circ \nu \models x \in X$.
— $w, \nu \models x = y$ iff $\nu(x) = \nu(y)$ iff $\sigma(\nu(x)) = \sigma(\nu(y))$ iff $w', \sigma \circ \nu \models x = y$.
— The cases $\neg P_a(x)$, $x \notin X$ and $x \neq y$ follow since the above cases are equivalences.
— $w, \nu \models x\ E_{\mathrm{SD}}\ y$ iff $\nu(x) < \nu(y)$ and $\big(w(\nu(x)), w(\nu(y))\big) \in \mathrm{SD}$. Then, by Definition 4.4, $\sigma(\nu(x)) < \sigma(\nu(y))$ and $(w'(\sigma \circ \nu(x)), w'(\sigma \circ \nu(y))) \in \mathrm{SD}$. Hence, $w', \sigma \circ \nu \models x\ E_{\mathrm{SD}}\ y$.
— Conjunctions and disjunctions are trivial.
— $w, \nu \models \exists x.\varphi$ iff there exists $i \in \mathrm{Pos}(w)$ such that $w, \nu[x \mapsto i] \models \varphi$. Then, by induction hypothesis, $w', \sigma \circ (\nu[x \mapsto i]) \models \varphi$, and $w', (\sigma \circ \nu)[x \mapsto \sigma(i)] \models \varphi$, and then $w', \sigma \circ \nu \models \exists x.\varphi$
— $w, \nu \models \forall x.\varphi$ iff for all $i \in \mathrm{Pos}(w)$, $w, \nu[x \mapsto i] \models \varphi$. Then, by induction hypothesis, $w', \sigma \circ (\nu[x \mapsto i]) \models \varphi$ for all $i \in \mathrm{Pos}(w)$, and $w', (\sigma \circ \nu)[x \mapsto \sigma(i)] \models \varphi$. Since σ is bijective, we get $w', (\sigma \circ \nu)[x \mapsto i] \models \varphi$ for all $i \in \mathrm{Pos}(w)$ and then $w', \nu \models \forall x.\varphi$.
— $w, \nu \models \exists X.\varphi$ iff there exists $S \subseteq \mathrm{Pos}(w)$ such that $w, \nu[X \mapsto S] \models \varphi$. Then, by induction hypothesis, $w', \sigma \circ (\nu[X \mapsto S]) \models \varphi$, and $w', (\sigma \circ \nu)[X \mapsto \sigma(S)] \models \varphi$ and then $w', \nu \models \exists X.\varphi$.
— $w, \nu \models \forall X.\varphi$ iff for all $S \subseteq \mathrm{Pos}(w)$, $w, \nu[X \mapsto S] \models \varphi$. Then, by induction hypothesis, $w', \sigma \circ (\nu[X \mapsto S]) \models \varphi$ for all $S \subseteq \mathrm{Pos}(w)$, and $w', (\sigma \circ \nu)[X \mapsto \sigma(S)] \models \varphi$. Again, since σ is bijective, we get $w', (\sigma \circ \nu)[X \mapsto S] \models \varphi$ for all $S \subseteq \mathrm{Pos}(w)$ and then $w', \sigma \circ \nu \models \forall X.\varphi$. $\square$

We say that a given logic is closed under a semi-commutation relation, if, for all φ sentence of this logic, the set $[\![\varphi]\!] = \{w \in \Gamma^\infty \mid w \models \varphi\}$ is closed under this semi-commutation relation. From the preceding lemma, we immediately get the following proposition.

PROPOSITION 4.18. *The logic* $\mathsf{MSO}_{\mathsf{acc}}(\mathrm{SD})$ *is closed under* SC.

PROOF. Let $\varphi \in \mathsf{MSO}_{\mathsf{acc}}(\mathrm{SD})$ be a sentence and $L = [\![\varphi]\!] = \{w \in \Gamma^\infty \mid w \models \varphi\}$. Let $w' \in [L]$. Then, by Lemma 4.17, $w' \models \varphi$ and $w' \in L$. Hence, $L = [L]$. $\square$

From Proposition 4.18, we deduce the following corollary.

COROLLARY 4.19. *Given an architecture $\mathcal{A} = (\mathrm{Proc}, E, (\mathrm{In}_p)_{p \in \mathrm{Proc}}, (\mathrm{Out}_p)_{p \in \mathrm{Proc}})$, the logic $\mathsf{MSO}_{\mathsf{acc}}(\mathrm{SD}_{\mathcal{A}})$ is SC-closed for $\mathcal{A}$.*

Remark 4.20. For any process $p \in \mathrm{Proc}$, the set Σ_p is a semi-dependence clique, then $\mathsf{MSO}_{\mathsf{acc}}(\mathrm{SD}_{\mathcal{A}})$ restricted to p has the same expressive power as $\mathsf{MSO}(<)$.

Moreover, when $\mathrm{SD} = \mathrm{SD}_{\mathcal{A}}$, the reflexive and transitive closure $\leq_{\mathrm{SD}}$ of E_{SD} is simply given by $x \leq_{\mathrm{SD}} y = (x = y) \vee (x\ E_{\mathrm{SD}}\ y) \vee \exists z.(x\ E_{\mathrm{SD}}\ z\ E_{\mathrm{SD}}\ y)$. Indeed, assume that $w, \nu \models x\ E_{\mathrm{SD}}\ y\ E_{\mathrm{SD}}\ z\ E_{\mathrm{SD}}\ t$. Let p be such that $w(\nu(t)) \in \Sigma_p$. If $w(\nu(x)) \in \Sigma_p$ or

$w(\nu(t)) \in \mathrm{Out}_p$, then $w, \nu \models x\ E_{\mathrm{SD}}\ t$. Otherwise, $w(\nu(t)) \in \mathrm{In}_p$ and $w(\nu(x)) \notin \Sigma_p$. Since $w, \nu \models z\ E_{\mathrm{SD}}\ t$, we have $w(\nu(z)) \in \Sigma_p$. If $w(\nu((z)) \in \mathrm{Out}_p$ then $w, \nu \models x\ E_{\mathrm{SD}}\ z$ and $w, \nu \models x\ E_{\mathrm{SD}}\ z\ E_{\mathrm{SD}}\ t$. If now $w(\nu(z)) \in \mathrm{In}_p$ then $w(\nu(y)) \in \Sigma_p$ too. Then, y being on the same process as t, we have $w, \nu \models y\ E_{\mathrm{SD}}\ t$. Hence, $w, \nu \models x\ E_{\mathrm{SD}}\ y\ E_{\mathrm{SD}}\ t$.

5. DECIDABILITY RESULTS

In this section we give a necessary condition for the existence of a distributed strategy. This condition is the existence of a (centralized) strategy implementing the specification on a corresponding architecture consisting of a unique process (such architectures are called singleton architectures). Then we show that it becomes also a sufficient condition for the subclass of architectures having a strongly connected communication graph: every process can transmit messages to everyone (though maybe not directly). In the following, we will simply call them strongly connected architectures. This result allows to conclude that fair synthesis problem is decidable for the subclass of strongly connected architectures.

5.1. Singleton Architectures

A first step in solving the general problem is to handle the sequential case. This problem is slightly different from the asynchronous synthesis of Pnueli and Rosner [1989] (where the communication was through shared variables) and Madhusudan and Thiagarajan [2002] (where a single process does not evolve asynchronously with respect to its environment).

For singleton architectures, there is no internal action and then $\Sigma = \Gamma = \mathrm{In} \cup \mathrm{Out}$. We show that the fair synthesis problem for such architectures is decidable. In fact, in the remainder of the article, we will need a stronger result on singleton architectures. Hence, Theorem 5.1 states that the synthesis problem is decidable for the singleton architecture *whatever partition of controllable actions is chosen* to define the fairness condition. In that case, the partition is part of the input $(\mathcal{A}, \mathcal{P}, L)$ of the synthesis problem: given a singleton architecture $\mathcal{A}$, a specification language L and a partition $\mathcal{P}$ of Out, does there exist a winning strategy for $(\mathcal{A}, \mathcal{P}, L)$.

THEOREM 5.1. *The fair synthesis problem is decidable for singleton architectures, ω-regular specifications, and any partition of controllable actions.*

The rest of this subsection is devoted to the proof of this theorem. As in [Vardi 1995], the general idea is to reduce the synthesis problem to the emptiness problem for a suitably constructed tree automaton.

We suppose that the specification is effectively given by a *deterministic* word automaton (e.g., with Muller acceptance condition) accepting a regular language $L \subseteq \Gamma^\infty$. Let $\mathcal{P}$ be the partition of controllable actions.

We deal first with the case $\mathrm{In} = \emptyset$, for which we claim that there is a winning strategy if and only if $L \neq \emptyset$. Indeed, assume that $\emptyset \neq L \subseteq \mathrm{Out}^\infty$ and let $w \in L$. We define the strategy f on *strict* prefixes of w by $f(v) = a$ if $va \in \mathrm{Pref}(w)$. Note that the f-runs are exactly w itself and its strict prefixes. Now, a strict prefix of w is not f-maximal, hence not $(\mathcal{P}, f)$-fair. Moreover, we see easily that w is $(\mathcal{P}, f)$-fair. Hence w is the unique $(\mathcal{P}, f)$-fair f-run and since $w \in L$, the strategy f is winning. Conversely, let f be a winning strategy. There is a unique f-maximal $(\mathcal{P}, f)$-run $w = w_0 w_1 w_2 \cdots$ defined inductively by $w_i = f(w[i])$ as long as this is defined. If w is finite then $w \notin \mathrm{dom}(f)$, hence w is $(\mathcal{P}, f)$-fair. If w is infinite, then also w is $(\mathcal{P}, f)$-fair. Since f is winning, we get $w \in L \neq \emptyset$. Now, L being effectively given by an automaton, it is decidable to check its emptiness.

So in the following, we assume $\mathrm{In} \neq \emptyset$.

We recall now some notions and notations about trees and tree automata. Given a finite set X and a set Y, a Y-labeled X-tree, also called (X,Y)-tree, is a (partial) function $t : X^* \to Y$ whose domain is prefix-closed, in which elements of X are called *directions* and elements of Y *labels*. When the function is total, the tree is said to be *complete*. A word $w \in \mathrm{dom}(t)$ defines a *node* of t and $t(w)$ is its *label*. The empty word ε is the *root* of the tree. A *finite branch* of t is a node $w \in \mathrm{dom}(t)$ that is maximal: $wX \cap \mathrm{dom}(t) = \emptyset$. An infinite word $w \in X^\omega$ is a *branch* of t if all its finite prefixes are nodes of t: $\mathrm{Pref}(w) \subseteq \mathrm{dom}(t)$. We denote by $\mathrm{Br}(t)$ the set of finite or infinite branches of t. We define the *cumulative label* of a node $w \in \mathrm{dom}(t)$ by $\overline{t}(w) = t(w[0])t(w[1]) \cdots t(w)$. This is extended to (infinite) branches $w \in \mathrm{Br}(t)$ by $\overline{t}(w) = t(w[0])t(w[1])t(w[2]) \cdots$.

A *tree automaton* over (X,Y)-trees is a tuple $\mathfrak{A} = (Q, X, Y, Q_0, \delta, \alpha)$, where Q is a finite set of states, $Q_0 \subseteq Q$ is the set of possible initial states, X and Y are two finite alphabets, $\delta \subseteq Q \times Y \times \bigcup_{S \subseteq X} Q^S$ is the transition function, and α is the acceptance condition, which defines a subset of Q^∞.

