Synthesis in Distributed Environments*†

Bernd Finkbeiner[1] and Paul Gölz[2]

1 **Saarland University, Saarbrücken, Germany**
finkbeiner@react.uni-saarland.de
2 **Saarland University, Saarbrücken, Germany**
pgoelz@cs.cmu.edu

Abstract

Most approaches to the synthesis of reactive systems study the problem in terms of a two-player game with complete observation. In many applications, however, the system's environment consists of several distinct entities, and the system must actively communicate with these entities in order to obtain information available in the environment. In this paper, we model such environments as a team of players and keep track of the information known to each individual player. This allows us to synthesize programs that interact with a distributed environment and leverage multiple interacting sources of information.

The synthesis problem in distributed environments corresponds to solving a special class of Petri games, i.e., multi-player games played over Petri nets, where the net has a distinguished token representing the system and an arbitrary number of tokens representing the environment. While, in general, even the decidability of Petri games is an open question, we show that the synthesis problem in distributed environments can be solved in polynomial time for nets with up to two environment tokens. For an arbitrary but fixed number of three or more environment tokens, the problem is NP-complete. If the number of environment tokens grows with the size of the net, the problem is EXPTIME-complete.

1998 ACM Subject Classification F.3.1 Specifying and Verifying and Reasoning about Programs

Keywords and phrases reactive synthesis, distributed information, causal memory, Petri nets

Digital Object Identifier 10.4230/LIPIcs.FSTTCS.2017.28

1 Introduction

Automating the creation of programs is one of the most ambitious goals in computer science. Given a specification, a synthesis algorithm either generates a program that satisfies the specification or determines that no such program exists. The promise of synthesis is to let programmers work on a more abstract level and thus to fundamentally simplify the development of complex software.

Most current synthesis approaches (cf. [16, 5, 3, 15, 7]) are based on the game-theoretic approach, originally introduced by Büchi and Landweber [4], in which the synthesis problem is seen as a two-player game with complete observation, played between a *system* player and an *environment* player. The goal of the system player is to ensure that the specification is satisfied; the goal of the environment player is to ensure a violation. A winning strategy for the system player defines a control program that reads in the decisions of the environment as its inputs and produces the decisions of the system as its outputs.

* Supported by the European Research Council (ERC) Grant OSARES (No. 683300).
† Omitted proofs, notation for multisets and the algorithm mentioned in Theorems 6 and 9 can be found in the full version [11].

37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017).
Editors: Satya Lokam and R. Ramanujam; Article No. 28; pp. 28:1–28:14

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

A fundamental limitation of the standard game-theoretic formulation is that the environment is a monolithic block. In many applications, however, the environment consists of several distinct entities, and the system must actively communicate with these entities in order to obtain information available in the environment. In this paper, we introduce the *synthesis problem in distributed environments*. As in the standard approach, we view the synthesis problem as a game between the system and the environment. However, rather than considering the environment as a single *player* in this game, we consider it as a *team* consisting of several players that may carry different information. Both the individual environment players and the system player can increase their knowledge by interacting with other players.

The problem is related to, but very different from, the *distributed synthesis* problem [18]. In distributed synthesis, it is the *system* that is partitioned into multiple players, corresponding to multiple processes. The key difficulty here is to coordinate the strategies of the system players. In the synthesis problem in distributed environments, it is instead the *environment* that consists of multiple entities. The key difficulty here is for the system player to synchronize with the right environment players at the right points in time.

We study the synthesis problem in distributed environments in the framework of *Petri games* [12]. The players of a Petri game are represented as the tokens of a Petri net, partitioned into the system and environment players. Synthesis in distributed environments corresponds to Petri games with a single system token and multiple environment tokens. We assume that the underlying Petri net is bounded, i.e., only a bounded number of players can be generated over the course of a game. For unbounded nets, Petri games are known to be undecidable [12].

The players of a Petri game advance asynchronously except for synchronous interactions, in which players exchange knowledge. We assume that, whenever multiple players interact, they exchange information both truthfully and maximally. This model of knowledge is called *causal memory*. In this paper, we restrict our synthesis to *safety* specifications, i.e., the system must prevent the global state from entering certain bad configurations.

