# DISTRIBUTED ASYNCHRONOUS GAMES WITH CAUSAL MEMORY ARE UNDECIDABLE

HUGO GIMBERT

LaBRI, CNRS, Université de Bordeaux, France
*e-mail address*: hugo.gimbert@cnrs.fr

Abstract. We show the undecidability of the distributed control problem when the plant is an asynchronous automaton, the controllers use causal memory and the goal of the controllers is to put each process in a local accepting state.

## Introduction

The decidability of the distributed version of the Ramadge and Wonham control problem [RW89], where both the plant and the controllers are modeled as asynchronous automata [Zie87, DR95] and the controllers have causal memory has been an open problem for some time [GLZ04, MWZ09, Mus15].

In this setting a controllable plant is distributed on several finite-state processes which interact asynchronously using shared actions. On every process, the local controller can choose to block some of the actions, called *controllable* actions, but it cannot block the *uncontrollable* actions from the environment. The choices of the local controllers are based on two sources of information. First, the local controller monitors the sequence of states and actions of the local process. This information is called the *local view* of the controller. Second, when a shared action is played by several processes then all the controllers of these processes can exchange as much information as they want. In particular together they can compute their mutual view of the global execution: their *causal past*.

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

The difficulty of controller synthesis depends on several factors, including the architecture of the information flow between processes (pipeline, ring, ...), the information available to the controllers, and the specification.

In early work on distributed controller synthesis, for example in the setting of [PR90], the only source of information available to the controllers is their local view, and the transmission of information between controllers is strictly limited to reading and writing shared variables

Received by the editors June 2021.

with fixed domains. In this setting, distributed synthesis is not decidable in general, but decidable for architectures without information forks [FS05].

The setting where controllers are allowed to freely communicate with each other upon synchronisation, and can rely on their full causal past to select the controllable actions appeared first in [GLZ04]. In this setting, the class of decidable architectures goes far beyond architectures without information forks, for example all series-parallel games are decidable [GLZ04].

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

We are interested in the following algorithmic problem:

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

**Our contribution.** This paper shows that the DISTRIBUTED CONTROL PROBLEM for plants with six processes is undecidable. The proof is in two steps: reduce the POST CORRESPONDENCE PROBLEM to an intermediary decision problem called the BIPARTITE COLORING PROBLEM (Theorem 2.2) and then reduce the latter problem to the DISTRIBUTED CONTROL PROBLEM (Theorem 3.1). Theorem 2.2 reformulates the POST CORRESPONDENCE PROBLEM as a problem of satisfaction of local constraints on a finite graph. The proof of Theorem 3.1 sheds light on the use of concurrency and causal memory to implement local constraints.

**Related work.** There are several classes of plants for which the DISTRIBUTED CONTROL PROBLEM has been shown decidable: when the dependency graph of actions is series-parallel [GLZ04]; when the processes are connectedly communicating [MTY05]; when the dependency graph of processes is a tree [GGMW13, MW14]; for the class of decomposable games, which encompasses these three former decidability results [Gim17] and includes games with four processes; and finally for plants where a single process have access to controllable actions, while all other processes have only access to uncontrollable actions [BFHH19, Corollary 7].

Petri games with causal memory are another, well-studied, model of distributed asynchronous game [FO17]. In general, the number of token in a Petri game is *unbounded*, and these games are undecidable, since they can encode games on vector addition systems with states [FO17, Theorem 6.9]. When the number of token is *bounded*, two subclasses of Petri games are known to be decidable: Petri games with a bounded number of system players against a single environment player [FO17] as well as Petri games with one system player against a bounded number of environment platers [FG18]. The decidability of the general case of Petri games with a bounded number of tokens was until now an open question, to our knowledge. The DISTRIBUTED CONTROL PROBLEM considered in the present paper can be encoded as the existence of a winning strategy in a Petri game with a *bounded* number of tokens [BFHH19, Theorem 5], thus the undecidability result of the present paper implies as a corollary that Petri games with a bounded number of tokens are undecidable, either with

the termination condition (every token ends up in a final state) or the deadlock-freeness condition (the latter condition is discussed in the conclusion).

A recent result shows that Petri games with *global* winning conditions are undecidable, even with only two system players and one environment player [FGHHO22, Theorem 9]. The latter result relies on an encoding of the Post Correspondence Problem. The global winning condition is used to enforce certain linearizations of the parallel runs and directly encode the synchronous setting of Pnueli and Rosner [PR90, Sch14]. The undecidability proof of the present paper relies as well on an encoding of the Post Correspondence Problem, however the winning condition is purely local, and there is no way to enforce particular linearizations of the play. Instead, we develop a specific proof technique using six processes whose behavior is constrained by the set of possible parallel runs. Remark that four processes would not be enough to get undecidability in our setting, since this case is known to be decidable [Gim17].

**Organisation of the paper.** Section 1 defines the distributed control problem. Section 2 defines the bipartite coloring problem and shows that the Post Correspondence Problem effectively reduces to the bipartite coloring problem. Section 3 effectively reduces the bipartite coloring problem to the distributed control problem.

## 1. Distributed Asynchronous Games with Causal Memory

The theory of Mazurkiewicz traces is very rich, for a thorough presentation see [DR95]. Here we only fix notations and recall the notions of traces, views, prime traces and parallel traces.

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

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

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

The concatenation on words naturally extends to traces. Given two traces $u, v \in A^*_{\equiv}$, the trace $uv$ is the equivalence class of any word in $uv$. The prefix relation $\sqsubseteq$ is defined by

$$(u \sqsubseteq v \iff \exists w \in A^*_{\equiv}, uw \equiv v) \ ,$$

and the suffix relation is defined similarly.

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

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

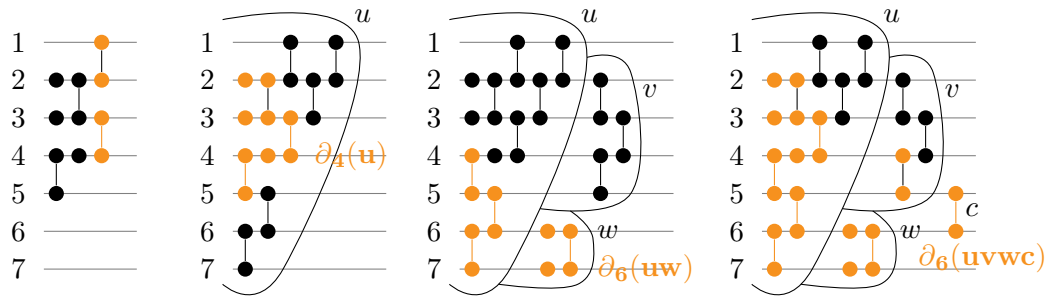


Figure 1: The set of processes is $\{1 \dots 7\}$. A letter is identified with its domain. Here the domains are either singletons, represented by a single dot, or pairs of contigous processes, represented by two dots connected with a vertical segment. The trace $\{2\}\{3\}\{4,5\}\{2,3\}\{4\}\{1,2\}\{3,4\}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

1.1. **Asynchronous automata.** Asynchronous automata are to traces what finite automata are to finite words, as witnessed by Zielonka's theorem [Zie87]. An asynchronous automaton is a collection of automata on finite words, whose transition tables do synchronize on certain actions.

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

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

Such an automaton works asynchronously: each time a letter $a$ is processed, the states of the processes in $\mathrm{dom}(a)$ are updated according to the corresponding transition, while the states of other processes do not change. This induces a natural commutation relation $\mathbb{I}$ on $A$: two letters commute iff they have no process in common i.e.

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

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

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

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

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

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

1.2. **Processes playing against their non-deterministic environment.** Given an automaton $\mathcal{A}$, we want the processes to choose actions in such a way that each one of them is guaranteed to eventually reach a final state.

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

$$A = A_c \sqcup A_e$$

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

We adopt a modern terminology and call the automaton $\mathcal{A}$ together with the partition $A = A_c \sqcup A_e$ a *distributed game*, or even more simply a *game*. In this game, the processes form a team of players which share the same goal but not the same information. The plays of this game are the plays of the automaton and the common goal of all processes is to cooperate with each other so that the play eventually terminates, with every process in a final state. At the beginning of the game, every process $p$ selects, among the letters $A_p$ whose domain contains $p$, a subset of controllable actions to block, possibly all of them. Then $p$ waits for a transition of the automaton on $A_p$. A transition on letter $a$ is possible if either $a$ is uncontrollable or if $a$ is controllable and is not blocked by any of the processes in $\mathrm{dom}(a)$. After the transition, each process in $\mathrm{dom}(a)$ updates the set of controllable actions being blocked.

Every process $p$ follows a *distributed strategy* which dictates how to update the set of controllable blocked actions after every transition. This choice is made by $p$ on the basis of the information available to $p$ about the play.

