

From Asynchronous to Synchronous Specifications for Distributed Program Synthesis

Julien Bernet and David Janin

LaBRI, Université de Bordeaux I
351, cours de la Libération
33 405 Talence cedex FRANCE
{bernet|janin}@labri.fr

Abstract. Distributed games [7] allow expression of various distributed program synthesis problems. In such games, a team of Process players compete against a unique Environment player, each Process playing on its own arena, without explicit communications with its teammates.

The standard definition of distributed games allows some degree of *asynchrony* : the Environment can play only on part of the arenas, therefore concealing to some Process players that the play is going on. While this is convenient for modeling distributed problems (especially those that themselves make use of asynchrony), these games are by no means easy to manipulate, and the existing constructions are much more tedious.

In this paper, we provide a *uniform* reduction of any finite state distributed game, synchronous or asynchronous, to a *synchronous* one with the same number of players. This reduction is shown to be correct in the sense that it preserves the existence of *finite state* distributed winning strategies for the team of Processes. It is uniform in the sense that it is designed for arbitrary input distributed game *without any prior knowledge* about its satisfiability. The size of the resulting synchronous game is also linear in the size of the original asynchronous one and, moreover, there is no blowup of the size of the winning strategies. Additional expected preservation properties (e.g. information flows) are studied. Surprisingly, it seems that no such reduction exists for arbitrary (infinite state) strategies.

Introduction

Asynchrony in a distributed environment can be defined in an abstract way as the ability of some processes or agents to perform some action in the distributed systems while other processes or agents not only perform no action but are even not aware that they are staying idle. In other words, in a model where every event in a process (e.g. a local action or an incoming or outgoing message) is triggered by the tick of a local clock, asynchrony occurs when there is no global clock on which local clocks are synchronized.

However, if the memory of a distributed system has finitely many possible states, it is commonly understood that synchronous and asynchronous behaviors are, to some extent, equivalent. In fact, the number of relevant asynchronous

events between any two synchronous events can be bounded by the number of global states. More precisely, the (presumably asynchronous) distributed system can be modeled (at a more abstract level) by an equivalent synchronous one.

For instance, within a distributed system with an asynchronous message passing mechanism, if the number of messages that have been sent but not yet received is bounded, there cannot be an unbounded number of (relevant) asynchronous events between any two synchronizations.

Of course asynchrony makes a difference at the implementation level. For instance communication protocols do have to cope with asynchrony to be correct. However, at some more abstract level, provided the global memory of a system is finite, it seems that, in some sense, asynchronous and synchronous systems have the same computational power.

The main objective of this paper is to substantiate this intuition. More precisely, in the setting of distributed program synthesis problem, we aim at proving that (1) when the distributed environment is finite state and (2) when one considers only finite state programs, any synchronous or asynchronous distributed program synthesis problem is equivalent to a synchronous distributed program synthesis problem.

An immediate difficulty for this task is however to find the appropriate formalism.

In fact, there exists plenty of models of distributed systems and behaviors; consider for instance the diversity of models of communications between processes or agents: from communication via global shared memory mechanisms to many types of message passing systems. For distributed program synthesis specification, where one not only aims at modeling a single behavior but a full class of behaviors that may, or may not, fulfill the design objectives, there may be even more formalisms. See for instance Pnueli and Rosner's notion of distributed architecture [10, 4] or Wonham et al.'s distributed control theory [5, 6].

In this paper, we choose to study the relationship between synchronous and asynchronous distributed synthesis problems in a fairly abstract though general model: the model of distributed games [7].

These games, designed after Peterson and Reif's partial information games [9], rely on an explicit modeling of the local information - projection of the global state - available to every agent (or process) for guiding its own behaviors. As a matter of fact, no explicit communication is modeled, henceforth no commitment has to be done in favor of such or such communication mechanism. Moreover, most distributed synthesis problem can be encoded and solved within distributed games problem [2, 7].

In a distributed game, a team of Process players compete against a unique Environment player, each Process playing on its own arena - with its own projection of the global state of the system - while Environment player has a complete knowledge of the global state.

Communications are implicitly modeled as follows. In a distributed game definition, one may restrict Environment player. From a given global state - partially known by every Process player - this may force the Environment player

to reach some global states with more informative local projections than other global states.

Asynchrony, that induces an extra layer of partial information - absence of global clock - is modeled as follows: in a distributed game, the Environment may play only on part of the arenas, therefore concealing to some Process players that the play is going on elsewhere.

In this paper we prove that, as far as finite systems and programs are concerned, asynchrony does not yield extra expressive power.

More precisely, we provide a reduction of any asynchronous finite state distributed game to a *synchronous* one with the same number of players. This reduction is shown to be correct in the sense that it preserves the existence of *finite state* distributed winning strategies for the team of Processes. The size of the resulting synchronous game is also linear in the size of the original asynchronous one and, moreover, there is no blowup of the size of the winning strategies. Additional expected preservation properties (e.g. information flows) are also studied.