We illustrate our setting with a small access control example. Suppose you would like to synthesize a lock controller for a safe that contains sensitive business information. Corporate policy mandates that the safe may only be jointly opened by two employees and that both must previously have confirmed their identity with a corresponding authentication authority. The environment of the lock controller thus consists of four independent players: the employees e^1 and e^2 and their authenticators a^1 and a^2. These entities interact with each other (when a^1 authenticates e^1 or a^2 authenticates e^2) and with the system player (when e^1 or e^2 request the safe to open). Since there is no direct interaction between the lock controller and the authenticators, the knowledge about the authentication must be provided to the lock controller by the employees.[1]

Figure 1 shows how our access control scenario can be modeled as a Petri game. Players are represented by tokens (dots) that move between places (circles) using transitions (squares). The system player, who only moves between places marked in gray, starts in a place indicating that the safe is closed. The game allows her to consult with any employee and remain in her position, or to move to the place s_{open} to open the safe. The first employee starts in e^1 and can either directly move to e^1_{attempt} or can synchronize with her authenticator to move there. In the latter case, the authenticator simultaneously moves to a^1_{auth}, where

[1] In Petri games, all players are truthful. Think of the tokens as carriers of information, e.g., a cryptographically secured smart card carried by the employee.

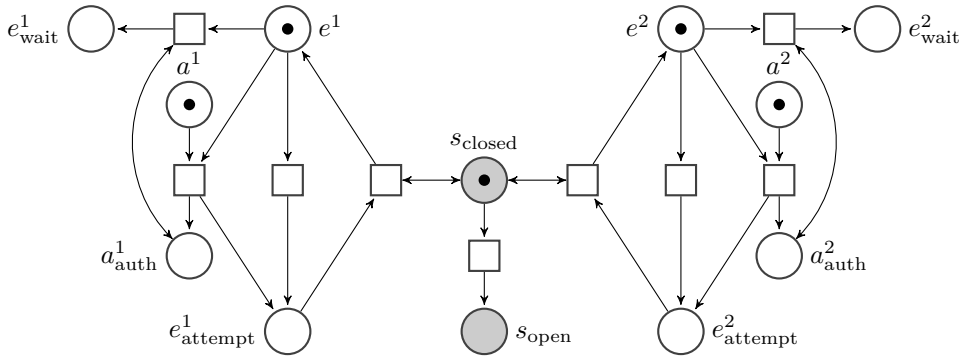

Figure 1 Petri game of the access control example. If the system player lies in s_{open} while there is still a player in a^1 or a^2, the system immediately loses the game.

she cannot authenticate e^1 a second time. When the employee is in e^1_{attempt}, the system player can choose to synchronize with her, moving the employee back to e^1 and exchanging all knowledge between the players. In particular, the system player learns whether the employee was authenticated. Afterwards, the employee can attempt to open the safe again, for example to make up for not being authenticated the last time. If the employee has already authenticated, she can alternatively move to e^1_{wait} and remain there. This possibility forces the locking mechanism to stop waiting for communication once it knows enough and to unlock the safe instead. The second employee is modeled symmetrically. To prevent the system from unlocking prematurely, we declare that all situations in which the safe is open but in which one authenticator has not moved yet as losing for the system.

A winning strategy for this game, as found by our synthesis algorithm, would be to allow communication with e^1 and nothing else until (possibly never) the system learns that the employee has authenticated. Then, it allows communication with e^2 until the same is true for the second employee. Finally, it opens the safe.

Related work. Synthesis in distributed environments is related to planning under partial observation [19, pp. 138–146] in that our strategies also combine information gathering and action. However, the classical partial-information setting does not capture the knowledge of different actors. With causal memory, a player's knowledge naturally refers to past observations and to the knowledge of other players. Synthesis in distributed environments can be expressed as a control problem [14, 17] for Zielonka's asynchronous automata [21]. Because this model is very expressive, all known decidability results assume strong restrictions on the communication architecture. Since our environment players are allowed to freely interact with each other and with the system, we cannot apply these results. Petri games were introduced in [12] and there is growing tool support for solving Petri games [10, 9]. The decidability of general Petri games is an open question. The only previously known decision procedure is restricted to the case of a single environment token [12]. In this paper, we solve the complementary case, where the number of environment tokens is unbounded (but there is only one system token). There is also a semi-algorithm for solving Petri games [8]. This approach finds finitely representable winning strategies, but does not terminate if no winning strategy exists.