A *run tree* of $\mathfrak{A}$ over a tree $t : X^* \to Y$ is another tree $\rho : X^* \to Q$ such that $\mathrm{dom}(\rho) = \mathrm{dom}(t)$, $\rho(\varepsilon) \in Q_0$, and for all $w \in \mathrm{dom}(t)$ we have $(\rho(w), t(w), (\rho(ws))_{s \in S}) \in \delta$ where S is the set of sons of w: $\mathrm{dom}(t) \cap wX = wS$. A branch w of ρ is accepting if $\overline{\rho}(w) \in Q^\infty$ satisfies the acceptance condition. The run tree is *accepting* if all its branches are accepting.

A tree t is *accepted* by a tree automaton $\mathfrak{A}$ if there is an accepting run tree of $\mathfrak{A}$ over t. We define $\mathcal{L}(\mathfrak{A}) = \{t : X^* \to Y \mid t \text{ is accepted by } \mathfrak{A}\}$. We say that $\mathfrak{A}$ is empty (respectively nonempty) if $\mathcal{L}(\mathfrak{A}) = \emptyset$ (respectively $\mathcal{L}(\mathfrak{A}) \neq \emptyset$).

For a given strategy f, we can gather in a *tree* the set of f-runs: we call such trees computation trees. Those are basically $(\mathrm{Out} \cup \{\#\})$-labeled Γ-trees, where the direction of a node represents the last action executed, and its label contains the value the strategy is willing to play after the finite run represented by the node, or $\# \notin \Gamma$ if the strategy does not advise any action (i.e., if f is undefined). Then, each node has as many sons as there are possible actions for the next step : one son for each input action, and one for the output action defined by the strategy, if any. The different possible runs are represented in the branches of the tree.

We will build a tree automaton over computation trees that will accept exactly those corresponding to winning strategies. For the tree automaton to be able to determine if runs corresponding to branches are correct, we make explicit the direction of each node in its label. So we define the labeling alphabet as

$$X = (\Gamma \cup \{\iota\}) \times (\mathrm{Out} \cup \{\#\}).$$

Here $\iota \notin \Gamma$ will be used only at the root since it has no direction. For any letter $(a,b) \in X$, we define the projections $\pi_1(a,b) = a$ and $\pi_2(a,b) = b$, and we extend this definition to words over X^∞ in the natural way.

Formally, for a strategy $f : \Gamma^* \to \mathrm{Out}$, we define its computation tree $\mathrm{Comp}_f : \Gamma^* \to X$ in the following way:

$$\mathrm{Comp}_f(\varepsilon) = \begin{cases} (\iota, f(\varepsilon)) & \text{if } f(\varepsilon) \text{ is defined} \\ (\iota, \#) & \text{otherwise} \end{cases}$$

and, for all w such that $\mathrm{Comp}_f(w)$ is already defined, for all $a \in \mathrm{In} \cup \{f(w)\} = \mathrm{In} \cup \{\pi_2(\mathrm{Comp}_f(w))\} \setminus \{\#\}$,

$$\mathrm{Comp}_f(wa) = \begin{cases} (a, f(wa)) & \text{if } f(wa) \text{ is defined} \\ (a, \#) & \text{otherwise.} \end{cases}$$

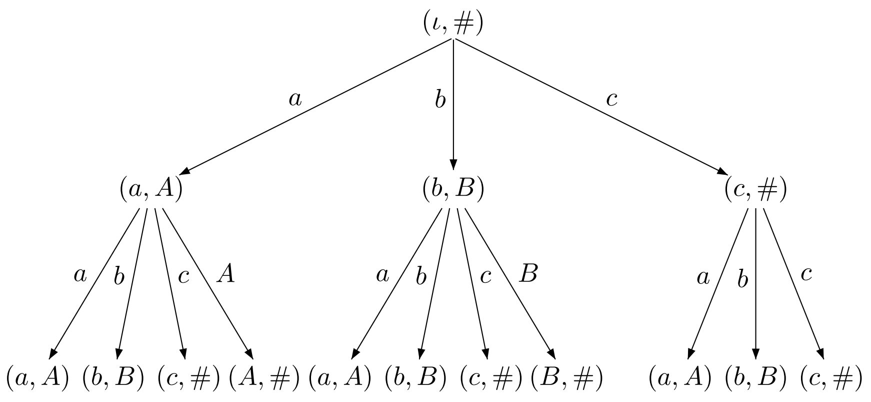

$(\iota, \#)$
a
b
c
(a, A)
(b, B)
$(c, \#)$
a b c A
a b c B
a b c
(a, A) (b, B) $(c, \#)$ $(A, \#)$ (a, A) (b, B) $(c, \#)$ $(B, \#)$ (a, A) (b, B) $(c, \#)$

Fig. 4. A tree Comp_f

Example 5.2. Consider a singleton architecture with alphabets $\mathrm{In} = \{a, b, c\}$ and $\mathrm{Out} = \{A, B\}$. The strategy $f : \Gamma^* \to \mathrm{Out}$ defined for $w \in \Gamma^*$ by

$$f(wa) = A \qquad f(wb) = B$$

is associated with the tree Comp_f that is partly represented on Figure 4.

Remark 5.3. We can easily show by induction that, for all $w \in \Gamma^*$, w is an f-run if and only if $w \in \mathrm{dom}(\mathrm{Comp}_f)$. Moreover, an f-run $w \in \Gamma^*$ is f-maximal if and only if $\pi_2(\mathrm{Comp}_f(w)) = \#$. We deduce also that, for all $w \in \Gamma^\omega$, w is an f-run if and only if w is a branch of Comp_f. Moreover, since $\mathrm{In} \neq \emptyset$, all branches of Comp_f are infinite.

We will show that the set of computation trees of winning strategies is regular. To this end, we define the set WF of *well-formed* trees $t : \Gamma^* \to X$ such that

— $\varepsilon \in \mathrm{dom}(t)$ and $\pi_1(t(\varepsilon)) = \iota$, moreover if $wa \in \mathrm{dom}(t)$ and $a \in \Gamma$ then $\pi_1(t(wa)) = a$,
— for all $w \in \mathrm{dom}(t)$, we have

$$\mathrm{dom}(t) \cap w\Gamma = \begin{cases} w\mathrm{In} & \text{if } \pi_2(t(w)) = \# \\ w(\mathrm{In} \cup \{\pi_2(t(w))\}) & \text{otherwise.} \end{cases}$$

Remark 5.4. The following properties are easy to check:

(1) The set WF is a regular tree language.
(2) If $t \in \mathrm{WF}$ then, for each $w \in \mathrm{Br}(t)$, we have $\pi_1(\overline{t}(w)) = \iota w$.
(3) If $f : \Gamma^* \to \mathrm{Out}$ is a strategy, then $\mathrm{Comp}_f \in \mathrm{WF}$ is well-formed.

We have seen in Remark 5.3 that f-runs correspond to nodes or branches of Comp_f. If the strategy is winning, these f-runs should be either unfair or in the specification language $L \subseteq \Gamma^\infty$. This will be checked on the label of branches using the following languages:

$$\begin{aligned} \mathrm{UF} &= \bigcup_{C \in \mathcal{P}} X^*((\Gamma \setminus C) \times C)^\omega \\ \mathrm{SPEC_{inf}} &= \{w \in X^\omega \mid \pi_1(w) \in \iota(L \cap \Gamma^\omega)\} \\ \mathrm{SPEC_{fin}} &= \{w \in X^\omega \mid \pi_1(u) \in \iota(L \cap \Gamma^*) \text{ for all } u \in \mathrm{Pref}(w) \\ &\qquad\qquad \text{such that } \pi_2(u) \in (\mathrm{Out} \cup \{\#\})^*\#\} \end{aligned}$$

Clearly the language UF for the unfair runs is ω-regular and since L is ω-regular it is also easy to see that the languages $\mathrm{SPEC_{inf}}$ and $\mathrm{SPEC_{fin}}$ for the runs satisfying the specification are ω-regular. We deduce that the tree language

$$T = \{t \in \mathrm{WF} \mid \forall w \in \mathrm{Br}(t),\ \overline{t}(w) \in \mathrm{SPEC_{fin}} \cap (\mathrm{UF} \cup \mathrm{SPEC_{inf}})\}$$

is regular. Moreover, given a *deterministic* word automaton (e.g., with Muller acceptance condition) for L, we can effectively construct a tree automaton for T. The following proposition describes the link between T and the winning strategies for $(\mathcal{A}, \mathcal{P}, L)$.

PROPOSITION 5.5. *We have*

$$T = \{\mathrm{Comp}_f \mid f : \Gamma^* \to \mathrm{Out} \textit{ is a winning strategy for } (\mathcal{A}, \mathcal{P}, L)\}.$$

PROOF. First, let $f : \Gamma^* \to \mathrm{Out}$ be a winning strategy for $(\mathcal{A}, \mathcal{P}, L)$. We show that $\mathrm{Comp}_f \in T$. From Remark 5.4, we already know that $\mathrm{Comp}_f \in \mathrm{WF}$ is well-formed. Let $v \in \mathrm{Br}(\mathrm{Comp}_f)$ and let $w = \overline{\mathrm{Comp}_f}(v)$. By Remark 5.3, we know that $v \in \Gamma^\omega$ is an infinite f-run.

If v is $(\mathcal{P}, f)$-fair then $v \in L$ since f is winning. From Remark 5.4, we know that $\pi_1(w) = \iota v$ and we deduce that $w \in \mathrm{SPEC_{inf}}$.

If v is not $(\mathcal{P}, f)$-fair then we find $C \in \mathcal{P}$ and a prefix v' of v such that $f(v'') \in C$ for all $v' \le v'' < v$ and such that v' contains all the finitely many C-events of v $(\mathrm{alph}(v'^{-1}v) \cap C = \emptyset)$. We deduce that $w \in X^*((\Gamma \setminus C) \times C)^\omega \subseteq \mathrm{UF}$.

It remains to show that $w \in \mathrm{SPEC_{fin}}$. So let u be a finite prefix of w such that its second component ends with a $\#$: $\pi_2(u) \in (\mathrm{Out} \cup \{\#\})^*\#$. Let v' be the prefix of v corresponding to u: $\overline{\mathrm{Comp}_f}(v') = u$. We have $\pi_1(u) = \iota v'$ and $\pi_2(\mathrm{Comp}_f(v')) = \#$. By Remark 5.3 we deduce that v' is an f-maximal f-run. Hence it is $(\mathcal{P}, f)$-fair and we get $v' \in L$ since f is winning. Therefore, $\pi_1(u) = \iota v' \in \iota(L \cap \Gamma^*)$ as desired. We deduce that $w \in \mathrm{SPEC_{fin}}$.

Conversely, let $t \in T$. We define the strategy $f : \Gamma^* \to \mathrm{Out}$ as follows: for all $v \in \Gamma^*$, we let

$$f(v) = \begin{cases} \pi_2(t(v)) & \text{if } v \in \mathrm{dom}(t) \text{ and } \pi_2(t(v)) \neq \# \\ \text{undefined} & \text{otherwise.} \end{cases}$$

CLAIM 5.6. $t = \mathrm{Comp}_f$.