For strategies with unbounded memory, it is conjectured that there is no such a reduction. Still; one may ask whether such a result could be extended to more general classes of programs with infinite memory such as, for instance, pushdown strategies. This remains an open problem. However, even decidability questions are yet not settled for these more general strategies.

1 Notations

For any finite alphabet A , a *word* over A is a function $w : \mathbb{N} \rightarrow A$ whose domain is an initial segment of \mathbb{N} . The only word such with empty domain is called the empty word, and denoted by ϵ .

The following notations are used : A^* is the set of finite words (*i.e.* whose domain is finite) over A , A^ω is the set of infinite words over A , $A^\infty = A^* \cup A^\omega$ is the set of finite and infinite words over A , and $A^+ = A^* - \{\epsilon\}$ is the set of finite non-empty words over A .

For any finite word w , its length $|w|$ is the cardinal of its domain. For any words $u \in A^*$ and $v \in A^\omega$, the word $u \cdot v$ is the concatenation of words u and v . For any integer k , the word u^k is built by concatenating k copies of u . When $u \neq \epsilon$, the infinite word u^ω is the infinite concatenation of u with itself.

These operations are extended to the languages. Given $X \subseteq A^*$ and $Y \subseteq A^\infty$, we use the notations $X \cdot Y = \{u \cdot v : u \in X, v \in Y\}$, $X^0 = \{\epsilon\}$, $X^k = \{w \cdot u : w \in X^{k-1}, u \in X\}$ for any $k > 0$, $X^* = \bigcup_{k \geq 0} X^k$ and, when $\epsilon \notin X$, X^ω for the set of all words that can be defined as an infinite product of words of X . The following additional notations are also used : $X + Y = X \cup Y$, $X - Y = X \setminus Y$ and $X^? = \{\epsilon\} \cup X$.

Given n sets X_1, \dots, X_n , consider their direct product $X = X_1 \times \dots \times X_n$. For every set $I = \{i_1, \dots, i_k\} \subseteq \{1, \dots, n\}$, consider the natural projection $\pi_I : X \rightarrow X_{i_1} \times \dots \times X_{i_k}$ defined by $\pi_I((x_1, \dots, x_k) = (x_{i_1}, \dots, x_{i_k})$. For any $x \in X$, we also use the more convenient notation $x[I] = \pi_I(x)$.

Following the same idea, for any set $Y \subseteq X$, denote by $Y[I]$ the set $\pi_I(Y)$. If $R \subseteq X^m$ is a m -ary relation over X , we should also write use $R[I] = \{(x_1[I], \dots, x_m[I]) : (x_1, \dots, x_m) \in R\}$. When I in an interval of integers of the form $[i, j]$ we use the notations $x[i, j], Y[i, j], R[i, j]$. Whenever I is reduced to a single integer i , these notations simplify to $x[i], Y[i], R[i]$.

Moreover, these notations are extended to words and languages : for any word $w = x_0 \cdot x_1 \cdots \in X^\infty$, then $w[I] = x_0[I] \cdot x_1[I] \cdots$.

2 Distributed games

A n -process *distributed arena* is a game arena built from some product of n local standard game arena.

Definition. Given n arenas $G_i = \langle P_i, E_i, T_{P,i}, T_{E,i} \rangle$ for $i \in [1, n]$ with disjoint sets of Process position P_i , sets of Environment position E_i , sets of Process moves $T_{P,i} \subseteq P_i \times E_i$ and sets of Environment moves $T_{E,i} \subseteq E_i \times P_i$, an n -Process *distributed game arena* built from the local game arenas $\{G_i\}_{i \in [1, n]}$ is any game arena $G = \langle P, E, T_P, T_E \rangle$ such that:

- *Environment positions* : $E = \prod_{i \in [1, n]} E_i$,
- *Processes positions* : $P = \prod_{i \in [1, n]} (E_i \cup P_i) - \prod_{i \in [1, n]} E_i$,
- *Processes moves* : T_P is the set of all pairs $(p, e) \in (P \times E)$ such that, for $i \in [1, n]$:
 - **either** $p[i] \in P_i$ and $(p[i], e[i]) \in T_{P,i}$ (Process i is *active in* p),
 - **or** $p[i] \in E_i$ and $p[i] = e[i]$ (Process i is *inactive in* p),
- and *Environment moves* : T_E is **some subset** of the set of all pairs $(e, p) \in (E \times P)$ such that, for $i \in [1, n]$:
 - **either** $p[i] \in P_i$ and $(e[i], p[i]) \in T_{P,i}$ (Environment activates Process i),
 - **or** $p[i] \in E_i$ and $p[i] = e[i]$ (Environment keeps Process i inactive).