Contributions. Our main technical contribution is an EXPTIME algorithm for deciding bounded Petri games with one system player and an arbitrary number of environment players.

Previously, the synthesis problem for Petri games with more than one environment player was open. We provide a matching lower bound to show that our algorithm is asymptotically optimal. If the number of environment players is kept constant, we show that the problem can be solved in polynomial time for up to two environment players whereas it is NP-complete for three or more environment players. The following table sums up the complexity of deciding k-bounded Petri games with one system player and e environment players, for any $k \geq 1$:

$e \leq 2$	P
$e \geq 3$	NP-complete
e grows with net	EXPTIME-complete

2 Petri nets

We recall notions from the theory of Petri nets as used in [12]. A tuple $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, In)$ is called a *Petri net* if it satisfies the following conditions:

- The set of *places* $\mathcal{P}$ and the set of *transitions* $\mathcal{T}$ are disjoint;
- The *flow relation* $\mathcal{F}$ is a multiset over $(\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P})$, i.e., $\mathcal{N}$ is a directed, bipartite multigraph with nodes $\mathcal{P} \cup \mathcal{T}$ and edges given by $\mathcal{F}$. We use the term *nodes* to refer to places and transitions simultaneously. For nodes x, y, we write $x \mathcal{F} y$ to denote $(x, y) \in \mathcal{F}$;
- The *initial marking* In is a finite multiset over $\mathcal{P}$;
- We require finite synchronization and nonempty pre- and postconditions: For a node x, define the *precondition* as a multiset $pre(x)$ such that $pre(x)(y) = \mathcal{F}(y, x)$ for all nodes y and similarly define the *postcondition* by $post(x)(y) = \mathcal{F}(x, y)$. Then, all transitions t must satisfy $0 < |pre(t)| < \infty$ and $0 < |post(t)| < \infty$.

A net is called *finite* if it contains finitely many nodes.

By convention, the components of a net $\mathcal{N}$ are named $\mathcal{P}$, $\mathcal{T}$, $\mathcal{F}$ and In, and similarly for nets named $\mathcal{N}_1$, $\mathcal{N}^\sigma$, $\mathcal{N}^U$, etc. We graphically specify Petri nets as multigraphs, where places are represented by circles, transitions by squares and the flow relation by arrows. In addition, the number of dots in a place reflects the multiplicity of this place in the initial marking. Apart from the gray color of certain places, Fig. 1 shows a Petri net with named places.

A *marking* $\mathcal{M}$ of $\mathcal{N}$ is a finite multiset over $\mathcal{P}$. We think of the Petri net as a board on which a finite number of *tokens* moves between places by using transitions. A marking then represents a certain configuration by listing the current number of tokens on every place. We can move from one marking to another by firing a transition t, i.e., by removing tokens in $pre(t)$ and putting tokens into $post(t)$ instead. If the total number of tokens changes in this process, we think of such transitions as generating or consuming tokens. We say that t is *enabled* in a marking $\mathcal{M}$ if $pre(t) \subseteq \mathcal{M}$. If this is the case, we can obtain a new marking $\mathcal{M}' := \mathcal{M} - pre(t) + post(t)$ by *firing* t, and we write $\mathcal{M} \,|t\rangle\, \mathcal{M}'$ to denote that $\mathcal{M}'$ can be constructed from $\mathcal{M}$ and t in this way. A marking is said to be *reachable* if it can be reached from the initial marking by firing a finite sequence of transitions. We generalize preconditions and postconditions to sets S of nodes by defining $pre(S) := \biguplus_{x \in S} pre(x)$ and analogously for $post(S)$. A Petri net is *k-bounded* for a natural number $k \geq 1$ if, for all reachable markings $\mathcal{M}$ and places p, $\mathcal{M}(p) \leq k$ holds. We call a net *bounded* if it is k-bounded for some k.

We are mainly interested in Petri nets as a model for the causal dependencies between events. These dependencies are made explicit in *occurrence nets*, certain acyclic nets in which each place has a unique causal history. Before giving their definition, we introduce notation to capture different kinds of causal relationships between nodes. We denote the transitive closure of the support of $\mathcal{F}$ by $<$ and its reflexive and transitive closure by $\leq$.