We first show that if $v \in \mathrm{dom}(t) \cap \mathrm{dom}(\mathrm{Comp}_f)$ then $t(v) = \mathrm{Comp}_f(v)$. For the first component, we have $\pi_1(t(\varepsilon)) = \iota = \pi_1(\mathrm{Comp}_f(\varepsilon))$ and if $v = v'a$ then $\pi_1(t(v)) = a = \pi_1(\mathrm{Comp}_f(v))$ since t is well-formed and by definition of Comp_f. For the second component, if $\pi_2(t(v)) = \#$ then $f(v)$ is undefined by definition of f and we obtain $\pi_2(\mathrm{Comp}_f(v)) = \#$ by definition of Comp_f. Next, if $\pi_2(t(v)) = b \in \mathrm{Out}$ then $f(v) = b$ by definition of f and we obtain $\pi_2(\mathrm{Comp}_f(v)) = b$ by definition of Comp_f.

Second, we show by induction that $\mathrm{dom}(t) = \mathrm{dom}(\mathrm{Comp}_f)$. Clearly, $\varepsilon \in \mathrm{dom}(t) \cap \mathrm{dom}(\mathrm{Comp}_f)$. Let now $v \in \mathrm{dom}(t) \cap \mathrm{dom}(\mathrm{Comp}_f)$. Since both Comp_f and t are well-formed, we have

$$\mathrm{dom}(\mathrm{Comp}_f) \cap v\Gamma = \begin{cases} v\mathrm{In} & \text{if } \pi_2(\mathrm{Comp}_f(v)) = \# \\ v(\mathrm{In} \cup \{\pi_2(\mathrm{Comp}_f(v))\}) & \text{otherwise,} \end{cases}$$

$$\mathrm{dom}(t) \cap v\Gamma = \begin{cases} v\mathrm{In} & \text{if } \pi_2(t(v)) = \# \\ v(\mathrm{In} \cup \{\pi_2(t(v))\}) & \text{otherwise.} \end{cases}$$

Since $t(v) = \mathrm{Comp}_f(v)$ we deduce that $\mathrm{dom}(\mathrm{Comp}_f) \cap v\Gamma = \mathrm{dom}(t) \cap v\Gamma$, which concludes the proof of the claim.

It remains to show that f is winning. Let first $v \in \Gamma^\omega$ be an infinite $(\mathcal{P}, f)$-fair f-run. From the claim and Remark 5.3 we deduce that $v \in \mathrm{Br}(t)$. Then, $w = \bar{t}(v) \in \mathrm{UF} \cup \mathrm{SPEC}_{\mathrm{inf}}$. Assume towards a contradiction that $w \in \mathrm{UF}$. Then, we find $C \in \mathcal{P}$ such that $w \in X^*((\Gamma \setminus C) \times C)^\omega$. We deduce that there is a finite prefix v' of v such that $v \in v'(\Gamma \setminus C)^\omega$ (hence contains finitely many C-events) and $\pi_2(t(v'')) \in C$ for all $v' \leq v'' < v$. We deduce that $f(v'') \in C$ for all $v' \leq v'' < v$, a contradiction with v being $(\mathcal{P}, f)$-fair. Therefore, $w \in \mathrm{SPEC}_{\mathrm{inf}}$ and we obtain $\iota v = \pi_1(w) \in \iota(L \cap \Gamma^\omega)$. Hence, $v \in L$ satisfies the specification.

Let now $v \in \Gamma^*$ be an f-maximal f-run. From the claim and Remark 5.3, we know that $v \in \mathrm{dom}(\mathrm{Comp}_f) = \mathrm{dom}(t)$. Since t is well-formed, we have $v\mathrm{In}^\omega \subseteq \mathrm{Br}(t)$. Let $v' \in v\mathrm{In}^\omega$ be such a branch. We have $w = \bar{t}(v') \in \mathrm{SPEC}_{\mathrm{fin}}$. Now, the word $u = \bar{t}(v)$ is a finite prefix of w that ends with $t(v)$. Since v is f-maximal, $f(v)$ is undefined, hence $\pi_2(t(v)) = \#$. Therefore, $\pi_2(u) \in (\mathrm{Out} \cup \{\#\})^*\#$ and we deduce that $\iota v = \pi_1(u) \in \iota(L \cap \Gamma^*)$. Hence, $v \in L$ satisfies the specification.

We have seen that all $(\mathcal{P}, f)$-fair f-run satisfy the specification. Hence, the strategy f is winning. $\square$

We deduce from the above proposition that there exists a winning strategy for $(\mathcal{A}, \mathcal{P}, L)$ if and only if the regular tree language T is non-empty. Since we can effectively construct a tree automaton for T from a deterministic word automaton for L, and emptiness for tree automata is decidable, we have proved Theorem 5.1. Moreover, if the tree automaton accepts a nonempty language, then we can construct an accepted *regular* tree (i.e., with finitely many subtrees). This regular tree has a finite representation, yielding the existence of a strategy with finite memory.

5.2. Distributed Architectures

We now consider the general case of distributed architectures. We first give a necessary condition for the existence of a distributed strategy for an architecture $\mathcal{A} = (\mathrm{Proc}, E, (\mathrm{In}_p)_{p \in \mathrm{Proc}}, (\mathrm{Out}_p)_{p \in \mathrm{Proc}})$. For that, we define the singleton architecture $\overline{\mathcal{A}}$ with a single process and external signals $\mathrm{In} = \bigcup_{p \in \mathrm{Proc}} \mathrm{In}_p$ and $\mathrm{Out} = \bigcup_{p \in \mathrm{Proc}} \mathrm{Out}_p$. We consider the partition $\overline{\mathcal{P}}$ defined by $\overline{\mathcal{P}} = \{\mathrm{Out}_p \mid p \in \mathrm{Proc}\}$. The existence of a winning strategy for $(\overline{\mathcal{A}}, \overline{\mathcal{P}}, L)$ is a necessary condition for the existence of a winning strategy and communication alphabets for $(\mathcal{A}, \mathcal{P}, L)$.

PROPOSITION 5.7. *Let L be an ω-regular specification. If there are internal signal sets and a distributed winning strategy for $(\mathcal{A}, L)$ then there is a winning strategy for $(\overline{\mathcal{A}}, \overline{\mathcal{P}}, L)$.*

PROOF. Prima facie, it seems easy to simulate a distributed strategy with a centralized one. However, we will have to deal with fairness conditions, which requires some care. Let $(\Sigma_{p,q})_{(p,q) \in E}$ be internal communication sets used by processes of the architecture $\mathcal{A}$ and let $F = (f_p)_{p \in \mathrm{Proc}}$ be the distributed winning strategy. Since the distributed strategy is winning, any F-run that is $(\mathcal{P}, F)$-fair will belong to L. Recall that $\mathcal{P} = \{\Sigma_{p,C} \mid p \in \mathrm{Proc}\}$.

To simulate the distributed strategy F, the singleton architecture should implement a fair scheduling of the processes and then play the actions of the different processes according to this scheduling. For this, we first define for any $p \in \mathrm{Proc}$ a map $\mathrm{rank}(f_p) : \Sigma^* \to \mathbb{N}$ representing the priority associated with process p after a given history on Σ^*.

For $p \in \mathrm{Proc}$, $v \in \Sigma^*$ and $a \in \Sigma$, we define

$$\mathrm{rank}(f_p)(\varepsilon) = \begin{cases} 1 & \text{if } f_p(\varepsilon) \text{ is defined} \\ 0 & \text{otherwise.} \end{cases}$$

$$\mathrm{rank}(f_p)(va) = \begin{cases} \mathrm{rank}(f_p)(v) + 1 & \text{if } a \neq f_p(v) \text{ and } f_p(va) \text{ is defined} \\ 1 & \text{if } a = f_p(v) \text{ and } f_p(va) \text{ is defined} \\ 0 & \text{otherwise.} \end{cases}$$

The priority of a process in Proc strictly increases as long as its strategy is defined and the process is not scheduled. Also, the rank of a process whose strategy is defined is always strictly greater than the rank of a process whose strategy is not.

In the following, we assume that processes in Proc are totally ordered by a given relation $\leq$. For $v \in \Sigma^*$, we define $\mathrm{rank}(F)(v) = \max\{\mathrm{rank}(f^p)(v) \mid p \in \mathrm{Proc}\}$ and $\mathrm{proc}_F(v) = \max\{p \in \mathrm{Proc} \mid \mathrm{rank}(f_p)(v) = \mathrm{rank}(F)(v)\}$, the maximal process among those with maximal priority. It defines the process that will be simulated by the singleton architecture after the prefix v. Observe that, with the definition of $\mathrm{rank}(f_p)$, the singleton architecture will not try to simulate a process whose strategy is undefined if there are other processes enabled.

Recall that the set of signals of $\overline{\mathcal{A}}$ is $\Gamma = \mathrm{In} \cup \mathrm{Out}$. To simulate F with a strategy f of the singleton architecture, we need to turn a sequence in Γ^* (the history available to the singleton architecture) into one of $\mathcal{A}$ that includes internal signals from $(\Sigma_{p,q})_{(p,q)\in E}$.

To this end, for each $v \in \Sigma^*$, we define the sequence $\mathrm{Com}(v)$ of internal communications triggered by v. Formally $\mathrm{Com}(v) = u_0u_1u_2\cdots \in (\Sigma_C \setminus \Gamma)^\infty$ is the maximal word such that for all $i \geq 0$ we have $u_i = f_{p_i}(v \cdot u[i])$ where $p_i = \mathrm{proc}_F(v \cdot u[i])$. Notice that for any prefix u of $\mathrm{Com}(v)$ we have $\mathrm{Com}(v) = u \cdot \mathrm{Com}(v \cdot u)$. The sequence $\mathrm{Com}(v)$ is finite if at the end, all the processes have their rank equal to 0, hence their strategies are all undefined, or if the strategy of the first process in the priority list is to output a signal from Out.

We define now the map $\Phi : \Gamma^* \to \Sigma^*$, which enriches a sequence $v \in \Gamma^*$ with the internal communications obtained using Com in order to get a prefix $\Phi(v)$ of an F-run of $\mathcal{A}$. If the sequence of internal communications advised by Com is finite, it is entirely inserted. Otherwise, only the first internal communication is inserted. When $\mathrm{Com}(u) \neq \varepsilon$, we denote by $\mathrm{FirstCom}(u) = \mathrm{Com}(u)[1]$ the first communication action of $\mathrm{Com}(u)$. Formally, the map Φ is defined inductively as follows:

$$\Phi(\varepsilon) = \begin{cases} \mathrm{Com}(\varepsilon) & \text{if } \mathrm{Com}(\varepsilon) \text{ is finite} \\ \mathrm{FirstCom}(\varepsilon) & \text{otherwise} \end{cases}$$

and for $v \in \Gamma^*$ and $a \in \Gamma$

$$\Phi(va) = \begin{cases} \Phi(v)a\mathrm{Com}(\Phi(v)a) & \text{if } \mathrm{Com}(\Phi(v)a) \text{ is finite} \\ \Phi(v)a\mathrm{FirstCom}(\Phi(v)a) & \text{otherwise.} \end{cases}$$

Since Φ is (strictly) increasing, it can be extended to infinite words $w \in \Gamma^\omega$ using the least upper bound on finite prefixes: $\Phi(w) = \bigsqcup_{v \leq w} \Phi(v) \in \Sigma^\omega$.

After a sequence of actions $v \in \Gamma^*$, the singleton architecture decides which output signal to emit by consulting the distributed strategy F applied to the enriched history $\Phi(v)$ for the top process in the priority list. Formally, the strategy $f : \Gamma^* \to \mathrm{Out}$ of the singleton architecture is defined for $v \in \Gamma^*$ and $p = \mathrm{proc}_F(\Phi(v))$ by

$$f(v) = \begin{cases} f_p(\Phi(v)) & \text{if } f_p(\Phi(v)) \in \mathrm{Out}_p \\ \text{undefined} & \text{otherwise.} \end{cases}$$

Hence, the strategy of the singleton architecture is undefined if $f_p(\Phi(v))$ is undefined or if $f_p(\Phi(v)) \in \Sigma_{p,C} \setminus \mathrm{Out}_p$.