The information available to $p$ is modelled by a prefix of the global play $u$, called the *causal view* of $p$ and denoted $\partial_p(u)$. In general $\partial_p(u)$ is a strict prefix of $u$, because $p$ cannot directly observe the transitions on actions whose domain does not contain $p$. However, $p$ can learn indirectly about such transitions: every time $p$ performs a transition on some action $a$, he can communicate with all other processes participating in the transition, i.e. all processes in $\mathrm{dom}(a)$. These processes update their causal view to a common mutual value which includes all transitions known by at least one of the processes in $\mathrm{dom}(a)$, plus the current transition. The computation of the causal view $\partial_p(u)$ is illustrated on Fig. 1 and defined formally as follows.

**Definition 2** (Causal view)**.** *For every process $p \in \mathbb{P}$ and trace $u$, the causal view of $u$ by $p$, or equivalently the $p$-*view *of $u$, denoted $\partial_p(u)$, is the unique trace such that $u$ factorizes as $u = \partial_p(u) \cdot v$ and $v$ is the longest suffix of $u$ such that $p \notin \mathrm{dom}(v)$.*

The strategic choices of a process are made solely on the basis of its causal view of the global run, using a distributed strategy.

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

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

*A* distributed strategy *is a tuple $\sigma = (\sigma_p)_{p\in\mathbb{P}}$ where each $\sigma_p$ is a strategy of process $p$.*

*Let $\sigma$ be a distributed strategy. A play $u = a_1 \cdots a_{|u|} \in \mathrm{plays}(\mathcal{A})$ is* consistent with $\sigma$*, or equivalently is a $\sigma$-*play *if:*

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

*A $\sigma$-play is* maximal *if it is not the strict prefix of another $\sigma$-play.*

Albeit a distributed strategy for the controllers limits which plays can occur, by blocking some of the controllable actions, in general there are still many different possible plays consistent with the strategy. There are two sources of non-determinism: first, processes cannot block uncontrollable actions; second the processes may strategically decide to allow several controllable actions, without knowing in advance which one may be used by the next transition. This is illustrated by the example on Fig. 2, whose detailed analysis is provided at the end of the section.

Note that a strategy is forced to allow every environment action to be executed at every moment. This may seem to be a huge strategic advantage for the environment. However depending on the current state, not every action can be effectively used in a transition because the transition function is not assumed to be total. So in general not every environment actions can actually occur in a play, and from some states there may be no outgoing transition, in which case the corresponding process terminates its computation.

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

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

A winning strategy should guarantee the goal in all maximal plays consistent with the strategy. Thus, when one seeks to synthesise a correct controller, the non-deterministic choice of play can be seen as another, antagonistic player, called the environment, trying to pick-up the worst possible transitions compatible with the strategy and prevent the processes to achieve their goal.

In this paper however we do not equip the environment with strategies, the only players are the processes and we are interested in finding a strategy which guarantees a win for the processes, whatever play is non-deterministically selected by the environment.

**An example.** We analyse the example of Fig. 2 in details, in order to highlight the fact that processes may strategically need to unlock several controllable actions at once. Both processes start in the top state and wants to terminate by reaching the starred state on the bottom. From the top state, the process $L$ waits for the environment to trigger one of the uncontrollable actions $a$ or $b$, both local to $L$. Both actions $A$ and $B$ are controllable and shared by both processes, hence the next transition of $L$ and $R$, if any, depends on the

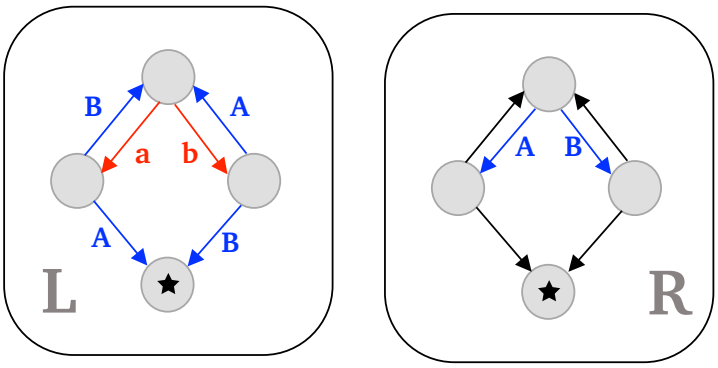


Figure 2: In this game there are two processes $L$ and $R$ (for left and right), whose goal is to reach the starred state. The two red actions $a$ and $b$ are uncontrollable actions, local to the $L$ process. The two blue actions $A$ and $B$ are controllable actions, shared between processes $L$ and $R$. There are also four controllable local actions on process $R$, depicted as black arrows. The processes have a winning strategy.

strategic choice of both processes. What should process $R$ do? He cannot observe $L$ thus his choice is independent of the transitions on $L$.

Assume first that $R$ blocks action $A$ and only allows action $B$. This is not a very good idea. In parallel, the environment may trigger the uncontrollable action $a$. To avoid a deadlock in a non-final state, process $L$ should clearly not block the action $B$ and instead allow a synchronization with $R$ on action $B$. When this occurs, both processes exchange their causal past, and $R$ knows that $L$ is back to its initial state. Then it would a selfish losing strategy on the behalf of $R$ to perform the controllable local transition to its final starred state, since $R$ would terminate but $L$ would then never be able to reach its own starred state, and the processes would lose. Instead, $R$ should return as well to its initial state. As long as $R$ selects a single action, such a loop may repeat forever, in which case the processes (unfairly) lose the game.

What happens instead if $R$ allows both actions $A$ and $B$? Then $L$ can wait for either of the uncontrollable actions $a$ and $b$ to occur, and allow only the action leading to the bottom starred state, say $A$ in case $a$ has been played. The only possible transition is then a synchronization on $A$, in which case $R$ learns that $L$ has reached its final starred state. Then $R$ can select the controllable local transition to its own final starred state, and the processes win. This is a winning strategy.

**The decision problem.** The following decision problem is the central motivation for this work:

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

Whether this problem is decidable has been an open problem for some time [GLZ04, MWZ09, Mus15]. In this paper we show that this problem is undecidable.

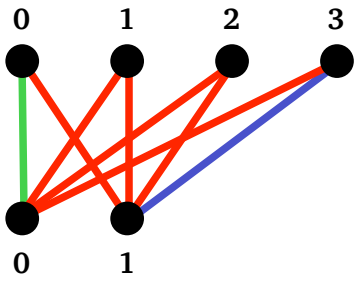


Figure 3: A coloring $f : [n] \times [m] \to C$ with $n = 4$ and $m = 3$. $C$ contains three colours $R, G, B$ (red, green and blue). The edge $(0, 0)$ is green, the edge $(3, 1)$ is blue and all other edges are red. There are three squares in $f$: the square $(G, R) = (f(0, 0), f(1, 1))$, the square $(R, R) = (f(1, 0), f(2, 1))$ and the square $(R, B) = (f(2, 0), f(3, 1))$. The pair $(G, R)$ is also both an upper-triangle $(f(0, 0), f(1, 0))$ and a lower-triangle $(f(0, 0), f(0, 1))$. The pair $(G, B)$ is neither a square nor a triangle, since the corresponding edges are not adjacent.

## 2. The bipartite coloring problem

The proof of the undecidability of the DISTRIBUTED CONTROL PROBLEM makes use of an intermediary decision problem: In the sequel, we use the notation $[n]$ for the integer interval

$$[n] = \{0, \dots, (n-1)\} \ .$$

**Definition 5** (Finite bipartite $C$-colorings)**.** *Fix a finite set $C$ called the set of* colors*. A* finite bipartite $C$-coloring *is a function $f : [n] \times [m] \to C$ where $n$ and $m$ are positive integers. The* initial and final colors *of $f$ are respectively $f(0, 0)$ and $f(n-1, m-1)$.*

We often use the simpler term *coloring* to refer to a finite bipartite $C$-coloring.

A coloring induces a set of patterns called *squares, upper-triangles and lower-triangles*, this is illustrated on Fig. 3 and defined as follows.

**Definition 6** (Patterns induces by a coloring)**.** *Let $f : [n] \times [m] \to C$ be a coloring. The* patterns induced by *$f$ are the following three subsets of $C^2$:*

- *the squares of $f$ are all pairs $\{(f(x, y), f(x+1, y+1)) \mid (x, y) \in [n-1] \times [m-1]\}$.*
- *the upper-triangles of $f$ are all pairs $\{(f(x, y), f(x+1, y)) \mid (x, y) \in [n-1] \times [m]\}$.*
- *the lower-triangles of $f$ are all pairs $\{(f(x, y), f(x, y+1)) \mid (x, y) \in [n] \times [m-1]\}$.*

The BIPARTITE COLORING PROBLEM asks whether there exists a coloring satisfying some constraints on the initial and final colors and on the induced patterns. A *coloring constraint* is given by three subsets $S, UT, LT$ of $C^2$, called the *forbidden patterns*, and two subsets $C_i, C_f$ of $C$ called the sets of *allowed initial and final* colors, respectively. A coloring $f$ *satisfies* the constraint if its initial color is in $C_i$, its final color is in $C_f$ and none of the patterns induced by $f$ is forbidden: no square of $f$ belongs to $S$, no upper-triangle of $f$ belongs to $UT$ and no lower-triangle of $f$ belongs to $LT$.