We call x and y *causally related* if $x \leq y$ or $y \leq x$. The *causal past* of a node x is the set $past(x) \coloneqq \{y \in \mathcal{P} \cup \mathcal{T} \mid y \leq x\}$. We extend this notion to sets S of nodes by setting $past(S) \coloneqq \bigcup_{x \in S} past(x)$. Apart from being causally related, two nodes x, y might also be mutually exclusive, i.e., they might be the result of alternative, nondeterministic choices. We say that x and y are *in conflict*, for short $x \sharp y$, if there exists a place p, $p \neq x, p \neq y$, such that x and y can be reached following the flow relation from p via different outgoing transitions. If x and y are neither causally related nor in conflict, we call them *concurrent*.

An *occurrence net* is a net $\mathcal{N}$ that satisfies all of the following conditions: the pre- and postconditions of transitions are sets, not general multisets; each place has at most one incoming transition; the initial marking is the set $\{p \in \mathcal{P} \mid pre(p) = \emptyset\}$; the inverse flow relation $\mathcal{F}^{-1}$ is well-founded, i.e., if we start from any node and follow the flow relation backwards, we eventually reach a place in the initial marking; no transition is in conflict with itself. Occurrence nets are 1-bounded, i.e., their reachable markings are sets.

We call a maximal set $\mathcal{C}$ of pairwise concurrent places in an occurrence net a *cut*. The *finite* cuts of an occurrence net are exactly its reachable markings [11, App. B.1]. Since the occurrence nets that we will work with only have finite cuts, we can use the terms interchangeably [11, Corollary 17].

A *homomorphism* from a Petri net $\mathcal{N}_1$ to a Petri net $\mathcal{N}_2$ is a function $\lambda : \mathcal{P}_1 \cup \mathcal{T}_1 \to \mathcal{P}_2 \cup \mathcal{T}_2$ that only maps places to places and transitions to transitions such that, for all $t \in \mathcal{T}_1$, $\lambda[pre(t)] = pre(\lambda(t))$ and $\lambda[post(t)] = post(\lambda(t))$. λ is called *initial* if additionally $\lambda[In_1] = In_2$ holds.

An *initial branching process* β of a net $\mathcal{N}$ is a pair $(\mathcal{N}^U, \lambda)$ where $\mathcal{N}^U$ is an occurrence net and λ is an initial homomorphism from $\mathcal{N}^U$ to $\mathcal{N}$ such that $\forall t_1, t_2 \in \mathcal{T}^U.\ (pre(t_1) = pre(t_2) \wedge \lambda(t_1) = \lambda(t_2)) \to t_1 = t_2$. Conceptually, a branching process describes a subset of the possible behavior of a net as an occurrence net. If a place or a transition in the original net can be reached on different paths or with different knowledge, the branching process splits up this node. The homomorphism λ is used to label those multiple instances with the original node in $\mathcal{N}$. The additional condition means that the branching process may not split up a transition unnecessarily: For the same precondition, at most one instance of a certain transition can be present in the branching process.

3 Petri games

In a Petri game, we partition the places of a finite Petri net into two disjoint subsets: the *system places* $\mathcal{P}_S$ (represented in gray) and the *environment places* $\mathcal{P}_E$ (represented in white). For convenience, we write $\mathcal{P}$ for the set of all places of the game $\mathcal{P}_S \cup \mathcal{P}_E$. A token on a system place represents a *system player*, a token on an environment place an *environment player*. Additionally, a Petri game also identifies a set of *bad markings* $\mathcal{B}$, which the system players need to avoid.[2] If the game reaches a marking $\mathcal{M}$ in $\mathcal{B}$, the environment wins; the system wins if this is never the case. Formally, a *Petri game* $\mathcal{G}$ is a tuple $(\mathcal{P}_S, \mathcal{P}_E, \mathcal{T}, \mathcal{F}, In, \mathcal{B})$. We call $\mathcal{N}^{\mathcal{G}} \coloneqq (\mathcal{P}, \mathcal{T}, \mathcal{F}, In)$ the *underlying net* of $\mathcal{G}$.

Transitions whose entire precondition belongs to the environment are called *purely environmental*. Otherwise, we call the transition a *system transition*.

[2] This is more general than in [12], where instead of avoiding a set of arbitrary markings, the system tries to avoid all markings that have a nonempty intersection with a set of bad places. [8] also uses arbitrary sets of bad markings. Since the hardness proofs in Theorems 7 and 8 only use bad markings of this shape, this generalization does not increase the computational hardness of our setting. In our complexity analyses in Theorems 6, 8 and 9, we do not commit to a specific input encoding of bad markings such that our results remain valid if a set of bad places is given instead.