When the set T_E of Environment moves is maximal, we call such an arena the *free asynchronous product* of arenas $\{G_i\}_{i \in [1, n]}$ and it is written $G_1 \otimes G_2 \otimes \cdots \otimes G_n$.

Remark. The essential idea behind this definition is to get a definition of a multiplayer game in which a team of Processes compete against a unique Environment to achieve some infinitary goal. The following point is important : this definition allows the Environment to play only on a subset of the arenas, therefore hiding that the play is going on to the Processes on which arenas it does not play. This will be referred in the following as *asynchronous move*, and allows to encode neatly many distributed synthesis problems from the literature, such as [10, 4] or [5].

In distributed games, asynchrony occurs when Environment player decides to keep one or more Process players inactive. A synchronous distributed arena can thus be defined as follows.

Definition. An n -process distributed arena $G = \langle P, E, T_P, T_E \rangle$ is a *synchronous distributed arena* when $T_E \subseteq E \times \prod_{i \in [1, n]} P[i]$.

Since a distributed arena is built upon n simple arenas, we need a definition to speak about its local components:

Definition. Given a distributed arena $G = \langle P, E, T_P, T_E \rangle$ as above, given a non empty set $I \subseteq [1, \dots, n]$ we define the *canonical projection* $G[I]$ of G on I

as the arena $G[I] = \langle P', E', T'_P, T'_E \rangle$ given by: $P' = P[I] - E[I]$ (possibly smaller than $P[I]$!), $E' = E[I]$, $T'_P = T_P[I] \cap (P' \times E')$, and $T'_E = T_E[I] \cap (E' \times P')$.

A distributed game arena is, at first sight, a particular case of standard discrete and turn base two player game arena. Standard notions of plays and strategies are still defined. However, in order to avoid confusion with what may happen in the local arena a distributed game is build upon, we shall speak now of a *global play* and *global and local strategy*. Partial information is then captured by means of the notion of *local view of play* and *distributed strategy*.

Definition. Given an n -process distributed arena G , a *global play* from an initial position $e \in E$ is defined as a path in G (seen as a bipartite graph) emanating from position e that is built alternatively by the Environment player and the Process team.

More precisely, from a current position $x \in P \cup E$, either $x \in E$ and it is Environment player turn to play by choosing some position $y \in P$ such that $(x, y) \in T_E$ or $x \in P$ and it is Process team turn to play by choosing some position $y \in E$ such that $(x, y) \in T_P$.

Accordingly, a *global strategy* for the Process team is a partial function $\sigma : (E.P)^+ \rightarrow E$ such that for every play of the form $w.p \in \text{dom}(\sigma)$ with $w \in E.(P.E)^*$ and $p \in P$ one has $(p, \sigma(w.p)) \in T_P$.

A play w is said *compatible* with strategy σ when, for every integer $n \geq 0$ such that $w[n] \in P$ one has $w[n+1] = \sigma(w[0, n])$ where $w[0, n]$ is the prefix of w of length $n+1$.

Definition. A game $G = \langle P, E, T_E, T_P, e_0, \mathcal{W} \rangle$ is a game arena $\langle P, E, T_E, T_P \rangle$ equipped with an extra initial position $e_0 \in E$ and a distinguished set $\mathcal{W} \subseteq (E+P)^\omega$ called infinitary condition for the Process team.

We say that global strategy σ is a *winning strategy* for the Process team from position $e_0 \in E$ with condition \mathcal{W} when every maximal plays starting in e_0 and compatible with strategy σ is either finite and ends in an environment position or is infinite and belongs to \mathcal{W} .

A *strategy with finite memory* for the Process team is given as a tuple $\mathcal{M} = \langle M, m_0, \mu : M \times (P \cup E) \rightarrow M, h : M \times P \rightarrow E \rangle$, where M is a finite set of *memory states*, m_0 is the *initial memory*, μ is the *update function*, and h is the *hint function*. The induced strategy $\sigma_{\mathcal{M}} : (E.P)^+ \rightarrow E$ is then defined, for any play $w.p \in (E.P)^+$, by $\sigma_{\mathcal{M}} = h(\mu^*(m_0, w), p)$ (where μ^* is defined by $\mu^*(m, \epsilon) = m$, and $\mu^*(m, w.x) = \mu(\mu^*(m, w), x)$ for every $m \in M$, $w \in (E \cup P)^*$, $x \in (E \cup P)$).

In distributed games, it is intended that, within the Process team, every process has only a partial view of a global play. Not only every process only sees its own projection of every global positions, but, when idle, a process is even not aware that the play is going on. This intention is formally defined as follows.