For example, the coloring depicted on Fig. 3 satisfies the constraint $C_i = \{G, R\}, C_f = \{B\}$ and $S = UT = LT = \{(B, G), (G, B)\}$.

**Definition 2.1** (BIPARTITE COLORING PROBLEM)**.** Given a finite set of colors $C$ and a coloring constraint $(C_i, C_f, S, UT, LT)$, decide whether there exists a finite bipartite $C$-coloring satisfying the constraint.

We illustrate the BIPARTITE COLORING PROBLEM with two examples.

Set $C = \{0, +\}$ and consider the coloring constraint given by $C_i = \{0\}$, $C_f = \{0, +\}$, $UT = \{(+, 0), (0, 0)\}$ $LT = \{(0, +)\}$ and $S = \emptyset$. A coloring $f : [n] \times [m] \to C$ satisfies this constraint iff the edges $(x, y)$ colored by 0 are exactly those whose first coordinate is $x = 0$. The initial constraint enforces $f(0, 0) = 0$. The upper-triangle constraint $UT$ prevents the symbol 0 from appearing on any edge $(x, y)$ such that $x > 0$. By induction, the lower-triangle constraint $LT$ enforces the symbol 0 to appear on any edge $(0, y)$. One can extend this last example on the product alphabet $\{0, +\}^2$ to enforce a coloring to"detect" 0 coordinates in both components $x$ and $y$.

Another example is $C = \{0, -, +\}$ and the coloring constraint given by $C_i = \{0\}$, $C_f = \{0, -, +\}$, $S = C^2 \setminus \{(0, 0), (-, -), (+, +)\}$, $UT = C^2 \setminus \{(0, +), (+, +), (-, 0)\}$ and $LT = C^2 \setminus \{(0, -), (-, -), (+, 0)\}$. We show that for every positive integers $n, m$, there is a unique coloring $f : [n] \times [m] \to C$ which satisfies this constraint, and it is defined by

$$f(x, y) = \begin{cases} 0 & \text{if } x = y \ , \\ + & \text{if } x > y \ , \\ - & \text{if } x < y \ . \end{cases}$$

This definition of $f$ clearly satisfies the constraints, for example the square constraint is satisfied since $x > y$ is equivalent to $x + 1 > y + 1$. To prove unicity, remark first that $C_i$ enforces $f(0, 0) = 0$ and $S$ propagates this constraint to every edge $(x, y)$ such that $x = y$. The constraint $UT$ enforces $f(0, 1) = +$ because $(0, +)$ is the only upper-triangle allowed with a 0 on the left edge. The + is propagated by $S$ which enforces $f(x, x + 1) = +$, whenever $x < \min(n, m - 1)$. And $UT$ enforces all edges $f(x, x + 1), f(x, x + 2), \ldots$ to be marked with + because $(+, +)$ is the only upper-triangle allowed with a + on the left edge. Thus $f(x, y) = +$ whenever $x < y$. The case $x > y$ is symmetric, and the corresponding edges are marked by $-$. If we remove the colors $-$ and + from $C_f$, this creates the extra constraint $n = m$.

The BIPARTITE COLORING PROBLEM can encode problems far more complicated than these toy examples:

**Theorem 2.2.** *There is an effective reduction from the* POST CORRESPONDENCE PROBLEM *to the* BIPARTITE COLORING PROBLEM*.*

The proof of this theorem proceeds in two steps: first characterize a solution to POST CORRESPONDENCE PROBLEM with local rewriting rules and second express these local rewriting rules as coloring constraints. Remark that the reduction generates coloring constraints which allow only solutions $f : [n] \times [m] \to C$ such that $n = m$ (if any). On purpose we do not impose this constraint in the definition of the BIPARTITE COLORING PROBLEM.

*Proof of Theorem 2.2.* An instance of POST CORRESPONDENCE PROBLEM is a finite collection $(u_i, v_i)_{i \in I}$ of tiles on an alphabet $\Sigma$. Each tile $(u_i, v_i)$ is a pair of non-empty words on the alphabet $\Sigma$, $u_i$ is the top word and $v_i$ is the bottom word. The problem is to determine whether there exists a non-empty and finite sequence of indices in $I$ such that

the concatenation of the tiles produce the same two words on the top and the bottom. Duplicates are allowed: the same index $i$ can appear several times in the sequence. In case such a sequence exists, the instance $(u_i, v_i)_{i\in I}$ is said to have a solution. Checking whether an instance has a solution is known to be an undecidable problem [Pos46].

The following result characterizes the existence of a solution to an instance of Post Correspondence Problem.

**Lemma 1** (Local characterization of a PCP solution)**.** *Let $(u_i, v_i)_{i\in I}$ be an instance of* Post Correspondence Problem *on the alphabet $\Sigma$. Let $i_0, i_1, \dots, i_k$ and $j_0, j_1, \dots, j_\ell$ be non-empty and finite sequences of indices and $u = u_{i_0} \dots u_{i_k}$ and $v = v_{j_0} \dots v_{j_\ell}$ the corresponding tiles concatenations. Denote $X = [|u|] \times [|v|]$.*

*The equality $u = v$ holds if and only if there exists a subset* `SameLength` *of $X$ with the following properties.*

1. `SameLength` *contains* $(|u| - 1, |v| - 1)$.
2. *Let $x = (x_u, x_v) \in$* `SameLength` *such that $x_u > 0$ and $x_v > 0$. Then $(x_u - 1, x_v - 1) \in$* `SameLength`*.*
3. $(0, 0)$ *belongs to* `SameLength` *and this is the only element $x = (x_u, x_v) \in$* `SameLength` *with either coordinate equal to* $0$*.*
4. *For every $x = (x_u, x_v) \in$* `SameLength` *the letter of index $x_u$ in $u$ is the same as the letter of index $x_v$ in $v$.*

*The equality $i_0, i_1, \dots, i_k = j_0, j_1, \dots, j_\ell$ holds if and only if there exists a subset* `SameTile` *of $X$ with the following properties:*

5. `SameTile` *contains* $(|u| - 1, |v| - 1)$.
6. *Let $x = (x_u, x_v) \in$* `SameTile` *such that $x_u > 0$ and $x_v > 0$. Say that $x$* starts a new tile in $u$ *if $x_u$ is the first index of a tile in the factorisation $u = u_{i_0} \dots u_{i_k}$ i.e. if there exists $0 \le m \le k$ such that $x_u = |u_{i_0} \dots u_{i_m}|$. Say that $x$* starts a new tile in $v$ *if the symmetrical condition holds for $x_v$ and $v = v_{i_0} \dots v_{j_\ell}$. If $x$ starts a new tile in both $u$ and $v$ then $(x_u - 1, x_v - 1) \in$* `SameTile`*. If $x$ does* not *start a new tile in $u$ then $(x_u - 1, x_v) \in$* `SameTile`*. If $x$ does* not *start a new tile in $v$ then $(x_u, x_v - 1) \in$* `SameTile`*.*
7. $(0, 0)$ *belongs to* `SameTile`*. For every $(x_u, x_v) \in$* `SameTile`*, $(x_u < |u_{i_0}| \iff x_v < |v_{j_0}|)$.*
8. *Let $x = (x_u, x_v) \in$* `SameTile` *and denote $i_u(x) \in I$ the index of the tile appearing at index $x_u$ in the factorisation $u = u_{i_0} \dots u_{i_k}$. Define $i_v(x) \in I$ symmetrically with respect to the factorisation $v = v_{i_0} \dots v_{j_\ell}$. Then $i_u(x)$ and $i_v(x)$ are equal.*

*Proof.* The proof is elementary.

We prove the first part of the lemma: the characterisation of $u = v$. For the direct implication, assume $u = v$. Then the set `SameLength` $= \{(x, x) \mid 0 \le x < |u| = |v|\}$ has properties 1. to 4. In the opposite direction, assume there exists `SameLength` $\subseteq X$ with properties 1. to 4. We show that:

$$\text{for every } 0 < i \le \min(|u|, |v|), \texttt{SameLength} \text{ contains } (i, i) \quad . \tag{2.1}$$

Assume w.l.o.g. that $|u| = \min(|u|, |v|)$. A simple induction shows that for every $0 < i \le \min(|u|, |v|)$, `SameLength` contains $(|u| - i, |v| - i)$: the case $i = 1$ follows from property 1. and the induction step from property 2. Then $(0, |v| - |u|) \in$ `SameLength` thus, by property 3, $|u| = |v|$. Hence property (2.1). Finally $u = v$ follows by property 4.