Since Petri games aim to model the information flow in a system, a system player's decisions may only depend on information that she has witnessed herself or that she has obtained by communicating with other players. We thus describe strategies of the system as branching processes of the underlying net of the game, where the causal dependencies are made explicit. While the game is played on the underlying net, the strategy keeps track of the current state of the game as well as its causal history. Every reachable marking of the branching process corresponds to a reachable marking in the underlying net [11, Lemma 14]. However, the marking in the strategy might have less enabled transitions than the one in the underlying net, which means that the strategy can prevent certain transitions from firing. The game progresses by nondeterministically firing transitions that are allowed by the strategy. No matter which transitions are fired in which order, the system players need to ensure certain properties of the game. Because of this, it is sometimes useful to think of these choices as being made by an adversarial scheduler.

A winning, deadlock-avoiding *strategy* is an initial branching process $\beta_\sigma = (\mathcal{N}^\sigma, \lambda)$ of the underlying net of the game that satisfies the following four conditions:

justified refusal Let S be a set of pairwise concurrent places in $\mathcal{P}^\sigma$ and $\boldsymbol{t}$ be a transition *in the underlying net*, where $\lambda[S] = pre(\boldsymbol{t})$ but there is no $t \in \mathcal{T}^\sigma$ such that $\lambda(t) = \boldsymbol{t}$ and $pre(t) = S$. Then, there must be a place $s \in S \cap \mathcal{P}^\sigma_S$ such that $\boldsymbol{t} \notin \lambda(post(s))$.

safety For all $\mathcal{M} \in \mathcal{R}(\mathcal{N}^\sigma)$, $\lambda[\mathcal{M}] \notin \mathcal{B}$.

determinism For all $s \in \mathcal{P}^\sigma_S$ and all reachable markings $\mathcal{M}$ in $\mathcal{N}^\sigma$ that contain s, there is at most one transition $t \in post(s)$ that is enabled in $\mathcal{M}$.

deadlock avoidance For all $\mathcal{M} \in \mathcal{R}(\mathcal{N}^\sigma)$ we require that, if any transition of the underlying net is enabled in $\lambda[\mathcal{M}]$, then some transition in the strategy must be enabled in $\mathcal{M}$.

In the above conditions, we extended the notion of system places to the strategy by setting $\mathcal{P}^\sigma_S := \mathcal{P}^\sigma \cap \lambda^{-1}(\mathcal{P}_S)$. We similarly define the environment places of the strategy as $\mathcal{P}^\sigma_E := \mathcal{P}^\sigma \cap \lambda^{-1}(\mathcal{P}_E)$. To distinguish more clearly between nodes in the strategy and nodes in the underlying net, we always use bold variable names such as $\boldsymbol{p}$ or $\boldsymbol{t}$ for the latter.

Justified refusal means that a system player influences the course of the game by refusing to take part in certain transitions in her postcondition. Even if every place in $pre(\boldsymbol{t})$ contains a token for some $\boldsymbol{t} \in \mathcal{T}$, the transition can fire iff, for every place in $pre(\boldsymbol{t}) \cap \mathcal{P}_S$, the corresponding system player allows this transition. In particular, purely environmental transitions cannot be restricted by the strategy. More precisely, the condition refers to all possible preconditions S where a transition could have been added to the strategy, but was not. If no instance t of $\boldsymbol{t}$ with the right precondition exists so far, there must be a system place in S that refuses to take part in any instance of $\boldsymbol{t}$. Note that a system player can only refuse all transitions in the strategy with the label $\boldsymbol{t}$ or must allow all of them.

The safety objective requires that the game never reaches a bad marking. Determinism enforces that, from a system player's perspective, all sources of uncertainty are in the vicinity of an environment player. This does not prevent a system player from allowing multiple transitions, as long as these transitions are enabled in different markings.