Below, we fix some $(\overline{\mathcal{P}}, f)$-fair f-run $w \in \Gamma^\infty$ of $\overline{\mathcal{A}}$. When the environment emits infinitely many signals, $w \in \Gamma^\omega$ and thus $\Phi(w) \in \Sigma^\omega$. But if after some point the environment does not emit signals anymore, then it may happen that the singleton architecture does not emit any signal either. However, the sequence $\mathrm{Com}(\Phi(w))$ may be infinite, i.e., processes in $\mathcal{A}$ may decide to exchange internal communication signals indefinitely. In that case, $\Phi(w)$ is not a $(\mathcal{P}, F)$-fair F-run. This is why the enriched sequence associated with w is defined by

$$w' = \begin{cases} \Phi(w)\mathrm{Com}(\Phi(w)) & \text{if } w \text{ is finite} \\ \Phi(w) & \text{otherwise.} \end{cases}$$

Notice that $\pi_\Gamma(w') = w$. Moreover, by definition of Φ and Com, we see that w' is finite if and only if w is finite and $\mathrm{Com}(\Phi(w)) = \varepsilon$.

We show that w' is a $(\mathcal{P}, F)$-fair F-run over $\mathcal{A}$.

We show first that w' is an F-run. Let $v'a < w'$ with $a \in \Sigma_{p,C}$ for some $p \in \mathrm{Proc}$. Either a is an internal communication and by definition of Φ and Com, we deduce that $a = f_p(v')$. Or $a \in \mathrm{Out}_p$ and by definition of Φ, we have that $v' = \Phi(v)$ for some prefix v of w. Moreover, since w is an f-run, we deduce that $a = f(v) = f_p(\Phi(v))$ by definition, and again, $a = f_p(v')$. Hence, w' is indeed an F-run.

We show now that w' is $(\mathcal{P}, F)$-fair. We distinguish two separate cases, depending on whether w' is finite or not.

Suppose first that w' is finite. Then, w is finite, $w' = \Phi(w)$ and $\mathrm{Com}(\Phi(w)) = \varepsilon$. By definition of Com, this implies that $f_p(\Phi(w)) \in \mathrm{Out}_p$ for $p = \mathrm{proc}_F(\Phi(w))$, or $f_p(\Phi(w))$ is undefined, for all $p \in \mathrm{Proc}$. The first case would imply that $f(w) \in \mathrm{Out}_p$, a contradiction with w being a $(\overline{\mathcal{P}}, f)$-fair f-run. Therefore, $w' = \Phi(w)$ is F-maximal, hence $(\mathcal{P}, F)$-fair by Remark 3.5.

Assume now that w' is infinite and that w' is not $(\mathcal{P}, f)$-fair. Let $P \subseteq \mathrm{Proc}$ be the set of processes that are eventually always enabled, but never scheduled: there exists $v'_0 < w'$ such that for all $p \in P$ and all $v'_0 \le v' < w'$, we have $f_p(v')$ defined and $\mathrm{alph}({v'_0}^{-1}w') \cap \Sigma_{p,C} = \emptyset$. We get $\mathrm{rank}(f_p)(v'a) = 1 + \mathrm{rank}(f_p)(v')$ for all $v'_0 < v'a < w'$ and all $p \in P$. Let $p \in P$ be the maximal process (wrt. the assumed total ordering on processes) such that $\mathrm{rank}(f_p)(v'_0) = \max(\mathrm{rank}(f_q)(v'_0))_{q \in P}$. We first show that p has eventually always the highest priority.

Claim 5.8. *There is $v'_0 < u' < w'$ such that $p = \mathrm{proc}_F(v')$ for all $u' \le v' < w'$.*

Proof. Indeed, let $q \notin P$. By definition of P, we find $v_q a_q$ such that $v'_0 < v_q a_q < w'$ and

— either $a_q \in \Sigma_{q,C}$ in which case, as shown above, we have $a_q = f_q(v_q)$ and we obtain $\mathrm{rank}(f_q)(v_q a_q) \le 1$,

— or $f_q(v_q a_q)$ is undefined, in which case $\mathrm{rank}(f_q)(v_q a_q) = 0$.

In both cases, $\mathrm{rank}(f_q)(v_q a_q) \le 1 < \mathrm{rank}(f_p)(v_q a_q)$. Therefore, $\mathrm{rank}(f_q)(v') < \mathrm{rank}(f_p)(v')$ for all $v_q a_q \le v' < w'$. We get $\mathrm{proc}_F(v') \neq q$ for all $v_q a_q \le v' < w'$. Let $u' = \bigsqcup_{q \notin P} v_q a_q$. We obtain $\mathrm{proc}_F(v') \in P$ for all $u' \le v' < w'$. By definition of p, we deduce that $\mathrm{proc}_F(v') = p$ for all $u' \le v' < w'$. □

Now, we show that if process p stays continuously on top of the priority list and none of its actions are added in w' then it advises only output signals.

Claim 5.9. *For all $u' < v'a < w'$ we have $a \in \mathrm{In}$ and $f_p(v'a) \in \mathrm{Out}_p$.*

PROOF. Let $u' < v'a < w'$. Since $p = \mathrm{proc}_F(v')$, $a \in \mathrm{In} \cup \Sigma_{p,C}$ (by definition of Com and f). By hypothesis, $a \notin \Sigma_{p,C}$, hence $a \in \mathrm{In}$.

Let $b \in \Sigma$ such that $v'ab < w'$. As above, we get $b \in \mathrm{In}$. By definition of Φ, since $a \in \mathrm{In} \subseteq \Gamma$, $\mathrm{FirstCom}(v'a) = \varepsilon$ or $\mathrm{FirstCom}(v'a) = b \in \Sigma \setminus \Gamma$. The second case contradicts $b \in \mathrm{In}$. Hence, $\mathrm{FirstCom}(v'a) = \varepsilon$ and since $f_p(v'a)$ is defined and $p = \mathrm{proc}_F(v'a)$, we deduce $f_p(v'a) \in \mathrm{Out}_p$. $\square$

We are now in a position to derive a contradiction from our assumption that w' is not $(\mathcal{P}, F)$-fair. Let $u = \pi_\Gamma(u')$. From the claim above, we deduce that $w = \pi_\Gamma(w') = u \cdot ({u'}^{-1}w')$. Then, $\Phi(v) = u' \cdot u^{-1}v$ for all $u < v \leq w$. Then, again by the claim above, $f_p(\Phi(v)) \in \mathrm{Out}_p$. By the first claim, $p = \mathrm{proc}_F(\Phi(v))$, then, by definition of f, we have $f(v) \in \mathrm{Out}_p$. Since the run w is $(\overline{\mathcal{P}}, f)$-fair, $\mathrm{alph}(u^{-1}w) \cap \mathrm{Out}_p \neq \emptyset$. Hence, $\mathrm{alph}({u'}^{-1}w') \cap \mathrm{Out}_p \neq \emptyset$, a contradiction.

Therefore, for each f-run $w \in \Gamma^\infty$ that is $(\overline{\mathcal{P}}, f)$-fair, we can construct an F-run $w' \in \Sigma^\infty$ that is $(\mathcal{P}, F)$-fair and such that $w = \pi_\Gamma(w')$. Since F is a winning strategy, $w = \pi_\Gamma(w') \in L$. This proves that f is a winning strategy for $(\overline{\mathcal{A}}, \overline{\mathcal{P}}, L)$. $\square$

5.2.1. Strongly Connected Architectures. We consider now the class of architectures having a strongly connected communication graph: each process can transmit messages to any other (though maybe not directly). In the following, we will simply call them strongly connected architectures. We show that for a strongly connected architecture $\mathcal{A}$, when the specification L is SC-closed, the existence of a winning strategy for $(\overline{\mathcal{A}}, \overline{\mathcal{P}}, L)$ is a sufficient condition for the existence of a distributed strategy over $(\mathcal{A}, L)$. By Proposition 5.7, it is then a *necessary and sufficient condition*. By Theorem 5.1, we can then state the following result.

THEOREM 5.10. *The fair synthesis problem over strongly connected architectures is decidable for ω-regular SC-closed specifications.*

The proof of the theorem relies on the following proposition.

PROPOSITION 5.11. *Let $\mathcal{A}$ be a strongly connected architecture, and L an ω-regular SC-closed specigication. If there is a winning strategy for $(\overline{\mathcal{A}}, \overline{\mathcal{P}}, L)$, then one can define internal signal sets, and a winning distributed strategy for $(\mathcal{A}, L)$. Moreover, if there is a finite-memory strategy for $(\overline{\mathcal{A}}, \overline{\mathcal{P}}, L)$, then one can construct finite internal communication sets and a finite-memory distributed strategy for the strongly connected architecture $\mathcal{A}$.*

We want to simulate a run of $\overline{\mathcal{A}}$ in the distributed system $\mathcal{A}$. But the processes only observe the projections (on Σ_p) of the actual run, and the (totally ordered) actual run cannot be rebuilt from its projections on Σ_p. Since the processes have to simulate the strategy of the singleton architecture $\overline{\mathcal{A}}$, they have to guess and agree on an *imaginary* totally ordered run that is "compatible" with the actual one: the actual run will be in the semi-commutation closure of the imaginary run. To do so, we use a token passing algorithm: we select a cycle in the communication graph and force the processes to communicate in a sequential way through this virtual ring – note that there may be no simple cycle, and a process may appear several times in the (virtual) ring. The process that has the token will simulate the singleton architecture according to the current imaginary run. While passing the token, it will also transmit enough information to allow the receiver to extend the imaginary run.

Let $f : \Gamma^* \to \Gamma$ be a winning strategy for $(\overline{\mathcal{A}}, \overline{\mathcal{P}}, L)$. We suppose that f is described by a deterministic automaton with ouput – an automaton without accepting conditions, and to which we add an output function defined on states. We say that the strategy has *finite memory* if the automaton that computes it is finite (i.e., has a finite number of states). Let $\mathfrak{A} = (Q, \Gamma, \delta, s_0, \overline{f})$ with

— Q the set of states (finite if the strategy has finite memory) and $s_0 \in Q$ the initial state,
— $\delta : Q \times \Gamma \to Q$ the deterministic and complete transition function,
— $\overline{f} : Q \to \mathrm{Out}$ the partial map describing the strategy: for $v \in \Gamma^*$ we have $f(v) = \overline{f}(\delta(s_0, v))$.

We define for each process $p \in \mathrm{Proc}$ an automaton with output $\mathfrak{A}_p = (Q_p, \Sigma_p, \delta_p, s_0^p, \overline{f}_p)$ computing the local strategy of process p. To do so, we select a cycle of size n in the architecture. We use the auxiliary function ring defined by

$$\mathrm{ring} : \{1, \ldots, n\} \to \mathrm{Proc}$$

It is a surjective map associating each element of the ring to a process of the architecture. It satisfies the property that $(\mathrm{ring}(i), \mathrm{ring}(i+1)) \in E$, for all $1 \leq i < n$ and $(\mathrm{ring}(n), \mathrm{ring}(1)) \in E$. As already pointed out, ring is not necessarily injective and a given process may appear several times on the ring.