We now prove the second part of the lemma: the characterisation of $i_0, i_1, \dots, i_k = j_0, j_1, \dots, j_\ell$. The proof relies on the subset $A \subseteq X$ containing all pairs $(x_u, x_v) \in X$ such

that the number of tiles appearing *after* position $x_u$ in $u$ is the same that the number of tiles appearing *after* position $x_v$ in $v$. Formally,

$$A = \{(x_u, x_v) \in X \mid \exists m \in 0, \dots, \min(k, \ell) \\ |u_{i_0} \cdots u_{i_{k-m-1}}| \leq x_u < |u_{i_0} \cdots u_{i_{k-m}}| \text{ and } |v_{j_0} \cdots v_{j_{\ell-m-1}}| \leq x_v < |v_{j_0} \cdots v_{j_{\ell-m}}|\} \ .$$

Remember that all the words $u_i, i \in I$ and $v_i, i \in I$ appearing in the tiles are non-empty. Thus, $A$ contains $(|u|-1, |v|-1)$. And $A$ contains $(0,0)$ iff $k = \ell$. Following its definition parametrized by $m \in 0 \dots \min(k, \ell)$, $A$ is partitioned in $A_0, \dots, A_{\min(k,\ell)}$.

For the direct implication, Assume $i_0, i_1, \dots, i_k = j_0, j_1, \dots, j_\ell$ (hence in particular $k = \ell$ ). Then the set $\texttt{SameTile} = A$ clearly satisfies all properties (5) to (8).

Conversely, assume $\texttt{SameTile} \subseteq X$ satisfies all properties (5) to (8), and show that $i_0, i_1, \dots, i_k = j_0, j_1, \dots, j_\ell$.

We show first that SameTile contains all elements in $A$. Assume by contradiction that some element $x$ in $A$ is missing from SameTile, and assume $x$ is maximal for the lexicographic ordering in $A \setminus \texttt{SameTile}$. Let $m$ such that $x \in A_m$. The set $A_m$ is the cartesian product of two intervals, hence it has a maximum, namely $x_m = (|u_{i_0} \dots u_{i_{k-m}}| - 1, |u_{i_0} \dots u_{i_{\ell-m}}| - 1)$. There are two cases, depending whether $x = x_m$ or not. Assume first $x = x_m$. According to property (5), the maximal element $(|u|-1, |v|-1)$ in $A$ belongs to SameTile thus $x = x_m \neq (|u|-1, |v|-1)$. Since $(|u|-1, |v|-1) \in A_0$ then $m > 0$. As a consequence, $(x_u + 1, x_v + 1)$ is the minimal element in $A_{m-1}$ and by induction hypothesis, $(x_u + 1, x_v + 1) \in \texttt{SameTile}$. Moreover, $(x_u + 1, x_v + 1)$ starts a new tile in both $u$ and $v$ and by property (6), we get $x \in \texttt{SameTile}$ as well. The second case is when $x$ is not maximum in $A_m$. Since $A_m$ is the cartesian product of two intervals, either $(x_u + 1, x_v)$ or $(x_u, x_v + 1)$ is in $A_m$ as well, and $x$ does not start a new tile on the corresponding coordinate. We conclude that $x \in \texttt{SameTile}$ using the induction hypothesis and property (6).

Now we show $k = \ell$. Assume w.l.o.g. $k = \min(k, \ell)$. Since SameTile contains all $A$ then in particular SameTile contains $(0, |v_{j_0} \cdots v_{j_{\ell-k}}| - 1)$ (take $m = k$ in the definition of $A$). According to property (7), $|v_{j_0} \cdots v_{j_{\ell-k}}| \leq |v_{j_0}|$ thus $k = \ell$.

Using property (8), we conclude that $i_0, i_1, \dots, i_k = j_0, j_1, \dots, j_\ell$. □

We now define an instance of the BIPARTITE COLORING PROBLEM encoding the constraints in Lemma 1. We start with defining the set of colors. Let $Q_u$ be the finite subset of $\Sigma \times I \times \mathbb{N} \times \{0,1\}^2$ containing all tuples $(a, i, m, b_0, b_1)$ where $m < |u_i|$ and $a$ is the $m$-th letter of $u_i$. Given $q = (a, i, m, b_0, b_1) \in Q_u$, $a$ is called the letter of $q$, $i$ its tile index, $m$ its tile position, $b_0$ its initial letter flag and $b_1$ its initial tile flag. The tile position is maximal in $q$ if it is equal to $|u_i| - 1$. Define $Q_v$ similarly with respect to the bottom parts of the tiles $(v_i)_{i \in I}$. Let $C$ be the set of colors

$$Q_u \times Q_v \times \{\texttt{SameLength}, -\} \times \{\texttt{SameTile}, -\} \ .$$

We define a constraint coloring problem reflecting conditions (1) to (8) of the previous lemma. We use $*$ as a wildcard character which can be replaced by any symbol.

First, we set constraints which enforce the $Q_u$-component of $f(x_u, x_v)$ to depend only on $x_u$ and the $Q_v$-component of $f(x_u, x_v)$ to depend only on $x_v$. For that we include in the set of forbidden lower-triangle $LT$ all pairs $(q_u, *, *, *)(q'_u, *, *, *)$ with $q_u \neq q'_u$. A simple induction shows that if a coloring $f : [n] \times [m] \to C$ does not induce such lower-triangles pattern, then there exists $f_u : [n] \to Q_u$ such that for every $x = (x_u, x_v) \in [n] \times [m]$, the first component of $f(x)$ is $f_u(x_u)$. Symmetrically, we add to the set of forbidden upper-triangles

$UT$ all pairs $(*, q_v, *, *)(*, q'_v, *, *)$ with $q_v \neq q'_v$ so that the $Q_v$-component of $f$ is induced by $f_v : [m] \to Q_v$.

Denote $u \in \Sigma^n$ to be the word of length $n$ whose $x_u$-th letter is the first component of $f_u(x_u) \in Q_u$. Define $v \in \Sigma^m$ symmetrically with respect to $f_v$.

Second, we want to ensure that $u$ is a concatenation of upper tiles of the PCP instance, i.e. there exists $i_0, i_1, \dots, i_k$ such that $u = u_{i_0} \cdots u_{i_k}$. This regular constraint can be checked by a deterministic automaton on $Q_u$, which we implement using the coloring constraints. First, we require that in every initial color $(q, *, *, *)$, the tile position in $q$ is 0. Moreover, we put constraints on all upper-triangles $(q, *, *, *), (q', *, *, *)$: either the tile position is maximal in $q$ and has value 0 in $q'$, or the tile index does not change between $q$ and $q'$ while the tile position is incremented of exactly one unit. Finally, we require that in every final color $(q, *, *, *)$, the tile position in $q$ is maximal. With a symmetric construction, we ensure the symmetric constraint on $v$: there exists $j_0, j_1, \dots, j_\ell$ such that $v = v_{j_0} \cdots v_{j_\ell}$.

Third, we want that the initial letter flag and the initial tile flag of $f_u(x_u)$ indicate respectively whether or not $x_u = 0$ and whether or not $x_u < |u_{i_0}|$. We require that all initial colors have both flags set to 1. We forbid any upper triangle $(q_u, *, *, *)(q'_u, *, *, *)$ where $q'_u$ has the initial letter flag set to 1. We forbid any upper triangle $(q_u, *, *, *)(q'_u, *, *, *)$ where $q'_u$ has the initial tile flag set to 1 unless $q_u$ has also the initial tile flag set to 1 and the tile position in $q_u$ is not maximal. A symmetric construction ensures that the initial letter and tile flags of $f_v(u_v)$ indicate respectively whether or not $u_v = 0$ and $u_v < |v_{j_0}|$.

This way we have a natural correspondence between finite bipartite coloring satisfying the above constraints and pairs of factorisations $u = u_{i_0} \cdots u_{i_k}$ and $v = v_{j_0} \cdots v_{j_\ell}$ like in Lemma 1. Using this correspondence, we can rephrase the eight conditions in Lemma 1 as eight coloring constraints on a set of colours $C' \subseteq C$.

(1) Every final colour has type $(*, *, \texttt{SameLength}, *)$.
(2) Forbid all squares $(*, *, -, *), (*, *, \texttt{SameLength}, *)$.
(3) Every initial colour has type $(q_u, q_v, \texttt{SameLength}, *)$, with both initial letter flags in $q_u$ and $q_v$ set to 1. Remove from the alphabet $C$ any color $(q_u, q_v, \texttt{SameLength}, *)$ such that the initial letter flags of $q_u$ and $q_v$ are different.
(4) Remove from the alphabet $C$ any color $(q_u, q_v, \texttt{SameLength}, *)$ such that the letters in $q_u$ and $q_v$ (i.e. their projections on $\Sigma$) are different.
(5) Every final colour has type $(*, *, *, \texttt{SameTile})$.
(6) Forbid all squares $(*, *, *, -), (q'_u, q'_v, *, \texttt{SameTile})$ such that both tile positions in $q'_u$ and $q'_v$ are 0. Forbid all upper-triangles $(*, *, *, -), (q'_u, *, *, \texttt{SameTile})$ unless the tile position in $q'_u$ is 0. Forbid all lower-triangles $(*, *, *, -), (*, q'_v, *, \texttt{SameTile})$ unless the tile position in $q'_v$ is 0.
(7) Every initial colour has type $(q_u, q_v, *, \texttt{SameTile})$, with both initial tile flags in $q_u$ and $q_v$ set to 1. Remove from the alphabet $C$ any color $(q_u, q_v, *, \texttt{SameTile})$ such that the initial tile flags in $q_u$ and $q_v$ are different.
(8) Remove from the alphabet $C$ any color $(q_u, q_v, *, \texttt{SameTile})$ such that the tile indices in $q_u$ and $q_v$ are different.