Definition. The *local view* Process i has of a global play in a distributed game G is given by the map $\text{view}_i : (E.P)^* \cdot E^? \rightarrow (E_i.P_i)^* \cdot E_i^?$ defined in the following way:

- $\text{view}_i(\epsilon) = \epsilon$
- $\text{view}_i(x) = x[i]$

$$- \text{view}_i(w \cdot x \cdot y) = \begin{cases} \text{view}_i(w \cdot x) & \text{if } x[i] = y[i] \\ \text{view}_i(w \cdot x) \cdot y[i] & \text{otherwise.} \end{cases}$$

A play $w \in (E \cdot P)^+$ is said to be active for Process i when w ends in a position $p \in P$ such that $p[i] \in P[i]$.

Remark. Observe that in a synchronous distributed arena, as expected, for every play $w \in (E \cdot P)^* \cdot E^?$ one has $\text{view}_i(w) = w[i]$, i.e. the local view of a global play is just the projection of this play.

Definition. A global strategy σ for the Process team is a *distributed strategy* when, for every $i \in [1, n]$, there is a process strategy $\sigma_i : (E[i] \cdot P[i])^+ \rightarrow E[i]$ in the local game $G[i]$, from now on called *local strategy* for Process i , such that, for any play of the form $w \cdot p \in (E \cdot P)^+$, given the set $I \subseteq \{1, \dots, n\}$ of active processes in the global Processes position p , $\sigma(w \cdot p) = e$ if and only if

$$\begin{aligned} - e[i] &= \sigma_i(\text{view}_i(w) \cdot p[i]) \text{ for } i \in I \\ - e[i] &= p[i] \text{ for } i \in \{1, \dots, n\} - I \end{aligned}$$

In this case, we write $\sigma_1 \otimes \sigma_2 \otimes \dots \otimes \sigma_n$ for the distributed strategy σ .

Remark. Observe that when G is synchronous, the distributed strategy $\sigma = \sigma_1 \otimes \dots \otimes \sigma_n$ can simply be defined for any global play $w \in (E \cdot P)^*$ by :

$$\sigma(w) = (\sigma_1(w[1]), \dots, \sigma_n(w[n]))$$

Remark. Global strategies are not always distributed. In particular, there are distributed games where Process team has a winning global strategy, but no winning distributed strategy. For more details, the reader can refer to [7].

3 From asynchronous game to synchronous game

We prove here that every (asynchronous) distributed game is equivalent in some sense to a synchronous distributed game. More precisely:

Theorem 1. *There exists a mapping that maps every distributed game G to exists a distributed game \tilde{G} such that the Process team have a distributed winning strategy with finite memory in G if and only if they have one in \tilde{G} . Moreover, game \tilde{G} has the same number of Process players as game G with only a linear increase of number of positions. Moreover, this mapping is defined uniformly on distributed games, be them winning for the process team or not.*

The remaining of this section is devoted to the proof of this result.

Let $G = \langle P, E, T, e_0, \mathcal{W} \rangle$ be a n -processes distributed game. First, we are going to describe the synchronous game \tilde{G} , then we will show that it is equivalent to G in terms of distributed strategies with finite memory.

For any set E_i , define \widehat{E}_i as an equipotent set, such that $E_i \cap \widehat{E}_i = \emptyset$; for any $e \in E_i$, denote by \widehat{e}_i its image in \widehat{E}_i . Let $\widehat{E} = \prod_{i \in \{1, \dots, n\}} \widehat{E}_i$ and let

$\widehat{P} = \prod_{i \in \{1, \dots, n\}} \widehat{P}_i$ (i.e. we restrict to relevant process positions in a synchronous game: process positions where every process is active).

Consider $\widetilde{G} = \langle \widetilde{P}, \widetilde{E}, \widetilde{T}, e_0, \widetilde{W} \rangle$, whose positions are:

$$\widetilde{P}_i = P_i \cup \widehat{E}_i \quad ; \quad \widetilde{E}_i = E_i \quad (\text{for all } i \in \{1, \dots, n\})$$

For any position $x \in P \cup E$, denote by \widehat{x} the position of \widetilde{P} obtained by replacing in e each component from E_i with their image in \widehat{E}_i , i.e. :

$$\widehat{x}[i] = \begin{cases} x[i] & \text{if } x[i] \in E_i \\ x[i] & \text{if } x[i] \in P_i \end{cases}$$

The function that maps any $x \in P \cup E$ to its image \widehat{x} in \widetilde{P} is trivially a bijection. The moves of \widetilde{G} are defined as follows:

$$\begin{aligned} \widetilde{T}_i^P &= T_i^P \cup \{(\widehat{e}, e) : e \in E_i\} \\ \widetilde{T}^E &= \{(e, \widehat{p}) \in \widetilde{E} \times \widetilde{P} \mid (e, p) \in T^E\} \\ &\quad \cup \{(e, \widehat{e}) : e \in E\} \end{aligned}$$

The function $\text{cancel}(\widetilde{E} \cdot \widetilde{P})^* \rightarrow (E \cdot P)^*$ erases any asynchronous move from a global play: $\text{cancel}(\epsilon) = \epsilon$, $\text{cancel}(w \cdot e \cdot p) = \text{cancel}(w) \cdot e \cdot p$, and $\text{cancel}(w \cdot e \cdot \widehat{e}) = \text{cancel}(w)$ (where $p \in P$, $e \in E$).