Finally, we require the strategy to avoid deadlocks. Without this condition, a strategy might simply refuse to fire any system transition at all. In general, the system prefers to fire less transitions since they might potentially lead to bad markings and since allowing too many of them might cause nondeterminism. The criterion enforces that, whenever no purely environmental transition is enabled in a marking but some system transition is enabled, the strategy must allow one of them in order to keep the game going. This still allows the strategy to enter markings in which no transition is enabled at all. Similarly, a system player may refuse all transitions in her postcondition as long as she knows that the game will always allow another player to move.

$$\mathcal{V}_0' = \{(\mathcal{M}, \top, \{\boldsymbol{s_{\mathcal{M}}}\}) \mid \mathcal{M} \in \mathcal{R}(\mathcal{N})\}$$
$$\mathcal{V}_1' = \{(\mathcal{M}, c, R) \mid \mathcal{M} \in \mathcal{R}(\mathcal{N}); c \subseteq \mathit{post}(\boldsymbol{s_{\mathcal{M}}}); \boldsymbol{s_{\mathcal{M}}} \in R \subseteq \mathcal{M}\}$$
$$\mathcal{I}' = (\mathit{In}, \top, \{\boldsymbol{s_{In}}\})$$
$$\mathcal{E}' = \{(\mathcal{M}, \top, \{\boldsymbol{s_{\mathcal{M}}}\}) \to (\mathcal{M}, c, \{\boldsymbol{s_{\mathcal{M}}}\}) \mid c \subseteq \mathit{post}(\boldsymbol{s_{\mathcal{M}}})\} \tag{E'1}$$
$$\cup \left\{ (\mathcal{M}, c, R) \to (\mathcal{M}', c, R') \;\middle|\; \begin{array}{l} \boldsymbol{t} \text{ purely environmental transition}; \mathcal{M} \,|\boldsymbol{t}\rangle\, \mathcal{M}'; \\ \boldsymbol{o} \in \mathit{post}(\boldsymbol{t}); R' = R - \mathit{pre}(\boldsymbol{t}) + \{\boldsymbol{o}\} \end{array} \right\} \tag{E'2}$$
$$\cup \left\{ (\mathcal{M}, c, R) \to (\mathcal{M}', \top, \{\boldsymbol{s_{\mathcal{M}'}}\}) \;\middle|\; \begin{array}{l} \boldsymbol{t} \text{ system transition}; \boldsymbol{t} \in c; \mathcal{M} \,|\boldsymbol{t}\rangle\, \mathcal{M}'; \\ R \subseteq \mathit{pre}(\boldsymbol{t}) \end{array} \right\} \tag{E'3}$$
$$\mathcal{X}' = \{(\mathcal{M}, c, R) \mid \mathcal{M} \in \mathcal{B}\} \tag{X'1}$$
$$\cup \{(\mathcal{M}, c, R) \mid \boldsymbol{t}, \boldsymbol{t'} \in c; \boldsymbol{t} \neq \boldsymbol{t'}; \text{both enabled in } \mathcal{M}\} \tag{X'2a}$$
$$\cup \{(\mathcal{M}, c, R) \mid \boldsymbol{t} \in c; \text{enabled in } \mathcal{M}; 0 < \mathit{pre}(\boldsymbol{t})(\boldsymbol{p}) < \mathcal{M}(\boldsymbol{p}) \text{ for some } \boldsymbol{p} \in \mathcal{P}\} \tag{X'2b}$$
$$\cup \{(\mathcal{M}, c, R) \mid \text{Some } \boldsymbol{t} \in \mathcal{T} \text{ enabled; all such } \boldsymbol{t} \text{ involve the system and } \boldsymbol{t} \notin c\} \tag{X'3}$$

Figure 2 Description of the two graph games constructed from $\mathcal{G}$. For the components of $\mathit{Graph}(\mathcal{G})$, ignore all colored parts. Including them, we get the components of $\mathit{Graph}'(\mathcal{G})$.

4 Reduction to games over finite graphs

We wish to decide whether a k-bounded Petri game with one system player admits a winning, deadlock-avoiding strategy. In case of a positive answer, we also want to obtain a description of such a strategy. Note that the system player's decisions can be based on an unboundedly growing amount of information. Because of this, it is not at all obvious that the existence of a strategy is decidable and that strategies can be represented in finite space.

In this section and the next, we show that the decision problem is EXPTIME-complete in the size of the net. We establish the upper bound through a many-one reduction to a complete-observation game over a finite graph. We consider Petri games with a single system player, i.e., all reachable markings $\mathcal{M}$ contain exactly one system place, which we denote by $\boldsymbol{s_{\mathcal{M}}}$. In the cuts $\mathcal{C}$ of a strategy, we denote the unique system place by $s_{\mathcal{C}}$.