Applying conditions, 3,4, 7 and 8, we get a set of colours $C' \subset C$ on which the constraint coloring problem is defined. The initial condition $C_i$ is the set of colours in $C'$ of type $(q_u, q_v, \texttt{SameLength}, \texttt{SameTile})$ such that in both $q_u$ and $q_v$, the tile position is 0 and both initial letter flags and initial tile flags are set to 1 . The final condition $C_f$ is the set of colours of type $(q_u, q_v, \texttt{SameLength}, \texttt{SameTile})$ where the tile positions in both $q_u$ and $q_v$

are maximal. The set of forbidden patterns is obtained by taking the union of all forbidden squares, upper-triangles and lower-triangles mentioned above.

According to Lemma 1, there exists a coloring satisfies the constraints if and only if the PCP instance has a solution. This terminates the proof of Theorem 2.2. □

A direct consequence of Theorem 2.2 is that the BIPARTITE COLORING PROBLEM is undecidable.

**An example.** We analyse the example on Fig. 2 in details. Both processes wants to terminate by reaching their local smiley state. Actions in red are uncontrollable, from his initial state, $L$ has no influence on the play and can only wait for the environment to trigger $a$ or $b$, both local actions. The actions $A$ and $B$ are controllable and shared by both processes, hence the next transition of $L$ is bound to the strategic choice of $R$. Assume the environment has triggered $a$. Clearly $L$ should allow at least the action $A$.

What happens if $L$ allows both actions $A$ and $B$ after the environment has picked-up $a$? Then all depends on $R$, who has no knowledge about the current state of $L$, and cannot know which action he should block to get $L$ to terminate. When the action $A$ or $B$ occurs, then both processes are allowed to communicate and the right process learns, too late, what were the previous states of the left process and whether $L$ has finally reached his terminal state. Assume yes, then clearly $R$ can play $\top$ and terminate as well. Assume no, then it would a selfish losing strategy for $R$ to play $\top$, since $R$ would terminate but $L$ would be deadlocked forever in a non-final state. Thus as long as $L$ has not terminated, $R$ should play $\bot$. If $R$ is very unlucky then he may be wrong forever and the processes will (unfairly) lose the play. Thus letting $L$ unblock both actions $A$ and $B$ is not a winning strategy.

What happens instead if after the environment has picked-up $a$, process $L$ only allows action $A$ and block action $B$? Albeit process $R$ cannot guess whether $a$ or $b$ was triggered by the environment on the left, he can unblock both actions $A$ and $B$, and in the only possible continuation of the play, the process $L$ reaches his final state after which process $R$ can play $\top$. Symmetrically, process $L$ can block action $A$ if the environment selects $b$, and this strategy is winning.

## 3. Undecidability of the Distributed Synthesis Problem with 6 processes

The main result of this section is:

**Theorem 3.1.** *There is an effective reduction from the* BIPARTITE COLORING PROBLEM *to the* DISTRIBUTED CONTROL PROBLEM *with six processes.*

We show how to effectively transform an instance $(C_i, C_f, S, UT, LT)$ of the BIPARTITE COLORING PROBLEM on a set of colors $C$ into a distributed game $G$ such that:

**Lemma 3.2.** *There is a winning strategy in $G$ if and only if there is a finite bipartite coloring satisfying the constraints $(C_i, C_f, S, UT, LT)$.*

In the rest of the section we describe the construction of the game $G$ and then prove Lemma 3.2.

3.1. **Turning a coloring problem into a game.** The game $G$ has six processes divided into two pools: the top pool $T_0, T_1, T_2$ and the bottom pool $B_0, B_1, B_2$. The transitions of these processes are quite similar, thus we often use the notation $X_\ell$ to designate any of the six processes, a generic notation where $X \in \{T, B\}$ and $\ell \in \{0, 1, 2\}$. For every process $X_\ell$ we denote $\text{next}(X_\ell)$ the process $X_{(\ell+1) \mod 3}$ and we denote $\text{prev}(X_\ell)$ the process $X_{(\ell-1) \mod 3}$. For example, $\text{next}(T_0) = T_1$ and $\text{prev}(B_0) = B_2$.

**Uninterrupted plays.** These rare plays where only the following actions are used:

- There are controllable actions called *increments*, which synchronize two processes of the same pool. In each pool $X \in \{T, B\}$ there are three increment actions $\texttt{I}_{X,0}, \texttt{I}_{X,1}$ and $\texttt{I}_{X,2}$. Every increment $\texttt{I}_{X,\ell}$ is controllable and is shared between processes $X_\ell$ and $\text{next}(X_\ell)$.
- Every process $X_\ell$ has also a controllable local action $\texttt{END}_{X,\ell}$.
- There is a controllable action `WIN` synchronizing all six processes.

The transitions are defined so that every maximal uninterrupted play looks like the one on Figure 4: it is the parallel product of two plays $(\texttt{I}_{T,0}\texttt{I}_{T,1}\texttt{I}_{T,2})^n$ and $(\texttt{I}_{B,0}\texttt{I}_{B,1}\texttt{I}_{B,2})^m$ for some positive integers $n, m$, followed by the six local `END` actions and finally the global action `WIN`.

Since all the actions in an uninterrupted play are controllable, the number of rounds performed before playing actions `END` is determined by the players and is not influenced by the environment. Intuitively, when the number of rounds of the top and bottom pools are $n$ and $m$, respectively, the players claim to have a solution $f : [n] \times [m] \to C$ to the constrained coloring problem.

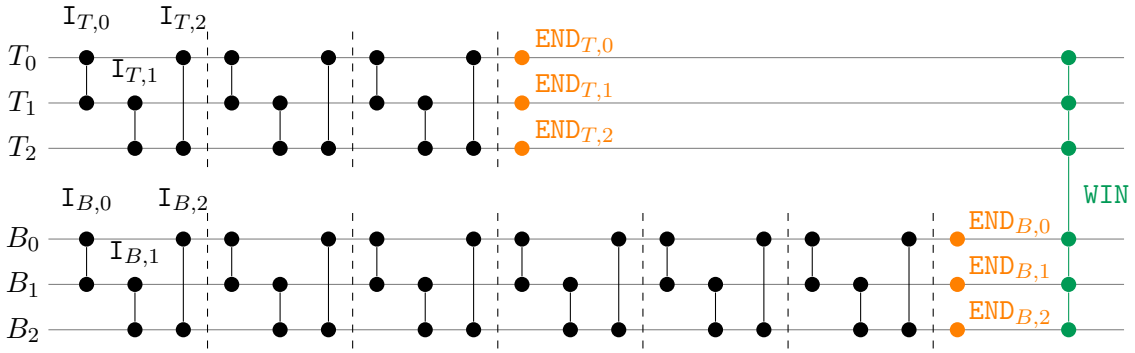


Figure 4: A maximal uninterrupted play where the top pool plays 3 rounds and the bottom pool plays 6 rounds. Rounds are separated by dashes. All actions are controllable.

In each pool $X \in \{T, B\}$, increments have to be played in a fixed order:

$$\texttt{I}_{X,0}, \texttt{I}_{X,1}, \texttt{I}_{X,2}, \texttt{I}_{X,0}, , \texttt{I}_{X,1}, \dots \ .$$

Note that no two increment actions of the same pool can commute, thus an interrupted play $(\texttt{I}_{X,0}\texttt{I}_{X,1}\texttt{I}_{X,2})^*$ in a pool $X$ has a single linearization. Each subword $\texttt{I}_{X,0}, \texttt{I}_{X,1}, \texttt{I}_{X,2}$ is called a *round*. This pattern is enforced by maintaining in the state space of every process $X_\ell$ a flag indicating whether or not the process is allowed to synchronize with $\text{next}(X_\ell)$. The flag is initially set to 1 for $X_0$ and to 0 for $X_1$ and $X_2$, and is toggled every time the process performs an increment.

The only way for processes to win is a transition on the global action WIN, in which case they immediately and definitively enter a final state, and no further transition can occur after that. A transition on WIN can be triggered once all six processes have performed a local transition on their END action.

The transition on the local action $\texttt{END}_{X,\ell}$ for process $X_\ell$ is available if and only if the process has played at least one increment action and has finished the current round. For $X_0$ and $X_2$ (resp. for $X_1$) it means that the last increment was $\texttt{I}_{X_2}$ (resp. was $\texttt{I}_{X_1}$). This constraint can easily be encoded with a flag in the state space of the process.

A play stays uninterrupted, like the one on Figure 4, when all actions are controllable and the environment does not play at all, in which case the game is easy to win. But in general, a winning strategy should also react correctly to uncontrollable actions of the environment called *checks*.