This function is generalized to infinite words by: $\text{cancel}(x_0 \cdot x_1 \cdot \dots) = \lim_{i \rightarrow \infty} \text{cancel}(x_0 \cdot \dots \cdot x_i)$ (it is a converging sequence, in the sense of the prefix topology over words, as defined for instance in [8]).

The winning condition of \widetilde{G} is then defined as follows:

$$\widetilde{W} = \text{cancel}^{-1}(\mathcal{W}) \cup (\widetilde{E} \cdot \widetilde{P})^* \cdot (E \cdot \widehat{E})^\omega$$

Remark. The underlying bipartite graph of G is embedded into the one from \widetilde{G} . The graph of the arena of \widetilde{G} is actually nothing more than the subgraph induced by this embedding, where on each position of the environment a loop of size 2 has been added, corresponding to a totally asynchronous move. Moreover, the winning condition \widetilde{W} is not much more complicated than \mathcal{W} : amongst the usual infinitary winning conditions (reachability, safety, parity, Muller, etc. ...), only the safety condition is not preserved by this construction.

Lemma 1 (From asynchronous to synchronous). *For any distributed and winning strategy for the Process team in G , there is a winning distributed strategy for the Process team in \widetilde{G} .*

The idea is actually to copy this strategy on \widetilde{G} , ensuring in the process that the Process players do not take the asynchronous moves played onto their arena into account.

For any $i \in \{1, \dots, n\}$, let us define a function $\text{cancel}_i : (\widetilde{E}_i \cdot \widetilde{P}_i)^* \rightarrow (E_i \cdot P_i)^*$ that erases the local asynchronous moves: $\text{cancel}_i(\epsilon) = \epsilon$, $\text{cancel}_i(w \cdot e \cdot p) =$

$\text{cancel}_i(w) \cdot e \cdot p$, and $\text{cancel}_i(w \cdot e \cdot \hat{e}) = \text{cancel}_i(w)$ for any $e \in E_i$, $p \in P_i$, and $w \in (\tilde{E}_i \cdot \tilde{P}_i)^*$.

Consider a winning distributed strategy $\sigma = \sigma_1 \otimes \cdots \otimes \sigma_n$ over G . The distributed strategy $\tilde{\sigma} = \tilde{\sigma}_1 \otimes \cdots \otimes \tilde{\sigma}_n$ is defined for any $i \in \{1, \dots, n\}$, for any local play $w \in \tilde{E}_i \cdot (\tilde{P}_i \cdot \tilde{E}_i)^*$, and for any positions $p \in P_i$, $e \in E_i$ by:

$$\tilde{\sigma}_i(w \cdot p) = \begin{cases} \sigma_i(\text{cancel}_i(w \cdot p)) & \text{if } \text{cancel}_i(w \cdot p) \in \text{Dom}(\sigma_i) \\ \text{undetermined} & \text{otherwise.} \end{cases}$$

$$\tilde{\sigma}_i(w \cdot \hat{e}) = e$$

It is clear that for any $i \in \{1, \dots, n\}$, the following diagram commutes:

$$\begin{array}{ccc} (\tilde{E} \cdot \tilde{P})^* & \xrightarrow{\text{cancel}} & (E \cdot P)^* \\ \pi_i \downarrow & & \downarrow \text{view}_i \\ (\tilde{E}_i \cdot \tilde{P}_i)^* & \xrightarrow{\text{cancel}_i} & (E_i \cdot P_i)^* \end{array}$$

Therefore, for any global play $w \in \text{Dom}(\tilde{\sigma})$, and for any Process $i \in \{1, \dots, n\}$ such that $w[i] \in P_i$, we have:

$$\begin{aligned} \tilde{\sigma}_i(w[i]) &= \sigma_i(\text{cancel}_i(w[i])) \\ &= \sigma_i(\text{view}_i \text{cancel}(w)) \end{aligned}$$

For any infinite play w in \tilde{G} which is consistent with $\tilde{\sigma}$, the corresponding play $\text{cancel}(w)$ in G is consistent with σ , hence belongs to \mathcal{W} when infinite. Then $w \in \tilde{\mathcal{W}}$ comes directly. The strategy $\tilde{\sigma}$ is therefore winning over G .