From their local observations, the different processes will guess an imaginary run of the singleton architecture, respecting the strategy given by $\mathfrak{A}$, such that the actual run being executed can be obtained by semi-commutation rewritings from this imaginary run. When a process receives the token, it obtains the current imaginary run and updates it by appending the sequence of local signals it has received since the last time it had had the token (in our algorithm, the only communication signals are for token passing). When the strategy of the singleton architecture is finite-state, the processes do not need to keep track of all actions that occurred between two token passings (this would ask for unbounded memory), but need only to compute the corresponding transition function of $\mathfrak{A}$.

Formally, we define the set of states of $\mathfrak{A}_p$, for a process $p \in \mathrm{Proc}$ by

$$Q_p = \Big(Q^Q \times \{\mathrm{NTok}\}\Big) \cup \big(Q \times \bigcup_{i \in \mathrm{ring}^{-1}(p)} \{\mathrm{Tok}_i, \mathrm{Tok}'_i\}\big)$$

where Tok_i, and Tok'_i are flags indicating that the process has the token while simulating the i-th element of the ring, in which case its internal state is the current state of $\mathfrak{A}$ in the imaginary run. If the flag NTok is on, it indicates that the process does not have the token, in which case it memorizes in its state a transition function of $\mathfrak{A}$ abstracting the sequence of actions it has observed.

When a process p does not have the token, it only memorizes the transition function of the sequences of actions it has received, without emitting any signal. Then, the only local actions occurring during that time are inputs from the environment. Hence, for any $p \in \mathrm{Proc}$, for any $\gamma \in Q^Q$ abstracting some input sequence from In_p^*, for any $a \in \mathrm{In}_p$, we let

$$\delta_p((\gamma, \mathrm{NTok}), a) = (\gamma_a \circ \gamma, \mathrm{NTok}) \qquad (\delta 1)$$

where $\gamma_a : Q \to Q$ is defined by $\gamma_a(s) = \delta(s, a)$ for $s \in Q$.

When a process $p \in \mathrm{Proc}$ has the token, it uses the singleton architecture strategy to choose the signal to emit. As long as the singleton architecture strategy advises an action in Out_p and process p has not been scheduled, the strategy of process p will be also to emit this signal. If at some point, the singleton architecture strategy is to output an action controlled by another process, or is undefined, then process p will try to pass on the token. Similarly, as soon as process p has been able to emit a signal in Out_p, it will try to transmit the token to the next process. To model the difference between the process willing to emit an output signal and the process willing to transmit the token (and thus, emit an internal signal), we will use respectively flags Tok_i and Tok'_i. Formally, for all $p \in \mathrm{Proc}$, $i \in \mathrm{ring}^{-1}(p)$, $s \in Q$ and $a \in \mathrm{In}_p \cup \mathrm{Out}_p$,

$$\delta_p((s, \mathrm{Tok}_i), a) = \begin{cases} (\delta(s,a), \mathrm{Tok}_i) & \text{if } a \in \mathrm{In}_p \text{ and } \overline{f}(\delta(s,a)) \in \mathrm{Out}_p \\ (\delta(s,a), \mathrm{Tok}'_i) & \text{otherwise.} \end{cases} \tag{$\delta 2$}$$

If process p was not scheduled and the strategy of the singleton architecture is still to emit a signal in Out_p then it keeps the token. If process p has been scheduled, or when the singleton architecture strategy is not in Out_p, then p changes its flag to Tok'_i in order to transmit the token.

As soon as a process p wants to pass on the token, it won't be able to emit any other signal until it has indeed transmitted the token. Formally, for all $p \in \mathrm{Proc}$, $i \in \mathrm{ring}^{-1}(p)$, for all $s \in Q$ and $a \in \mathrm{In}_p$

$$\delta_p((s, \mathrm{Tok}'_i), a) = (\delta(s,a), \mathrm{Tok}'_i). \tag{$\delta 3$}$$

When passing the token on, a process actually sends the current state of the automaton $\mathfrak{A}$. To make explicit the process that will receive the signal, and obtain pairwise disjoint communication alphabets, we add to the state the number in the ring of the emitting process. So the output function of the automaton $\mathfrak{A}_p$ is defined as follows: for all $s \in Q$, $i \in \mathrm{ring}^{-1}(p)$,

$$\overline{f}_p(s, \mathrm{Tok}_i) = \overline{f}(s)$$
$$\overline{f}_p(s, \mathrm{Tok}'_i) = (s, i)$$

Then, for all $(p,q) \in E$, we define the internal communication alphabets by

$$\Sigma_{p,q} = \biguplus_{i \mid p=\mathrm{ring}(i) \wedge q=\mathrm{ring}((i \bmod n)+1)} Q \times \{i\}.$$

When the process has emitted the signal transmitting the token, it resets its local state memorizing its local history, and goes back in a state NTok: for all $p \in \mathrm{Proc}$, $i \in \mathrm{ring}^{-1}(p)$ and $s \in Q$, we define:

$$\delta_p((s, \mathrm{Tok}'_i), (s,i)) = (\mathrm{id}, \mathrm{NTok}). \tag{$\delta 4$}$$

where $\mathrm{id} \in Q^Q$ is the identity mapping.

The process receiving the token will compute the new current state of $\mathfrak{A}$: for $\gamma \in Q^Q$, $i \in \{1, \cdots, n\}$, $j = (i \bmod n) + 1$ and $p = \mathrm{ring}(j)$,

$$\delta_p((\gamma, \mathrm{NTok}), (s,i)) = \begin{cases} (\gamma(s), \mathrm{Tok}_j) & \text{if } \overline{f}(\gamma(s)) \in \mathrm{Out}_p \\ (\gamma(s), \mathrm{Tok}'_j) & \text{otherwise.} \end{cases} \tag{$\delta 5$}$$

Finally, the initial state of $\mathfrak{A}_p$ is given by:

$$s_0^p = \begin{cases} (s_0, \mathrm{Tok}_1) & \text{if } \mathrm{ring}(1) = p \text{ and } \overline{f}(s_0) \in \mathrm{Out}_p \\ (s_0, \mathrm{Tok}'_1) & \text{if } \mathrm{ring}(1) = p \text{ and } \overline{f}(s_0) \notin \mathrm{Out}_p \\ (\mathrm{id}, \mathrm{NTok}) & \text{otherwise.} \end{cases}$$

Remark 5.12. We have the following invariant: if $\mathfrak{A}_p$ is in state (s, Tok_i) then $p = \mathrm{ring}(i)$, $\overline{f}(s)$ is defined and $\overline{f}_p(s) = \overline{f}(s) \in \mathrm{Out}_p$.

For each $p \in \mathrm{Proc}$, we have defined a deterministic (partial) transition function $\delta_p : Q_p \times \Sigma_p \to Q_p$. Then, we define the local strategy $f_p : \Sigma_p^* \to \Sigma_{p,C}$ by

$$f_p(v) = \overline{f}_p(\delta_p(s_0^p, v))$$

for all $v \in \Sigma_p^*$. Recall that a local strategy is extended to words over Σ^* by $f_p(v) = f_p(\pi_p(v))$ for $v \in \Sigma^*$. Since we will use it quite often, we denote the state reached by $\mathfrak{A}_p$ after $v \in \Sigma^*$ by

$$\overline{s}_p(v) = \delta_p(s_0^p, \pi_p(v))$$

that may be undefined if $\mathfrak{A}_p$ does not have a run over $\pi_p(v)$. With this notation, we have $f_p(v) = \overline{f}_p(\overline{s}_p(v))$.

In the following, we let $\mathrm{Tok}_p = \bigcup_{i \in \mathrm{ring}^{-1}(p)} \{\mathrm{Tok}_i, \mathrm{Tok}'_i\}$ for $p \in \mathrm{Proc}$. If $v \in \Sigma^*$ is a finite prefix of a run, we say that p *has the token in* v if $\overline{s}_p(v) \in Q \times \mathrm{Tok}_p$.

Using these notations, a run $w \in \Sigma^\infty$ is an F-run if, for all $p \in \mathrm{Proc}$ and $0 \leq i < |w|$, if $w_i \in \Sigma_{p,C}$, then $w_i = f_p(w[i]) = \overline{f}_p(\overline{s}_p(w[i]))$. As expected, such F-runs present good properties. Observe for instance that at each point of a given F-run, exactly one process has the token. Moreover, in a $(\mathcal{P}, F)$-fair F-run, at any point, any process is ensured to get the token eventually. This is formalized in the following lemma.

LEMMA 5.13. *Let $w \in \Sigma^\infty$ be a $(\mathcal{P}, F)$-fair F-run. Then, for all prefix v of w, for all process $r \in \mathrm{Proc}$, there is $v < v'a \leq w$ such that r has just received the token in a, i.e., $a \in \Sigma_{q,r}$ for some $q \in \mathrm{Proc}$.*

The proof of Lemma 5.13 uses the following claim:

CLAIM 5.14. *For all $v \leq w$, let p be the process having the token in v. Let $i \in \mathrm{ring}^{-1}(p)$ such that $\overline{s}_p(v) \in Q \times \{\mathrm{Tok}_i, \mathrm{Tok}'_i\}$. Let $j = (i \mod n) + 1$ and $q = \mathrm{ring}(j)$. Then, there exists $v < v'a \leq w$ such that $a \in \Sigma_{p,q}$.*

PROOF. Towards a contradiction, assume that $\mathrm{alph}(v^{-1}w) \cap \Sigma_{p,C} = \emptyset$. Then, for all $v < v' \leq w$, p still has the token in v', and $f_p(v') = \overline{f}_p(\overline{s}_p(v')) \in \Sigma_{p,C}$ is defined. Thus w is not $(\mathcal{P}, F)$-fair, which is a contradiction.

So let $a \in \Sigma_C$ such that $v < v'a \leq w$ and $\mathrm{alph}(v^{-1}v') \cap \Sigma_{p,C} = \emptyset$. By ($\delta 2$-$\delta 3$) we deduce that $\overline{s}_p(v') \in Q \times \{\mathrm{Tok}_i, \mathrm{Tok}'_i\}$. Since w is an F-run we deduce that $a = f_p(v')$. By definitions of f_p and $\overline{f}_p$, we obtain either $a \in Q \times \{i\}$ or $a \in \mathrm{Out}_p$. In the first case, we are done since $Q \times \{i\} \subseteq \Sigma_{p,q}$. Otherwise, $a \in \mathrm{Out}_p$ and $\overline{s}_p(v') \in Q \times \{\mathrm{Tok}_i\}$. Then, by ($\delta 2$), we obtain $\overline{s}_p(v'a) \in Q \times \{\mathrm{Tok}'_i\}$ and process p still has the token in $v'a$. Hence, as above, we can show that there is $b \in \Sigma_{p,C}$ such that $v'a < v'av''b \leq w$ and $\mathrm{alph}(v'') \cap \Sigma_{p,C} = \emptyset$. Here, ($\delta 3$) implies that $\overline{s}_p(v'av'') \in Q \times \{\mathrm{Tok}'_i\}$. Since w is an F-run we deduce that $b = f_p(v'av'') \in Q \times \{i\} \subseteq \Sigma_{p,q}$, which concludes the proof of the claim. □

Proof of Lemma 5.13. Since ring is surjective, we can apply the claim successively on all the processes having the token until we reach $q = \mathrm{ring}(i)$ when $r = \mathrm{ring}((i \mod n) + 1)$. Then, applying the claim a last time allows to conclude. □

We show now that F is a winning strategy for $(\mathcal{A}, L)$. Let $w \in \Sigma^\infty$ be a $(\mathcal{P}, F)$-fair F-run. We define precisely the imaginary run of the singleton architecture upon which the processes build their own strategy. For that, we use the following mappings defined by induction on the finite prefixes of w.

$$\mathrm{Fix} : \Sigma^* \to \Sigma^*$$
$$\mathrm{Loc}_p : \Sigma^* \to \mathrm{In}_p^* \text{ for all } p \in \mathrm{Proc}$$

We let $\mathrm{Fix}(\varepsilon) = \mathrm{Loc}_p(\varepsilon) = \varepsilon$ for all $p \in \mathrm{Proc}$, and, for $v \in \Sigma^*$ and $a \in \Sigma$,

(F1). if $a \in \mathrm{In}_p \cup \mathrm{Out}_p$, and if p has the token at v, then $\mathrm{Fix}(v \cdot a) = \mathrm{Fix}(v) \cdot a$, $\mathrm{Loc}_q(v \cdot a) = \mathrm{Loc}_q(v)$ for all $q \in \mathrm{Proc}$.