For a given Petri net $\mathcal{G}$ with underlying net $\mathcal{N}$, Fig. 2 defines the components of the translated graph game $\mathit{Graph}(\mathcal{G}) = (\mathcal{V}_0, \mathcal{V}_1, \mathcal{I}, \mathcal{E}, \mathcal{X})$ if we ignore all colored parts. The set of *vertices* $\mathcal{V}$ consists of two disjoint subsets $\mathcal{V}_0$ and $\mathcal{V}_1$, which describe the vertices belonging to players 0 and 1, respectively. The game begins in the initial vertex $\mathcal{I}$. From a vertex $v \in \mathcal{V}$, the current player chooses an outgoing *edge* in $\mathcal{E}$. A *play*, i.e., a maximal sequence $\mathcal{I} = v_0\, v_1\, \ldots$ of vertices with $(v_i, v_{i+1}) \in \mathcal{E}$ for all i, is winning for Player 0 if no vertex is an element of the bad vertices $\mathcal{X}$. A *strategy* T_σ (for Player 0) is a $\mathcal{V}$-labeled tree whose root is labeled with $\mathcal{I}$. If a node is labeled with a vertex in $\mathcal{V}_1$, its children are labeled with all successor vertices. Otherwise, it has a single child labeled with one particular successor. The strategy is *winning* if all maximal paths through it are labeled with winning plays. All such games are *memoryless determined*: If there is any winning strategy, there exists a winning strategy that selects, from any two nodes with the same label, the same successor vertex.

The vertices of the game essentially represent the reachable markings of the Petri game and Player 1 moves between markings by firing enabled transitions. This means that Player 1 plays the role of both the environment and the nondeterminism stemming from different schedulings. Player 0, who represents the system, can only act by refusing to allow some

transitions in the postcondition of the single system place in the marking. Since these decisions should not depend on scheduled, purely environmental transitions that the system would not yet know in the Petri game, Player 0 is forced to choose directly after the system player has taken a transition. Similarly to [12], we therefore add a *commitment*, i.e., a set $c \subseteq post(\boldsymbol{s_{\mathcal{M}}})$, to each vertex of the graph game. The commitment keeps track of the set of outgoing transitions of the current system place that the system player allows. Player 0's vertices are marked with $\top$ instead of a commitment to denote that she needs to decide on a commitment in the next step (E1). Player 1's choices are then restricted such that she can fire all purely environmental transitions (E2) but can only fire system transitions that appear in the commitment (E3). The bad vertices correspond to bad markings (X1), nondeterminism (X2a, X2b) and deadlock (X3).

To prove the reduction correct, we need to show that $\mathcal{G}$ has a winning, deadlock-avoiding strategy iff Player 0 has a winning strategy in $Graph(\mathcal{G})$. For this, we give translations between these types of strategies.

4.1 From Petri game strategies to graph game strategies

Assume that we are given a winning, deadlock-avoiding strategy $\beta_\sigma = (\mathcal{N}^\sigma, \lambda)$ for $\mathcal{G}$. We inductively build a strategy T_σ for $Graph(\mathcal{G})$. Whenever we encounter a node labeled with a vertex belonging to Player 0, we choose an outgoing edge, i.e., a suitable commitment.

For any such node, we look at the sequence of labels on the path that leads to it from the root. This sequence is a prefix of a play, which we denote by $v_0\, v_1\, \ldots\, v_r = (\mathcal{M}, \top)$. Edges of type (E1) in this prefix do not change the marking. All other edges are associated with firing a transition. Starting from the initial cut, we fire λ-preimages of these transitions one after another. If multiple transitions could be responsible for the edge or if multiple preimages are enabled, choose one canonically. For edges of type (E2), such preimages always exist because justified refusal does not allow β_σ to restrict purely environmental transitions. In the case of edges of type (E3), we make sure to only include transitions in the commitment if the existence of such preimages is ensured. By consecutively firing such a sequence of transitions, we reach a cut $\mathcal{C}$ such that $\lambda[\mathcal{C}] = \mathcal{M}$. Set $c \coloneqq \{\lambda(t) \mid t \in post(s_{\mathcal{C}})\}$ and choose the outgoing edge leading to $(\mathcal{M}, c)$ to construct the strategy.