Lemma 2 (From synchronous to asynchronous). *For any finite state winning distributed strategy for the Process team over \tilde{G} there exists a finite state winning distributed strategy for the Process team over G with a memory of the same size.*

The problem in proving this lemma is that we will obviously have to cope with any local strategy over \tilde{G}_i ($i \in \{1, \dots, n\}$) has the ability to somehow *count* the asynchronous moves, therefore getting additional information on the global play comparing to a local strategy over G .

The answer consists in showing that this counting is in any case useless, since each time the Environment can choose to play a totally asynchronous move. A Process i has therefore no interest in counting the local asynchronous moves, since he does not know whether they are *true* asynchronous moves (corresponding to asynchronous moves in G) or totally asynchronous ones.

The proof technique we use consists in *saturating* the memory of any distributed strategy over G , building in the process a distributed strategy that behaves like the one over G would do if the Environment played a large number of totally asynchronous moves each time he has to play.

Suppose the following distributed strategy with finite memory is given: $\tilde{\sigma} = \tilde{\sigma}_1 \otimes \cdots \otimes \tilde{\sigma}_n$, and suppose it is winning over \tilde{G} with the following memories:

$$\tilde{\mathcal{M}}_i = \langle \tilde{M}_i, \tilde{m}_{0,i} \in \tilde{M}_i, \tilde{\mu}_i : \tilde{M}_i \times (\tilde{P}_i \cup \tilde{E}_i) \rightarrow \tilde{M}_i, \tilde{h}_i : \tilde{M}_i \times \tilde{P}_i \rightarrow \tilde{E}_i \rangle$$

(for $(i \in \{1, \dots, n\})$).

Since there are finitely many local strategies, each of them with finite memory, pumping lemma arguments show that: there exists an integer L such that for any Process $i \in \{1, \dots, n\}$, for any memory element m in \tilde{M}_i , for any position $e \in \tilde{E}_i$, the following holds:

$$\mu_i^*(m, (e \cdot \hat{e})^L) = \mu_i^*(m, (e \cdot \hat{e})^{k \cdot L}) \quad \text{for any integer } k > 0 \quad (1)$$

Now, consider the distributed strategy over G $\sigma = \sigma_1 \otimes \cdots \otimes \sigma_n$ with finite memory $\mathcal{M}_i = \langle M_i, m_{0,i}, \mu_i, h_i \rangle$, where $M_i = \tilde{M}_i, m_{0,i} = \tilde{m}_{0,i}, h_i = \tilde{h}_i$, and:

$$\begin{aligned} \mu_i(m, e) &= \tilde{\mu}_i^*(m, e \cdot (\hat{e} \cdot e)^{2 \cdot L - 1}) \\ \mu_i(m, p) &= \tilde{\mu}_i(m, p) \end{aligned}$$

for $e \in E_i$ and $p \in P_i$.

We are going to show that σ is winning over G . First of all, define the function $\text{fill} : (E \cdot P)^* \rightarrow (\tilde{E} \cdot \tilde{P})^*$ as follows:

$$\text{fill}(e_0 \cdot p_0 \cdot e_1 \cdot p_1 \cdots p_n) = e_0 \cdot (\hat{e}_0 \cdot e_0)^{2 \cdot L - 1} \cdot \hat{p}_0 \cdot e_1 \cdot (\hat{e}_1 \cdot e_1)^{2 \cdot L - 1} \cdot \hat{p}_1 \cdots \hat{p}_n$$

fill can be generalized to infinite words in the same fashion than cancel .

Remark. fill is clearly a map from the plays where the Processes have to play in G to the plays of \tilde{G} . It is moreover easy to figure out that $\text{cancel} \circ \text{fill}$, restricted to the plays of G , is the identity function, and that therefore $\text{fill}(w) \in \tilde{\mathcal{W}}$ implies $w \in \mathcal{W}$.

Last, the following fact tells that σ behaves over G exactly like $\tilde{\sigma}$ does over \tilde{G} if the Environment plays $2 \cdot L - 1$ totally asynchronous moves each time it has to play.

Fact 21 *For any infinite play w in G consistent with σ , the play $\text{fill}(w)$ is consistent with $\tilde{\sigma}$.*

Knowing that $\tilde{\sigma}$ is winning, and using remark above, we conclude that σ is winning.

Remark. σ is not more complex than $\tilde{\sigma}$; it actually uses a memory of exactly the same size.

4 Synchronizing linear game

It is known that, in general, checking the existence of a winning distributed strategy for the Process team is undecidable, even in the case there are only two Process players with reachability ($\mathcal{W} = \emptyset$) or safety ($\mathcal{W} = (E + P)^\omega$) winning

condition [3]. However, when the information flows satisfies some linearity condition described below, the problem becomes decidable though non elementary [9].