(F2). If $a \in \mathrm{In}_p$ and if p does not have the token at v, then $\mathrm{Fix}(v{\cdot}a) = \mathrm{Fix}(v)$, $\mathrm{Loc}_p(v{\cdot}a) = \mathrm{Loc}_p(v) \cdot a$, and $\mathrm{Loc}_q(v \cdot a) = \mathrm{Loc}_q(v)$ for all $q \neq p$.
(F3). If $a \in \Sigma_{p,q}$ then $\mathrm{Fix}(v{\cdot}a) = \mathrm{Fix}(v){\cdot}\mathrm{Loc}_q(v){\cdot}a$, $\mathrm{Loc}_q(v{\cdot}a) = \varepsilon$, and $\mathrm{Loc}_r(v{\cdot}a) = \mathrm{Loc}_r(v)$ for all $r \neq q$.

The mapping $\mathrm{Fix}(v)$ is increasing, hence we can define $w' = \bigsqcup_{v \leq w} \mathrm{Fix}(v)$. The word w' is then a reordering of the word w and $\pi_\Gamma(w')$ is the imaginary run of the singleton architecture.

When the history of actions in the distributed run is $v \in \Sigma^*$, $\mathrm{Fix}(v)$ is the current prefix of the imaginary run that is used to build the strategy of the process having the token, while $\mathrm{Loc}_p(v)$ is the sequence of input events process p has received since the last time it had the token. Hence, $\mathrm{Loc}_p(v)$ memorizes the sequence of events that process p will append to the imaginary run when it receives the token again. In particular, if p has the token in v, $\mathrm{Loc}_p(v) = \varepsilon$. Moreover, v is in the semi-commutation closure of the word formed by $\mathrm{Fix}(v)$ concatenated with $\Pi_{p \in \mathrm{Proc}} \mathrm{Loc}_p(v)$. These properties are formalized in Lemma 5.15.

For $v \in \Sigma^*$, we define $\gamma_v \in Q^Q$ by $\gamma_\varepsilon = \mathrm{id}$, and $\gamma_{v \cdot a} = \gamma_a \circ \gamma_v$. We establish the following invariants about the mappings Fix and Loc_p.

LEMMA 5.15. *Let $v \in \Sigma^*$ and let p be the process having the token in v. Then,*

(1) $\mathrm{Fix}(v) \cdot \prod_{r \in \mathrm{Proc}} \mathrm{Loc}_r(v) \Rightarrow_{\mathrm{SC}} v$.
(2) $\mathrm{Loc}_p(v) = \varepsilon$ *and* $\overline{s}_p(v) \in \{\delta(s_0, \pi_\Gamma(\mathrm{Fix}(v)))\} \times \mathrm{Tok}_p$,
(3) $\overline{s}_r(v) = (\delta_{\mathrm{Loc}_r(v)}, \mathrm{NTok})$ *for all* $r \neq p$.

The proof of this lemma is postponed to the end of the section. We show now that $\pi_\Gamma(w')$ is indeed an f-run of the singleton architecture, which in addition is $(\overline{\mathcal{P}}, f)$-fair.

PROPOSITION 5.16. *$\pi_\Gamma(w')$ is an f-run.*

PROOF. Let ua be a finite prefix of $\pi_\Gamma(w')$ with $a \in \mathrm{Out}_p$ for some $p \in \mathrm{Proc}$. Let $u'a$ be the finite prefix of w' with $\pi_\Gamma(u') = u$. Remind that $w' = \bigsqcup_{v \leq w} \mathrm{Fix}(v)$. Since $a \in \mathrm{Out}_p$, it has been added to Fix by **(F1)**. Hence, we find a prefix va of w such that $\mathrm{Fix}(v) = u'$ and $\mathrm{Fix}(va) = u'a$. Since w is an F-run, we have $a = f_p(v) = \overline{f}_p(\overline{s}_p(v))$. Since $a \in \mathrm{Out}_p$, we deduce from the definition of $\overline{f}_p$ that $a = \overline{f}(s)$ and $\overline{s}_p(v) = (s, \mathrm{Tok}_i)$ for some $i \in \mathrm{ring}^{-1}(p)$. By Lemma 5.15 (2), $s = \delta(s_0, \pi_\Gamma(\mathrm{Fix}(v))) = \delta(s_0, \pi_\Gamma(u')) = \delta(s_0, u)$. Hence, $a = \overline{f}(s) = f(u)$. ◻

PROPOSITION 5.17. *$\pi_\Gamma(w')$ is $(\overline{\mathcal{P}}, f)$-fair.*

PROOF. Let $p \in \mathrm{Proc}$ and $u \leq \pi_\Gamma(w')$ such that for all $u \leq u' \leq \pi_\Gamma(w')$ we have $f(u') \cap \mathrm{Out}_p \neq \emptyset$ (hence $f(u') \in \mathrm{Out}_p$). By construction, $w' = \bigsqcup_{v \leq w} \mathrm{Fix}(v)$ then there exists $v \leq w$ such that $u \leq \pi_\Gamma(\mathrm{Fix}(v)) = u'$. By Lemma 5.13, we may assume that p has just received the token in v. By Lemma 5.15, $\overline{s}_p(v) \in \{\delta(s_0, u')\} \times \mathrm{Tok}_p$. However, since $u \leq u' \leq \pi_\Gamma(w')$, we have $\overline{f}(\delta(s_0, u')) = f(u') \in \mathrm{Out}_p$. By ($\delta5$), we deduce that $\overline{s}_p(v) = (\delta(s_0, u'), \mathrm{Tok}_i)$ for some $i \in \mathrm{ring}^{-1}(p)$.

By Lemma 5.13, we know that p will eventually pass the token to process $q = \mathrm{ring}((i \bmod n) + 1)$. To do so, it has first to visit a state in $Q \times \{\mathrm{Tok}'_i\}$. So let $v''a$ be minimal such that $v < v''a \leq w$ and

$$\overline{s}_p(v'') \in Q \times \{\mathrm{Tok}_i\} \qquad \text{and} \qquad \overline{s}_p(v''a) = (s, \mathrm{Tok}'_i)\ .$$

By Lemma 5.15 we have $s = \delta(s_0, \pi_\Gamma(\mathrm{Fix}(v''a)))$. Since $u \leq u' = \pi_\Gamma(\mathrm{Fix}(v)) \leq u''a = \pi_\Gamma(\mathrm{Fix}(v''a)) \leq \pi_\Gamma(w')$ we have $\overline{f}(s) = f(u''a) \in \mathrm{Out}_p$. By ($\delta2$) we deduce that $a \notin \mathrm{In}_p$, i.e., $a \in \mathrm{Out}_p$. Therefore, $\mathrm{alph}(u^{-1}\pi_\Gamma(w')) \cap \mathrm{Out}_p \neq \emptyset$. ◻

We show now that the observable actual run $\pi_\Gamma(w)$ is in the semi-trace (or semi-commutation closure) of $\pi_\Gamma(w')$:

LEMMA 5.18. *We have* $w' \Rightarrow_{\mathrm{SC}} w$ *and* $\pi_\Gamma(w') \Rightarrow_{\mathrm{SC}} \pi_\Gamma(w)$.

PROOF. From Lemma 5.15, for all $v \le w$, we have $\mathrm{Fix}(v) \cdot \Pi_{p\in\mathrm{Proc}}\mathrm{Loc}_p(v) \Rightarrow_{\mathrm{SC}} v$ with some bijection $\sigma_v : \mathrm{Pos}(\mathrm{Fix}(v) \cdot \Pi_{p\in\mathrm{Proc}}\mathrm{Loc}_p(v)) \to \mathrm{Pos}(v)$. For $v \le v' \le w$, we have $\mathrm{Fix}(v) \le \mathrm{Fix}(v')$ and

$$\sigma_v(i) = \sigma_{v'}(i) \text{ for all } i \in \mathrm{Pos}(\mathrm{Fix}(v)) \tag{1}$$

by construction of the unique bijection associated with the rewriting $\Rightarrow_{\mathrm{SC}}$ (see Remark 4.5). We let $\sigma : \mathrm{Pos}(w') \to \mathrm{Pos}(w)$ be such that

$$\sigma(i) = \sigma_v(i) \text{ for all } v \le w \text{ such that } i \in \mathrm{Pos}(\mathrm{Fix}(v)).$$

By (1), σ is well defined. We first show that σ is a bijection.

Let $i, j \in \mathrm{Pos}(w')$ with $i < j$. Let $v \le w$ such that $j \in \mathrm{Pos}(\mathrm{Fix}(v))$, then $i \in \mathrm{Pos}(\mathrm{Fix}(v))$ and $\sigma(i) = \sigma_v(i) \ne \sigma_v(j) = \sigma(j)$ since σ_v is injective.

Let now $i \in \mathrm{Pos}(w)$ and $p \in \mathrm{Proc}$ such that $w(i) \in \Sigma_p$. By Lemma 5.13, let v be a prefix of w such that $w[i+1] \le v \le w$ and p has the token in v. By Lemma 5.15, we have $\mathrm{Loc}_p(v) = \varepsilon$ and $v' = \mathrm{Fix}(v) \cdot \Pi_{p\in\mathrm{Proc}}\mathrm{Loc}_p(v) \Rightarrow_{\mathrm{SC}} v$ with σ_v. By Definition 4.4, we have $v'(\sigma_v^{-1}(i)) = v(i) = w(i) \in \Sigma_p$. Since $\mathrm{Loc}_r(v) \in \mathrm{In}_r^*$ and $\mathrm{In}_r \cap \Sigma_p = \emptyset$ for all $r \ne p$, we deduce that $\sigma_v^{-1}(i) \in \mathrm{Pos}(\mathrm{Fix}(v)) \subseteq \mathrm{Pos}(w')$. Hence, $\sigma(\sigma_v^{-1}(i)) = \sigma_v(\sigma_v^{-1}(i)) = i$ and σ is surjective.

Then, let $i \in \mathrm{Pos}(w')$ and let v be a prefix of w such that $i \in \mathrm{Pos}(\mathrm{Fix}(v))$ (such a prefix exists by construction of w'). We have $\sigma(i) = \sigma_v(i) \in \mathrm{Pos}(v)$. Hence, $w(\sigma(i)) = v(\sigma(i)) = \mathrm{Fix}(v)(i) = w'(i)$.

Finally, let $i, j \in \mathrm{Pos}(w')$ with $(i, j) \in E_{w'}$. Let v be a prefix of w such that $i, j \in \mathrm{Pos}(\mathrm{Fix}(v))$. Then, $\sigma_v(i), \sigma_v(j) \in \mathrm{Pos}(v)$ and $(i, j) \in E_{\mathrm{Fix}(v)}$. Therefore, $\sigma(i) = \sigma_v(i) < \sigma_v(j) = \sigma(j)$.