**Interrupting plays by color checks.** The environment has the ability to interrupt a play by triggering an uncontrollable *check action*. This is represented on Figure 5. Intuitively, when the environment interrupts the play after $x$ rounds of the top pool and $y$ rounds of the bottom pool, the players are asked to pick the color $f(x, y) \in C$ of the edge $(x, y)$ of the solution to the coloring problem. There are rules which enforce the players to define that way a coloring $f : [n] \times [m] \to C$ which satisfies the constraints, where $n$ and $m$ are the number of rounds of the top and bottom pools, respectively, before playing END. If the strategy of the players makes them cheat or describe a coloring $f$ which does not satisfy the constraints, the environment can trigger one or two checks which make the players lose.

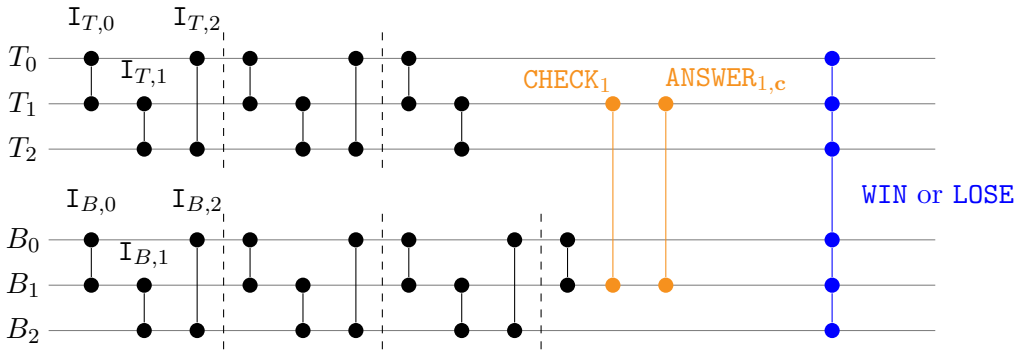


Figure 5: An uncontrollable check of processes $T_1$ and $B_1$ after 3 rounds of the top pool and 4 rounds of the bottom pool. Intuitively, the environment is asking to processes $T_1$ and $B_1$ "what is the value of $f(3, 4)$, where $f : [n] \times [m] \to C$ is your solution to the constrained coloring problem ?". The check is followed by the controllable action $\texttt{ANSWER}_{1,c}$ of the same two processes, parametrized by a color $c \in C$. This way the processes claim that $f(3, 4) = c$. The answer is followed by either the WIN or the LOSE action, which terminates the game.

For every index $\ell \in \{0, 1, 2\}$, there is an uncontrollable action $\texttt{CHECK}_\ell$ which synchronizes the two processes $T_\ell$ and $B_\ell$. A transition on action $\texttt{CHECK}_\ell$ is possible if and only if the last action of both processes is either an increment or the action END. In particular, both processes should have played at least one increment. After the check, the only possible

transitions rely on controllable actions $\texttt{ANSWER}_{\ell,c}$ synchronizing $T_\ell$ and $B_\ell$ and indexed by colors $c \in C$. The answer is stored in the state space of the processes.

After that, there are only two possible outcomes: either the `WIN` or the `LOSE` action synchronizes all processes, and then no further transition is available. There is no extra condition needed to execute a global transition on `WIN`: as soon as at least one pair of processes has performed an answer the transition on `WIN` is available.

**Six ways to lose.** Considering the non-deterministic environment as adversarial, it prefers transitions on `LOSE` rather than those on `WIN`. Losing can occur in six different ways, five of them correspond to the five constraints in the definition of the BIPARTITE COLORING PROBLEM.

A transition on `LOSE` is possible in case one of the following conditions holds:

(a) if both processes $T_\ell$ and $B_\ell$ have played a single increment and their answer is a color $c$ which is not initial (i.e. $c \notin C_i$). The condition "has played a single increment" can be stored in the state space of the processes, and used to allow or not this transition; or

(b) if both processes $T_\ell$ and $B_\ell$ have played `END` and their answer is a color $c$ which is not final (i.e. $c \notin C_f$).

There are four other ways to lose, which require that two checks occur in parallel, for two different pairs of processes indexed by $\ell$ and $\ell'$, respectively. This leads to two parallel answers corresponding to colors $c$ and $c'$, respectively. The conditions for losing rely on the definition of the *round index* of a play $u$ for a process $X_\ell$, defined as $\lfloor (h-1)/2 \rfloor$ where $h > 0$ is the number of increments played by $X_\ell$ in $u$ when the check occurs. The round index is denoted $R_{X_\ell}(u)$.

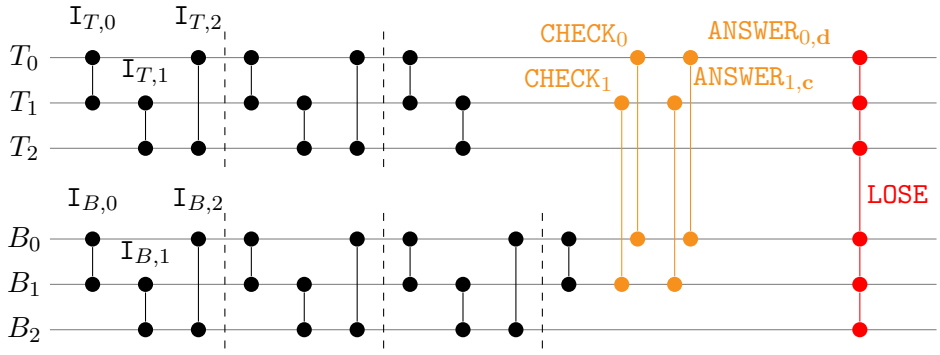


Figure 6: One of six ways to lose: two checks occur in parallel. In both pools, the two checks occur during the same round. The two pairs of processes give (in parallel as well) two different answers $c \neq d$. This allows the environment to trigger the `LOSE` action. Intuitively, the players have cheated since they gave two different answers to the question "what is the value of $f(3,4)$" ?

The conditions for `LOSE` are based on a comparison of $c$ and $c'$ as well as the respective rounds of the processes. For each pool $X \in \{T, B\}$ and every play $u$ there are two possibilities. W.l.o.g., assume we have chosen $\ell$ and $\ell'$ so that $\ell' < \ell$. Either both processes $X_\ell$ and $X_{\ell'}$ are in the same round (meaning they have the same round index in $u$) or process $X'_\ell$ is one

round ahead of $X_\ell$ (meaning that $R_{X'_\ell}(u) = 1 + R_{X_\ell}(u)$). No other case may occur because every uninterrupted play in pool $X$ is a prefix of a word in $(\mathtt{I}_{X,0}\mathtt{I}_{X,1}\mathtt{I}_{X,2})^*\mathtt{END}$ and $\ell' < \ell$.

A transition on LOSE is possible in case one of the following conditions holds:

(c) in both pools both processes are in the same round but the answers are different (i.e. $c \neq c'$); or

(d) in both pools process $X'_\ell$ is one round ahead of $X_\ell$ and the pair of answers $(c, c')$ is a forbidden square (i.e. $(c, c') \in S$); or

(e) in the top pool process $T'_\ell$ is one round ahead of $T_\ell$ while in the bottom pool both processes are in the same round and the pair of answers $(c, c')$ is a forbidden upper-triangle (i.e. $(c, c') \in UT$); or

(f) in the top pool both processes are in the same round while in the bottom pool process $B'_\ell$ is one round ahead of $B_\ell$ and the pair of answers $(c, c')$ is a forbidden lower-triangle (i.e. $(c, c') \in LT$).

Remark that the condition"being in the same round" can be easily implemented in the transition table. For that, it is enough that each process keeps track in its state space of the number of increments it has already played, modulo 4. From that counter, one can derive the parity of the round index and compare the respective parities for the two processes $X_\ell$ and $X_{\ell'}$.

Condition (c) is illustrated on Figure 6 while condition (d) is illustrated on Figure 7.

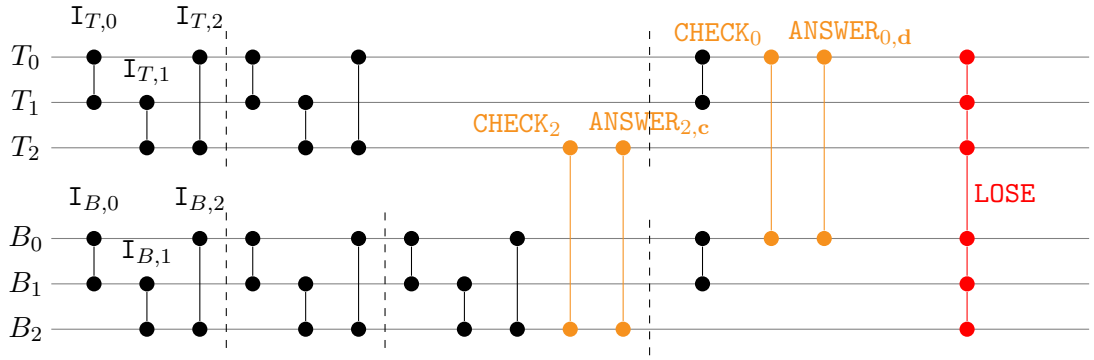


Figure 7: Another way to lose: two checks occur in parallel. $\mathtt{CHECK}_2$ occurs in round 2 of pool $T$ and round 3 of pool $B$ thus $T_2, B_2$ are assumed to answer with $d = f(2, 3)$. In parallel, $\mathtt{CHECK}_0$ occurs in round 3 of pool $T$ and round 4 of pool $B$. Hence $T_0, B_0$ are assumed to answer with $d = f(3, 4)$. We assume here that the pairs of answers $(c, d)$ is a forbidden square, which allows the environment to trigger the LOSE action.