In view of these properties, it occurs that our reduction of asynchronous distributed games to equivalent synchronous one is not that satisfactory. In fact, by introducing global non deterministic Environment moves everywhere, the linearity of the information flows in game G is lost in game \tilde{G} .

We provide in this section a modification of our construction that do preserve such a linearity property (built upon the notion of i -sequentiality in [7]).

Definition. Given an n -Process distributed game $G = \langle P, E, T_P, T_E, e_0, \mathcal{W} \rangle$, we say that game G is a *distributed linear game* when for every $i \in [1, n]$, for every Environment positions e and f , for every Process team positions p and $q \in P$ such that $(e, p) \in T_E$ and $(f, q) \in T_E$:

If $e[1, i] = f[1, i]$ and if $p[i] = q[i] \in P[i]$ or $p[i] \in E[i]$ or $q[i] \in E[i]$ then $p[1, i] = q[1, i]$.

This (local) linearity property first ensures that before every Environment moves, if a Process player i *knows* (in the epistemic sense) not only his own position $e[i]$ but also the position $e[1, i - 1]$ of positions of every Process player with *lower* index, then this remain the case after any (synchronous or asynchronous) Environment move.

Moreover, since Process players knows each other strategies, this properties also ensures that, from a given starting position, given a fixed distributed strategy, every Process player knows (again in the epistemic sense), at any time he is active during a play, the position of every Process player of smaller index.

The next definition gives a construction on distributed games that, when applied to linear games, can be seen as a normalization process shifting from implicit knowledge to explicit knowledge.

Definition. Let $G = \langle P, E, T_P, T_E, e_0, \mathcal{W} \rangle$ be an n -process distributed game, and let $lin(G) = \langle P', E', T'_P, T'_E, e'_0, \mathcal{W}' \rangle$, called the *linearization* of G , be the game defined from game G as follows:

1. for every $i \in [1, n]$:
 - (a) $P'_i = P[1, i] - E[1, i] + \{\perp_i\}$,
 - (b) $E'_i = E[1, i]$,
 - (c) $T'_{P,i} = T_P[1, i] \cap (P'_i \times E'_i)$,
2. and, for every $e \in E' = \prod_{i \in [1, n]} E'_i$:
 - (a) either position e is *coherent w.r.t. game G* in the sense that for every $i \in [1, n]$ one has $e[i] = (e[n])[1, i]$, then we put $(e, p) \in T'_E$ for every $p \in P'$ such that $\forall i \in [1, n], (e[i], p[i]) \in T_E[1, i]$,
 - (b) or position e is *incoherent* then we put $(e, \perp) \in T'_E$ with $\perp = (\perp_1, \dots, \perp_n)$.
3. $e'_0 = (e_0[1], e_0[1, 2], \dots, e_0[1, n - 1], e_0[1, n])$,
4. and $\mathcal{W}' = \{w \in (P' + E')^\omega : w[n] \in \mathcal{W}\}$.

Remark. Observe that, in game $lin(G)$ any time the Process team reach an incoherent position e , Environment player moves to position \perp where the Process team loses. It follows that relevant positions in game $lin(G)$ (positions

where the Process team will play to win) are only coherent positions that is to say position $x \in E' + P'$ such that, given $y = x[n] \in P + E$, one has $x = (y[1], y[1, 2], \dots, y[1, n-1], y[1, n])$. In other words, in every global coherent position x of game $\text{lin}(G)$, Process i *explicitly knows* position $x[j]$ for every index j such that $1 \leq j \leq i$.

More formally, distributed strategies in game G and $\text{lin}(G)$ can be related as stated in the following two lemmas.

Lemma 3. *For every winning distributed strategy $\sigma_1 \otimes \dots \otimes \sigma_n$ in game G , the distributed strategy $\sigma'_1 \otimes \dots \otimes \sigma'_n$ in game $\text{lin}(G)$ defined, for every $i \in [1, n]$, by $\sigma'_i = \sigma_1 \otimes \dots \otimes \sigma_i$, is a winning distributed strategy in game $\text{lin}(G)$.*

Proof. Immediate from definitions and remark above.

In general, there is no converse to such a lemma. In fact, games of the form $\text{lin}(G)$ are linear henceforth existence of winning distributed strategies is decidable which is not true for arbitrary game G . If, however game G is itself linear, a converse hold.

Lemma 4. *If game G is linear, for every winning distributed strategy $\sigma'_1 \otimes \dots \otimes \sigma'_n$ in game $\text{lin}(G)$ there is a winning distributed strategy $\sigma_1 \otimes \dots \otimes \sigma_n$ in game G such that, for every $i \in [1, n]$, $\sigma'_i = \sigma_1 \otimes \dots \otimes \sigma_i$.*

Proof. Observe first that, because the distributed strategy $\sigma'_1 \otimes \dots \otimes \sigma'_n$ is winning in game $\text{lin}(G)$ it only goes to coherent positions. Without lost of generality we can thus assume that the local strategies are themselves coherent. In other words, we can assume that, for every i and j with $1 \leq i < j \leq n$, for every global finite play w in game $\text{lin}(G)$, $\sigma'_i(\text{view}_i(w)) = \sigma'_j(\text{view}_j(w))[1, i]$

Now, the statement follows from the study of linear games presented in [1]. The distributed strategy $\sigma_1 \otimes \dots \otimes \sigma_n$ is defined inductively.