We have shown that σ satisfies all requirements of Definition 4.4, hence we obtain $w' \Rightarrow_{\mathrm{SC}} w$. Projecting on Γ, we deduce easily $\pi_\Gamma(w') \Rightarrow_{\mathrm{SC}} \pi_\Gamma(w)$. □

We can now conclude the proof of Proposition 5.11. By Propositions 5.16 and 5.17, $\pi_\Gamma(w')$ is a $(\overline{\mathcal{P}}, f)$-fair f-run. Since f is a winning strategy for $(\mathcal{A}, \overline{\mathcal{P}}, L)$, $\pi_\Gamma(w') \in L$. Since L is an SC-closed specification, by Lemma 5.18, $\pi_\Gamma(w) \in L$.

To summarize, we have shown that any $(\mathcal{P}, F)$-fair F-run w implements the specification given by the language L. Therefore, the strategy F is winning for $(\mathcal{A}, L)$. Moreover, if f has finite memory, i.e., if Q is finite, then the automata $(\mathfrak{A}_p)_{p\in\mathrm{Proc}}$ are finite and the internal communication sets $(\Sigma_{p,q})_{(p,q)\in E}$ are also finite. □

Proof of Lemma 5.15. We show it by induction on the length of v.

If $v = \varepsilon$ then $\mathrm{Fix}(v) = \varepsilon$ and $\mathrm{Loc}_r(v) = \varepsilon$ for all $r \in \mathrm{Proc}$, so item 1 holds trivially. We have $p = \mathrm{ring}(1)$ and $\overline{s}_p(v) = s_0^p \in \{s_0\} \times \mathrm{Tok}_p$, hence 2 holds. Now, for $r \ne p$ we have $\overline{s}_r(v) = s_0^r = (\mathrm{id}, \mathrm{NTok})$ and 3 also holds.

Assume now that the lemma holds for some $v \in \Sigma^*$ and process p having the token in v. Let $a \in \Sigma$. We distinguish three cases.

(F1). If $a \in \mathrm{In}_p \cup \mathrm{Out}_p$. Then p still has the token in va. By definition, we have $\mathrm{Fix}(va) = \mathrm{Fix}(v)a$ and $\mathrm{Loc}_r(va) = \mathrm{Loc}_r(v)$ for all $r \in \mathrm{Proc}$.

1. Since $\mathrm{Loc}_p(v) = \varepsilon$ and $\mathrm{Loc}_r(v) \in \mathrm{In}_r^*$ for all $r \in \mathrm{Proc}$ we deduce using the definition of the semi-commutation relation and the induction hypothesis that

$$\begin{aligned}
\mathrm{Fix}(va) \cdot \textstyle\prod_{r\in\mathrm{Proc}} \mathrm{Loc}_r(va) &= \mathrm{Fix}(v) \cdot a \cdot (\textstyle\prod_{r\in\mathrm{Proc}} \mathrm{Loc}_r(v)) \\
&\Rightarrow_{\mathrm{SC}} \mathrm{Fix}(v) \cdot (\textstyle\prod_{r\in\mathrm{Proc}} \mathrm{Loc}_r(v)) \cdot a \\
&\Rightarrow_{\mathrm{SC}} v \cdot a
\end{aligned}$$

2. We have $\mathrm{Loc}_p(va) = \mathrm{Loc}_p(v) = \varepsilon$. Since $a \in \Sigma_p$, we have $\overline{s}_p(va) = \delta_p(\overline{s}_p(v), a)$. By ($\delta 2$-$\delta 3$) and using the induction hypothesis, we deduce that $\overline{s}_p(va) \in \{\delta(\delta(s_0, \pi_\Gamma(\mathrm{Fix}(v))), a)\} \times \mathrm{Tok}_p = \{\delta(s_0, \pi_\Gamma(\mathrm{Fix}(va)))\} \times \mathrm{Tok}_p$.
3. Let $r \in \mathrm{Proc}$ with $r \neq p$. We have $\mathrm{Loc}_r(va) = \mathrm{Loc}_r(v)$ and $a \notin \Sigma_r$. Hence, $\overline{s}_r(va) = \overline{s}_r(v) = (\gamma_{\mathrm{Loc}_r(v)}, \mathrm{NTok}) = (\gamma_{\mathrm{Loc}_r(va)}, \mathrm{NTok})$.

(F2). If $a \in \mathrm{In}_q$ for some $q \neq p$. Then p still has the token in va. By definition, we have $\mathrm{Fix}(va) = \mathrm{Fix}(v)$, $\mathrm{Loc}_q(va) = \mathrm{Loc}_q(v)a$ and $\mathrm{Loc}_r(va) = \mathrm{Loc}_r(v)$ for all $r \neq q$.
1. By definition of the semi-commutation relation, we have $a\mathrm{Loc}_r(v) \Rightarrow_{\mathrm{SC}} \mathrm{Loc}_r(v)a$ for all $r \neq q$. We deduce that

$$\begin{array}{rcl}\mathrm{Fix}(va) \cdot \prod_{r\in\mathrm{Proc}} \mathrm{Loc}_r(va) & = & \mathrm{Fix}(v) \cdot (\prod_{r\in\mathrm{Proc}} \mathrm{Loc}_r(va)) \\ & \Rightarrow_{\mathrm{SC}} & \mathrm{Fix}(v) \cdot (\prod_{r\in\mathrm{Proc}} \mathrm{Loc}_r(v)) \cdot a \\ & \Rightarrow_{\mathrm{SC}} & v \cdot a\end{array}$$

2. We have $\mathrm{Loc}_p(va) = \mathrm{Loc}_p(v) = \varepsilon$. Since $a \notin \Sigma_p$, we have $\overline{s}_p(va) = \overline{s}_p(v) \in \{\delta(s_0, \pi_\Gamma(\mathrm{Fix}(v)))\} \times \mathrm{Tok}_p$ by induction hypothesis. We conclude using $\mathrm{Fix}(va) = \mathrm{Fix}(v)$.
3. Let $r \in \mathrm{Proc}$ with $r \neq p, q$. We have $\mathrm{Loc}_r(va) = \mathrm{Loc}_r(v)$ and $a \notin \Sigma_r$. Hence, $\overline{s}_r(va) = \overline{s}_r(v) = (\gamma_{\mathrm{Loc}_r(v)}, \mathrm{NTok}) = (\gamma_{\mathrm{Loc}_r(va)}, \mathrm{NTok})$.
Now, $\mathrm{Loc}_q(va) = \mathrm{Loc}_q(v)a$ and $a \in \Sigma_q$. Hence, $\overline{s}_q(va) = \delta_q(\overline{s}_q(v), a)$. By ($\delta 1$) and the induction hypothesis we get $\delta_q(\overline{s}_q(v), a) = (\gamma_a \circ \gamma_{\mathrm{Loc}_q(v)}, \mathrm{NTok}) = (\gamma_{\mathrm{Loc}_q(va)}, \mathrm{NTok})$.

(F3). Finally, assume that $a \in \Sigma_{p,q}$ for some $q \neq p$. Then q has the token in va. By definition, we have $\mathrm{Fix}(va) = \mathrm{Fix}(v)\mathrm{Loc}_q(v)a$, $\mathrm{Loc}_q(va) = \varepsilon$ and $\mathrm{Loc}_r(va) = \mathrm{Loc}_r(v)$ for all $r \neq q$.
1. By definition of the semi-commutation relation, we have $\mathrm{Loc}_r(v)\mathrm{Loc}_q(v) \Rightarrow_{\mathrm{SC}} \mathrm{Loc}_q(v)\mathrm{Loc}_r(v)$ for all $r \neq q$. Moreover, $a\mathrm{Loc}_r(va) \Rightarrow_{\mathrm{SC}} \mathrm{Loc}_r(va)a$ for all $r \in \mathrm{Proc}$ since $\mathrm{Loc}_p(va) = \varepsilon = \mathrm{Loc}_q(va)$. Therefore,

$$\begin{array}{rcl}\mathrm{Fix}(va) \cdot \prod_{r\in\mathrm{Proc}} \mathrm{Loc}_r(va) & = & \mathrm{Fix}(v)\mathrm{Loc}_q(v) \cdot a \cdot (\prod_{r\in\mathrm{Proc}} \mathrm{Loc}_r(va)) \\ & \Rightarrow_{\mathrm{SC}} & \mathrm{Fix}(v)\mathrm{Loc}_q(v) \cdot (\prod_{r\in\mathrm{Proc}} \mathrm{Loc}_r(va)) \cdot a \\ & \Rightarrow_{\mathrm{SC}} & \mathrm{Fix}(v) \cdot (\prod_{r\in\mathrm{Proc}} \mathrm{Loc}_r(v)) \cdot a \\ & \Rightarrow_{\mathrm{SC}} & v \cdot a\end{array}$$

By induction hypothesis, we have $\overline{s}_p(v) \in \{s\} \times \mathrm{Tok}_p$ with $s = \delta(s_0, \pi_\Gamma(\mathrm{Fix}(v)))$. Since $a \in \Sigma_{p,q}$ and w is an F-run, we deduce from the definition of $\overline{f}_p$ that $\overline{s}_p(v) = (s, \mathrm{Tok}'_i)$ and $a = (s, i)$ for some $i \in \mathrm{ring}^{-1}(p)$ with $q = \mathrm{ring}((i \mod n) + 1)$.
2. We have $\mathrm{Loc}_q(va) = \varepsilon$ as desired. By induction hypothesis, we have $\overline{s}_q(v) = (\gamma_{\mathrm{Loc}_q(v)}, \mathrm{NTok})$. Hence, $\overline{s}_q(va) = \delta_q(\overline{s}_q(v), a) = \delta^q(\overline{s}_q(v), (s, i))$. By ($\delta 5$), we get $\overline{s}_q(va) \in \{\gamma_{\mathrm{Loc}_q(v)}(s)\} \times \mathrm{Tok}_q$. We can conclude since $\gamma_{\mathrm{Loc}_q(v)}(s) = \gamma_{\mathrm{Loc}_q(v)}(\delta(s_0, \pi_\Gamma(\mathrm{Fix}(v)))) = \delta(s_0, \pi_\Gamma(\mathrm{Fix}(v))\mathrm{Loc}_q(v))$ and $\pi_\Gamma(\mathrm{Fix}(va)) = \pi_\Gamma(\mathrm{Fix}(v))\mathrm{Loc}_q(v)$.
3. We have $\mathrm{Loc}_p(va) = \mathrm{Loc}_p(v) = \varepsilon$. Using ($\delta 4$) we obtain $\overline{s}_p(va) = \delta_p(\overline{s}_p(v), a) = \delta_p(\overline{s}_p(v), (s, i)) = (\mathrm{id}, \mathrm{NTok}) = (\gamma_{\mathrm{Loc}_p(va)}, \mathrm{NTok})$.
Now, let $r \in \mathrm{Proc}$ with $r \neq p, q$. We have $\mathrm{Loc}_r(va) = \mathrm{Loc}_r(v)$ and $a \notin \Sigma_r$. Hence, $\overline{s}_r(va) = \overline{s}_r(v) = (\gamma_{\mathrm{Loc}_r(v)}, \mathrm{NTok}) = (\gamma_{\mathrm{Loc}_r(va)}, \mathrm{NTok})$.