3.2. **Proof of Lemma 3.2.** Lemma 3.2 is an equivalence. We start with the converse implication. Assume that there is a finite bipartite coloring $f : [n] \times [m] \to C$ satisfying the constraints $(C_i, C_f, S, UT, LT)$. A winning strategy for processes of the top (resp. bottom) pool, as long as they are not interrupted by a check, consists in playing $n$ rounds (resp. $m$ rounds) of increments, followed by the END action. If the environment triggers a check on $T_\ell$ and $B_\ell$ after some play $u$, the processes answer $f(x, y)$, where $x$ (resp. $y$) is the round index

of $T_\ell$ (resp. of $B_\ell$) in $u$. This information is available to both processes since they share the same causal past right after the check. Since $f$ satisfies the constraint, the environment cannot trigger a transition on LOSE. Condition (a) cannot occur because all the answers when both processes are in round 0 are equal to $f(0,0) \in C_i$. Condition (b) cannot occur because all the answers when both processes are in round $(n,m)$ are equal to $f(n,m) \in C_f$. Condition (c) cannot occur because the answers only depends on the rounds, independently of the identities of the processes and the number of increments they have performed in the current round. Condition (d), (e) and (f) cannot occur because no pattern induced by $f$ is forbidden.

We now prove the direct implication of Lemma 3.2. Assume that processes have a winning strategy $\sigma$. The first step is to define a finite bipartite $C$-coloring $f$. Let $n$ (resp. $m$) be round index of $T_1$ (resp. of $B_1$) in the maximal uninterrupted play consistent with $\sigma$ (which exists since $\sigma$ is winning). For every $(x,y) \in [n] \times [m]$ denote

$$\begin{aligned} u_{T,x} &= (\mathtt{I}_{T,0}\mathtt{I}_{T,1}\mathtt{I}_{T,2})^x \mathtt{I}_{T,0}\mathtt{I}_{T,1} \\ u_{B,y} &= (\mathtt{I}_{B,0}\mathtt{I}_{B,1}\mathtt{I}_{B,2})^y \mathtt{I}_{B,0}\mathtt{I}_{B,1} \\ u_{x,y} &\text{ the parallel product of } u_{T,x} \text{ and } u_{B,y} \end{aligned}$$

and let $f(x,y)$ be the answer of $T_1, B_1$ after a check on the play $u_{x,y}$ i.e.

$$f(x,y) = c \text{ such that } (u_{x,y}\mathtt{CHECK}_1\mathtt{ANSWER}_{1,c}) \text{ is a } \sigma\text{-play} \ .$$

This is well defined: since $\sigma$ is winning it creates no deadlock in a non-final state, thus at least one answer of $T_1, B_1$ is allowed by $\sigma$ after $u_{x,y}\mathtt{CHECK}_1$. In the sequel, we assume that exactly one answer is allowed by $\sigma$. This is w.l.o.g.: if there is a winning strategy which allows several answers, we can restrict it to allow a single answer after every check, and it will still be winning.

We show that $f$ satisfies the five constraints $(C_i, C_f, S, UT, LT)$.

**Constraint $C_i$ on initial colors.** Since $\sigma$ is winning, no transition on action LOSE is available after $u_{0,0}\mathtt{CHECK}_1\mathtt{ANSWER}_{1,f(0,0)}$. Thus according to condition (a) above, $f(0,0)$ is an initial color.

**Constraint $C_f$ on final colors.** We consider the checks

$$\begin{aligned} p_1 &= u_{n,m}\mathtt{CHECK}_1 \\ p_2 &= u_{n,m}\mathtt{I}_{T,2}\mathtt{I}_{B,2}\mathtt{CHECK}_2 \\ p_3 &= u_{n,m}\mathtt{END}_{T,1}\mathtt{END}_{B,1}\mathtt{CHECK}_1 \ . \end{aligned}$$

By definition of $f$, the answer to $p_1$ is $f(n,m)$. Denote $c_2$ and $c_3$ the answers to $p_2$ and $p_3$, respectively. Remark that $p_1$ and $p_2$ can occur in parallel, because processes $T_1, B_1$ do not play in $\mathtt{I}_{T,2}\mathtt{I}_{B,2}\mathtt{CHECK}_2$. For similar reasons, also $p_2$ and $p_3$ can occur in parallel. Moreover, in all these three plays, processes $T_1$ and $T_2$ are in the same round, and processes $B_1$ and $B_2$ are also in the same round (remember that END actions are not accounted for when computing the round index). Since $\sigma$ is winning, condition (c) will neither be satisfied by the pair of parallel checks $p_1, p_2$ nor by the pair $p_2, p_3$, thus $f(n,m) = c_2 = c_3$. Finally, after the check $p_3$, both processes $T_1, B_1$ are in the state End and they answer $f(n,m)$. According to condition (b) above, $f(n,m)$ is a final color.

**Constraint $S$ on forbidden squares.** Let $x, y \in [n-1] \times [m-1]$. We consider four possible checks

$$\begin{aligned} p_1 &= u_{x,y}\texttt{CHECK}_1 \\ p_2 &= u_{x,y}\texttt{I}_{T,2}\texttt{I}_{B,2}\texttt{CHECK}_2 \\ p_0' &= u_{x,y}\texttt{I}_{T,2}\texttt{I}_{B,2}\texttt{I}_{T,0}\texttt{I}_{B,0}\texttt{CHECK}_0 \\ p_1' &= u_{x+1,y+1}\texttt{CHECK}_1 \ . \end{aligned}$$

By definition of $f$, the answers to $p_1$ and $p_1'$ are $f(x,y)$ and $f(x+1,y+1)$, respectively. Denote $c$ the answer to $p_2$ and $d$ the answer to $p_0'$. The four checks are illustrated on Figure 8.

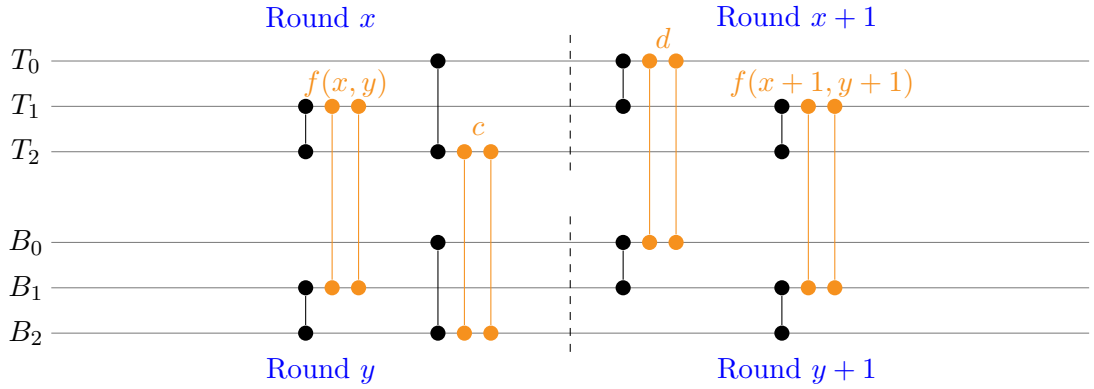


Figure 8: The four checks used to enforce the constraint on forbidden squares, and their answers. These four checks cannot happen altogether in the same play, but any pair of two successive checks can happen in parallel, in the same play.

The two checks $p_1$ and $p_2$ can occur in parallel during the same round, thus according to condition c), $f(x,y) = c$. The two checks $p_2$ and $p_0'$ can occur in parallel and the check $p_0'$ is one round ahead of the check $p_2$, with respect to both pools. Since $\sigma$ is winning, condition (d) cannot happen hence $(c,d)$ is not a forbidden square. The two checks $p_0'$ and $p_1'$ may occur in parallel during the same round, thus according to condition (c), $d = f(x+1,y+1)$. Finally $(f(x,y), f(x+1,y+1))$ is not a forbidden square.

**Constraint $UT$ on forbidden upper-triangles.** Let $x, y \in [n-1] \times [m]$. We consider the plays

$$\begin{aligned} p_1 &= u_{x,y}\texttt{CHECK}_1 \\ p_2 &= u_{x,y}\texttt{I}_{T,2}\texttt{I}_{B,2}\texttt{CHECK}_2 \\ p_0' &= u_{x,y}\texttt{I}_{T,2}\texttt{I}_{B,2}\texttt{I}_{T,0}\texttt{CHECK}_0 \\ p_1' &= u_{x+1,y}\texttt{CHECK}_1 \ . \end{aligned}$$

By definition of $f$, the answers to $p_1$ and $p_1'$ are $f(x,y)$ and $f(x+1,y)$, respectively. Denote $c$ the answer to $p_2$ and $d$ the answer to $p_0'$. The four checks are illustrated on Figure 9.