First, strategy σ_1 is just defined to be strategy σ'_1 . In fact, up to position \perp_1 , local games $\text{lin}(G)[1]$ and $G[1]$ are essentially isomorphic.

Next, for every $i \in [2, n]$, strategy σ_i is inductively built from strategy $\sigma'_{i-1} = \sigma_1 \otimes \dots \otimes \sigma_{i-1}$ and strategy σ'_i as follows. The key idea is to *simulate*, from the knowledge of the initial position e_0 , the knowledge of strategy σ'_{i-1} and any local play w_i in $G[i]$, the (unique by linearity) play w that has been played on the projection $G[1, i]$ such that $w_i = \text{view}_i(w)$. Then, we put $\sigma_i(w_i) = \sigma'_i(w)$. Linearity ensures that this simulation can indeed be performed. \square

Now, it occurs that

Theorem 2. *For every n -Process linear distributed game G the game $\text{lin}(\tilde{G})$ is linear and equivalent to game G in the sense that Process team has a finite memory winning strategy in game G if and only if it has one in game $\text{lin}(\tilde{G})$.*

Proof. The proof arguments are similar to the proof arguments for Theorem 1. There, they have been detailed. Here, we only give a sketch of them.

First, any winning distributed strategy $\sigma_1 \otimes \dots \otimes \sigma_n$ induces, by composing it with function cancel, a winning strategy in game \widetilde{G} that, in turn, applying Lemma 3, induces a winning strategy in game $\text{lin}(\widetilde{G})$.

Conversely, assuming there is a finite state distributed winning strategy $\tilde{\sigma}$ for the Process team in game $\text{lin}(\widetilde{G})$, it occurs that one can build, using similar pumping argument, a finite state distributed winning strategy σ' for the Process team in game $\text{lin}(G)$. Then, in turn, this strategy induces a finite state winning distributed strategy by applying Lemma 4.

References

1. J. Bernet and D. Janin. Tree automata and discrete distributed games. In Maciej Liškiewicz and Rüdiger Reischuk, editors, *Fundamentals of Computation Theory*, volume 3623 of *LNCS*, pages 540–551, Lübeck, August 2005. Springer.
2. J. Bernet and D. Janin. Distributed synthesis in zero-delayed architectures with cycles. In E. Najm, J.-F. Pradat-Peyre, and V. V. Donzeau-Gouge, editors, *26th IFIP WG 6.1 International Conference on Formal Methods for Networked and Distributed Systems (FORTE 2006)*, volume 4229 of *LNCS*, pages 175–190, Paris, September 2006. Springer.
3. D. Janin. On the (high) undecidability of distributed synthesis problems. In *Int. conf. on Current Trends in Theo. and Prac. Comp. Science (SOFSEM)*, volume 4362 of *LNCS*, pages 320–329. Springer, 2007.
4. O. Kupferman and M. Y. Vardi. Synthesizing distributed systems. In *In Proc. IEEE Symposium on Logic in Computer Science (LICS'01)*, pages 389–398, 2001.
5. F. Lin and M. Wonham. Decentralized control and coordination of discrete event systems with partial observation. *IEEE Transactions on automatic control*, 33(12):1330–1337, 1990.
6. P. Madhusudan and P.S. Thiagarajan. Distributed controller synthesis for local specifications. In *28th International Colloquium on Automata, Languages and Programming (ICALP'01)*, volume 2076 of *LNCS*, pages 396–407. Springer, 2001.
7. S. Mohalik and I. Walukiewicz. Distributed games. In P. K. Pandya and J. Radhakrishnan, editors, *In Proc. 23th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'03)*, volume 2914 of *LNCS*, pages 338–351. Springer, 2003.
8. D. Perrin and J.E. Pin. *Infinite Words ; Automata, Semigroups, Logic and Games*, volume 141 of *Pure and Applied Mathematics*. Elsevier, 2004.
9. G.L. Peterson and J.H. Reif. Multiple-person alternation. In *20th Annual IEEE Symposium on Foundations of Computer Science (FOCS'79)*, pages 348–363, october 1979.
10. A. Pnueli and R. Rosner. Distributed reactive systems are hard to synthesize. In *In Proc. 31th IEEE Symposium on Foundations of Computer Science (FOCS'90)*, pages 746–757, 1990.