□

Propositions 5.7 and 5.11 allow to state that there is a winning distributed strategy for $(\mathcal{A}, L)$ if and only if there is a winning strategy for $(\overline{\mathcal{A}}, \overline{\mathcal{P}}, L)$, where $\mathcal{A}$ is strongly connected and L an SC-closed ω-regular language. This last problem is decidable by Theorem 5.1. This concludes the proof of Theorem 5.10.

Moreover, as already remarked, if there is a winning strategy for the singleton architecture and a regular specification, there is a winning strategy with finite memory. So, by Propositions 5.7 and 5.11, if there is a distributed winning strategy for the strongly connected architecture, then there is a winning strategy with finite memory.

6. CONCLUSION AND FUTURE WORK

In this article, we have defined a new setting for the synthesis problem for distributed asynchronous systems, and proved that it is decidable for an interesting subclass of architectures. We believe that using signals in asynchronous systems, and restricting to SC-closed specifications will help to overcome a lot of the common difficulties that usually lead to undecidability results.

Future work should generalize our decidability result to larger classes of architectures. The final aim is to obtain decidability in general, with a modular algorithm working on subarchitectures. With this objective, the next step is to solve the problem for acyclic architectures, including pipelines and trees.

Other challenging questions arise regarding the specification. We have shown that it is decidable to check whether a given ω-regular language is SC-closed, but it would also be interesting to know whether the largest SC-closed subset of an ω-regular language is still regular. Expressivity of $\mathsf{MSO}_{\mathsf{acc}}(\mathrm{SC})$ can also be investigated. More precisely, is any SC-closed ω-regular language definable in $\mathsf{MSO}_{\mathsf{acc}}(\mathrm{SC})$?

ACKNOWLEDGMENT

The authors are grateful to anonymous referees whose suggestions and comments helped to significantly improve the presentation of the article.

REFERENCES

ANUCHITANKUL, A. AND MANNA, Z. 1994. Realizability and synthesis of reactive modules. In *Proceedings of the 6th International Conference on Computer Aided Verification (CAV'94)*, D. L. Dill, Ed. Lecture Notes in Computer Science Series, vol. 818. Springer, Stanford, California, USA, 156–168.

ARNOLD, A., VINCENT, A., AND WALUKIEWICZ, I. 2003. Games for synthesis of controllers with partial observation. *Theoretical Computer Science 1,* 303, 7–34.

BAUDRU, N. 2009. Distributed asynchronous automata. In *Proceedings of the 20th International Conference on Concurrency Theory (CONCUR'09)*, M. Bravetti and G. Zavattaro, Eds. Lecture Notes in Computer Science Series, vol. 5710. Springer, 115–130.

BOUAJJANI, A., MUSCHOLL, A., AND TOUILI, T. 2001. Permutation rewriting and algorithmic verification. In *Proceedings of the 16th IEEE Annual Symposium on Logic in Computer Science (LICS'01)*. IEEE Press.

BÜCHI, J. R. AND LANDWEBER, L. H. 1969. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society 138*, 295–311.

CANO, A., GUAIANA, G., AND PIN, J.-E. 2011. Closure of regular languages under semi-commutations. http://liafa.jussieu.fr/ jep/PDF/PartialCommutationsRed.pdf.

CARTON, O., PERRIN, D., AND PIN, J.-E. 2008. Automata and semigroups recognizing infinite words. In *Logic and Automata: History and Perspectives [in Honor of Wolfgang Thomas]*, J. Flum, E. Grädel, and T. Wilke, Eds. Texts in Logic and Games Series, vol. 2. Amsterdam University Press, 133–168.

CÉCÉ, G., HÉAM, P.-C., AND MAINIER, Y. 2008. Efficiency of automata in semi-commutation verification techniques. *RAIRO, Theoretical Informatics and Applications 42,* 2, 197–215.

CHATAIN, T., GASTIN, P., AND SZNAJDER, N. 2009. Natural specifications yield decidability for distributed synthesis of asynchronous systems. In *Proceedings of the 35th International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM'09)*. Lecture Notes in Computer Science. Springer.

CHURCH, A. 1963. Logic, arithmetics, and automata. In *Proceedings of the International Congress of Mathematicians*. 23–35.

CLERBOUT, M. AND LATTEUX, M. 1987. Semi-commutations. *Information and Computation 73,* 1, 59–74.

DIEKERT, V. 1994. A partial trace semantics for Petri nets. *Theoretical Computer Science*.

DIEKERT, V., GASTIN, P., AND PETIT, A. 1995. Rational and recognizable complex trace languages. *Information and Computation 116,* 1, 134–153.

EHLERS, R. 2011. Unbeast: Symbolic bounded synthesis. In *Proceedings of the 17thInternational Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS11)*.

EMERSON, E. A. AND CLARKE, E. M. 1982. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming 2,* 3, 241–266.

FILIOT, E., JIN, N., AND RASKIN, J.-F. 2009. An antichain algorithm for LTL realizability. In *Proceedings of the 21st International Conference on Computer Aided Verification (CAV'09)*, A. Bouajjani and O. Maler, Eds. Lecture Notes in Computer Science Series, vol. 5643. Springer, 263–277.

FINKBEINER, B. AND SCHEWE, S. 2005. Uniform distributed synthesis. In *Proceedings of the 20th IEEE Annual Symposium on Logic in Computer Science (LICS'05)*. IEEE Computer Society Press, 321–330.

FINKBEINER, B. AND SCHEWE, S. 2006. Synthesis of asynchronous systems. In *Proceedings of the International Symposium on Logic-based Program Synthesis and Transformation (LOPSTR'06)*, G. Puebla, Ed. Lecture Notes in Computer Science Series, vol. 4407. Springer, 127–142.

GASTIN, P., LERMAN, B., AND ZEITOUN, M. 2004. Causal memory distributed games are decidable for series-parallel systems. In *Proceedings of the 24th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'04)*, K. Lodaya and M. Mahajan, Eds. Lecture Notes in Computer Science Series, vol. 3328. Springer, 275 – 286.

GASTIN, P., SZNAJDER, N., AND ZEITOUN, M. 2009. Distributed synthesis for well-connected architectures. *Formal Methods in System Design 34,* 3, 215–237.

HUNG, D. V. AND KNUTH, E. 1989. Semi-commutations and Petri nets. *Theoretical Computer Science 64*, 67–81.

JOBSTMANN, B. AND BLOEM, R. 2006. Optimizations for LTL synthesis. In *Proceedings of the 6th International Conference on Formal Methods in Computer-Aided Design (FMCAD'06)*. IEEE Press, 117–124.

KUPFERMAN, O. AND VARDI, M. Y. 1999. Church's problem revisited. *The Bulletin of Symbolic Logic 5,* 2, 245–263.

KUPFERMAN, O. AND VARDI, M. Y. 2000. μ-calculus synthesis. In *Proceedings of the 25th International Symposium on Mathematical Foundations of Computer Science (MFCS'00)*, M. Nielsen and B. Rovan, Eds. Lecture Notes in Computer Science Series, vol. 1893. Springer, 497–507.

KUPFERMAN, O. AND VARDI, M. Y. 2001. Synthesizing distributed systems. In *Proceedings of the 16th IEEE Annual Symposium on Logic in Computer Science (LICS'01)*, J. Y. Halpern, Ed. IEEE Computer Society Press.

KUPFERMAN, O. AND VARDI, M. Y. 2005. Safraless decision procedures. In *Proceedings of the 46th Annual IEEE Symposium on Foundations of Computer Science (FOCS'05)*. 531–540.

LYNCH, N. AND TUTTLE, M. R. 1989. An introduction to input/output automata. *CWI-Quarterly 2,* 3, 219–246.

MADHUSUDAN, P. AND THIAGARAJAN, P. S. 2001. Distributed control and synthesis for local specifications. In *Proceedings of the 28th International Colloquium on Automata, Languages and Programming (ICALP'01)*, F. Orejas, P. G. Spirakis, and J. van Leeuwen, Eds. Lecture Notes in Computer Science Series, vol. 2076. Springer, 396–407.

MADHUSUDAN, P. AND THIAGARAJAN, P. S. 2002. A decidable class of asynchronous distributed controllers. In *Proceedings of the 13th International Conference on Concurrency Theory (CONCUR'02)*, L. Brim, P. Jančar, M. Kretínský, and A. Kucera, Eds. Lecture Notes in Computer Science Series, vol. 2421. Springer, 145–160.

MADHUSUDAN, P., THIAGARAJAN, P. S., AND YANG, S. 2005. The MSO theory of connectedly communicating processes. In *Proceedings of the 25th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'05)*, R. Ramanujam and S. Sen, Eds. Lecture Notes in Computer Science Series, vol. 3821. Springer, 201–212.

MANNA, Z. AND WOLPER, P. 1984. Synthesis of communicating processes from temporal logic specifications. *ACM Transactions on Programming Languages and Systems 6,* 1, 68–93.

MAZURKIEWICZ, A. 1977. Concurrent program schemes and their interpretations. DAIMI report PB 78, Aarhus University.

MOHALIK, S. AND WALUKIEWICZ, I. 2003. Distributed games. In *Proceedings of the 23rd Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'03)*, P. K. Pandya and J. Radhakrishnan, Eds. Lecture Notes in Computer Science Series, vol. 2914. Springer, 338–351.

MUSCHOLL, A. 1996. *Über die Erkennbarkeit unendlicher Spuren*. Teubner-Texte zur Informatik Series, vol. 17. Teubner.

MUSCHOLL, A., WALUKIEWICZ, I., AND ZEITOUN, M. 2009. A look at the control of asynchronous automata. In *Perspectives in Concurrency Theory*, K. Lodaya, M. Mukund, and R. R., Eds. Universities Press, 356–371.

OCHMANSKI, E. 1989. Semi-commutation for P-T Systems. *The Bulletin of the European Association for Theoretical Computer Science 38*, 191–198.

PELED, D., WILKE, T., AND WOLPER, P. 1998. An algorithmic approach for checking closure properties of temporal logic specifications and omega-regular languages. *Theoretical Computer Science*, 183–203.

PERRIN, D. AND PIN, J.-E. 2004. *Infinite words*. Pure and Applied Mathematics Series, vol. 141. Elsevier.

PNUELI, A. AND ROSNER, R. 1989. On the synthesis of an asynchronous reactive module. In *Proceedings of the 16th International Colloquium on Automata, Languages and Programming (ICALP'89)*, G. Ausiello, M. Dezani-Ciancaglini, and S. R. D. Rocca, Eds. Lecture Notes in Computer Science Series, vol. 372. Springer, 652–671.

PNUELI, A. AND ROSNER, R. 1990. Distributed reactive systems are hard to synthesize. In *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science (FOCS'90)*. Vol. II. IEEE Computer Society Press, 746–757.

RABIN, M. O. 1972. *Automata on Infinite Objects and Church's Problem*. American Mathematical Society.

STEFĂNESCU, A., ESPARZA, J., AND MUSCHOLL, A. 2003. Synthesis of distributed algorithms using asynchronous automata. In *Proceedings of the 14th International Conference on Concurrency Theory (CONCUR'03)*, R. Amadio and D. Lugiez, Eds. Lecture Notes in Computer Science Series, vol. 2761. Springer, 27–41.

VARDI, M. Y. 1995. An automata-theoretic approach to fair realizability and synthesis. In *Proceedings of the 7th International Conference on Computer Aided Verification (CAV'95)*, P. Wolper, Ed. Lecture Notes in Computer Science Series, vol. 939. Springer, 267–278.

Received Month Year; revised Month Year; accepted Month Year