The two checks $p_1$ and $p_2$ can occur in parallel during the same round thus according to condition (c), $f(x,y) = c$. The two checks $p_2$ and $p_0'$ can occur in parallel, with process $T_0$ one round ahead of $T_2$ and processes $B_0$ and $B_2$ in the same round thus according to

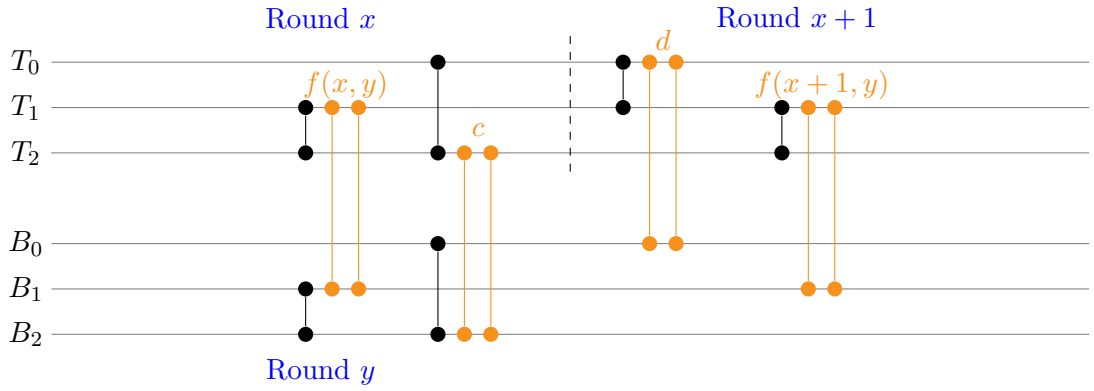


Figure 9: The four checks used to enforce the constraint on forbidden upper-triangles, and their answers. These four checks cannot happen altogether in the same play, but any pair of two successive checks can happen in parallel, in the same play.

condition (e), $(c,d)$ is *not* a forbidden upper-triangle. The two checks $p'_0$ and $p'_1$ can occur in parallel during the same round thus according to condition (c), $d = f(x+1,y)$. Finally, $(f(x,y), f(x+1,y))$ is not a forbidden upper-triangle.

**Constraint $LT$ on forbidden lower-triangles.** This case is symmetric with the $UT$ case above.

Finally, $f$ is a finite bipartite coloring satisfying all six constraints. This terminates the proof of the direct implication of Lemma 3.2. □

## Conclusion

We have proved that the distributed control problem is undecidable for six processes, thus closing an open problem which was first raised in [GLZ04] and has afterwards been shown decidable in several special cases [MTY05, GGMW13, Gim17, BFHH19]. The proof relies on a reduction from the Post Correspondence Problem, via an intermediary decision problem called the bipartite coloring problem. A direct reduction from the Post Correspondence Problem is of course also possible, but in our opinion it gives a less clear picture of the reasons for the undecidability.

The construction shows undecidability of the distributed control problem when the winning condition is local termination (every process eventually end up in a final state). Another natural condition in this setting is the *deadlock-freeness condition*, either global (the play is infinite) or local (every process plays infinitely often). The construction can be easily adapted to show undecidability for deadlock-freeness as well, using an encoding of the infinite version of the Post Correspondence Problem.

An interesting open problem is the decidability of the distributed control problem when the automaton is *grid-free*, i.e. when it does not allow two parallel runs of unbounded sizes both. The construction in this paper is clearly not grid-free: the two pools can run independently in parallel for an arbitrary long time before being synchronized by the environment. One cannot hope to rely on Monadic Second Order Logic (MSOL) to solve the problem, since MSOL is undecidable for grid-free systems [CC19]. However,

synthesis is easier than general logic: for example the DISTRIBUTED CONTROL PROBLEM is known to be decidable for four processes [Gim17] while MSOL is undecidable with only two processes [TY14].

The decidability of the DISTRIBUTED CONTROL PROBLEM for five processes remains an open question. Is it decidable like in the case of four processes? The decidability proof in [Gim17] relies on a small-model property of the winning strategies. When two processes do synchronize, they acquire simultaneously perfect knowledge about their respective states, hence an upper-bound on the number of transitions needed by two processes to win. When there are only four processes, two processes who synchronize can base their strategic decision on their current states and the boundedly many possible plays of the two other processes, hence a bound on the number of transitions needed by the four players to win. This property does not seem easy to get with five processes.

Is the DISTRIBUTED CONTROL PROBLEM undecidable for five processes, then? At first sight there is no easy way to adapt the present construction with two pools of three processes to show the undecidability for five processes. If the bottom pool is reduced to two processes only, it seems hard to enforce the lower-triangle constraints of the BIPARTITE COLORING PROBLEM. When the two sole processes of the bottom pool synchronize they acquire perfect knowledge of the state of their pool, and it is not possible anymore to constrain their future behaviour with respect to their past behaviour: what was done is gone.

## References


[BFHH19] Raven Beutner, Bernd Finkbeiner, and Jesko Hecking-Harbusch. Translating asynchronous games for distributed synthesis. In *30th International Conference on Concurrency Theory (CONCUR 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[CC19] Jérémie Chalopin and Victor Chepoi. 1-safe petri nets and special cube complexes: equivalence and applications. *ACM Transactions on Computational Logic (TOCL)*, 20(3):1–49, 2019.

[DR95] Volker Diekert and Grzegorz Rozenberg. *The Book of Traces*. World Scientific, 1995.

[FG18] Bernd Finkbeiner and Paul Gölz. Synthesis in Distributed Environments. In Satya Lokam and R. Ramanujam, editors, *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017)*, volume 93 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:14, Dagstuhl, Germany, 2018. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[FGHHO22] Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. Global winning conditions in synthesis of distributed systems with causal memory. In *30th EACSL Annual Conference on Computer Science Logic (CSL 2022)*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2022.

[FO17] Bernd Finkbeiner and Ernst-Rüdiger Olderog. Petri games: Synthesis of distributed systems with causal memory. *Information and Computation*, 253:181–203, 2017.

[FS05] Bernd Finkbeiner and Sven Schewe. Uniform distributed synthesis. In *Logic in Computer Science, 2005. LICS 2005. Proceedings. 20th Annual IEEE Symposium on*, pages 321–330. IEEE, 2005.

[GGMW13] Blaise Genest, Hugo Gimbert, Anca Muscholl, and Igor Walukiewicz. Asynchronous games over tree architectures. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II*, pages 275–286, 2013.

[Gim17] Hugo Gimbert. On the control of asynchronous automata. In Satya V. Lokam and R. Ramanujam, editors, *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2017, December 11-15, 2017, Kanpur, India*, volume 93 of *LIPIcs*, pages 30:1–30:15. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[GLZ04] Paul Gastin, Benjamin Lerman, and Marc Zeitoun. Distributed games with causal memory are decidable for series-parallel systems. In *FSTTCS 2004: Foundations of Software Technology and Theoretical Computer Science, 24th International Conference, Chennai, India, December 16-18, 2004, Proceedings*, pages 275–286, 2004.

[MTY05] P. Madhusudan, P. S. Thiagarajan, and Shaofa Yang. The MSO theory of connectedly communicating processes. In *FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science, 25th International Conference, Hyderabad, India, December 15-18, 2005, Proceedings*, pages 201–212, 2005.

[Mus15] Anca Muscholl. Automated synthesis of distributed controllers. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, pages 11–27, 2015.

[MW14] Anca Muscholl and Igor Walukiewicz. Distributed synthesis for acyclic architectures. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS 2014, December 15-17, 2014, New Delhi, India*, pages 639–651, 2014.

[MWZ09] Anca Muscholl, Igor Walukiewicz, and Marc Zeitoun. A look at the control of asynchronous automata. In M. Mukund K. Lodaya and eds. N. Kumar, editors, *Perspectives in Concurrency Theory*. Universities Press, CRC Press, 2009.

[Pos46] Emil L. Post. A variant of a recursively unsolvable problem. *Bull. Amer. Math. Soc.*, 52:264–268, 1946.

[PR90] Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *Foundations of Computer Science, 1990. Proceedings., 31st Annual Symposium on*, pages 746–757. IEEE, 1990.

[RW89] Peter JG Ramadge and W Murray Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

[Sch14] Sven Schewe. Distributed synthesis is simply undecidable. *Information Processing Letters*, 114(4):203–207, 2014.

[TY14] PS Thiagarajan and Shaofa Yang. Rabin's theorem in the concurrency setting: A conjecture. *Theoretical Computer Science*, 546:225–236, 2014.

[Zie87] Wieslaw Ziełonka. Notes on finite asynchronous automata. *ITA*, 21(2):99–135, 1987.