For well-definedness, it remains to show that, when Player 1 schedules a system transition $\boldsymbol{t} \in c$ the next time, a preimage of this transition will be enabled in the cut $\mathcal{C}'$ that corresponds to the node in the strategy. Since, in between, only purely environmental transitions will be fired, $s_{\mathcal{C}}$ will still be part of $\mathcal{C}'$. The system place has a preimage of $\boldsymbol{t}$ in its postcondition by the definition of c. Therefore, a preimage enabled in $\mathcal{C}'$ exists by justified refusal.

▶ **Theorem 1.** *T_σ is a winning strategy for Player 0.*

Proof sketch (detailed in full version [11, B.2]). Consider a node n in T_σ with the label $(\mathcal{M}, c)$. As in the construction of the graph game strategy, we canonically fire transitions corresponding to the prefix until we reach a cut $\mathcal{C}$ such that $\lambda[\mathcal{C}] = \mathcal{M}$. Now assume that n is a bad vertex. Each kind of bad vertices (X1), (X2a), (X2b) or (X3) translates to a violation of the properties of a winning, deadlock-avoiding strategy in $\mathcal{C}$, contradiction. Thus, no node is labeled with a bad vertex and the strategy is winning. ◀

4.2 From graph game strategies to Petri game strategies

The converse direction is harder to prove. So far, we have shown that, if the system can win a Petri game with incomplete information, Player 0 can also win a game with full information on the marking graph. This is not surprising. In this step however, we must show that this

additional information does not give an advantage to Player 0 that the system does not have. In the construction of $\mathit{Graph}(\mathcal{G})$, we have already introduced commitments, which prevent Player 0 from using information about the scheduling of purely environmental transitions for her subsequent move. However, Player 0 might still use this information to make her move after the next. If the system player does not learn about the environment transition in her next step, this is an illegal flow of information.

The main idea now is that, while some parts of the graph game strategy do not correspond to a valid information flow in the Petri game, others do. In these latter parts, the strategy contains all necessary decisions to win the Petri game. Conceptually, we need to cut away unreasonable plays from the strategy. Alternatively, we might say that a forbidden information flow only happens if Player 1 does not play in an intelligent way. From Player 1's point of view, it is dangerous and unnecessary to schedule a purely environmental transition and then schedule a system transition unless the former is needed to enable the latter. If she does so, Player 0 gains potentially useful information, which Player 1 could easily prevent by scheduling the purely environmental transition at a later point, i.e., when it is necessary to enable the next system transition or when a winning situation for Player 1 (bad marking, nondeterminism or deadlock) can be reached without any more moves by Player 0. To make this idea formal, we construct another graph game $\mathit{Graph}'(\mathcal{G})$, which restricts Player 1's moves to enforce the behavior described above. Then, we can easily show that any winning strategy for $\mathit{Graph}(\mathcal{G})$ translates to a winning strategy for $\mathit{Graph}'(\mathcal{G})$, where Player 1 has fewer options. In a second step, we will translate the strategy from $\mathit{Graph}'(\mathcal{G})$ back to a strategy for the Petri game, which will prove the desired equivalence.

The new graph game $\mathit{Graph}'(\mathcal{G}) = (\mathcal{V}_0', \mathcal{V}_1', \mathcal{I}', \mathcal{E}', \mathcal{X}')$ is defined in Fig. 2 by taking into account the colored parts. The vertices of $\mathit{Graph}(\mathcal{G})$ are extended by a third component, a *responsibility multiset* R over $\mathcal{P}$. This multiset $R \subseteq \mathcal{M}$ tracks the information generated by firing transitions. At any point in the Petri game, a subset S of the cut such that $\lambda[S] = R$ together carries the information about all fired transitions. This notion is made precise in [11, Lemma 18]. After a transition has been fired, every token in its postcondition carries the information about the causal pasts of all participating tokens and about the fired transition itself. For this reason, when an edge of type (E'2) fires a purely environmental transition $\boldsymbol{t}$, the tokens in $\mathit{pre}(\boldsymbol{t})$ are subtracted from R, and Player 1 chooses an arbitrary token $\boldsymbol{o} \in \mathit{post}(\boldsymbol{t})$, which will carry the information to the system player. Edges of type (E'3) deal with R similarly in that they also subtract the precondition from R and instead add one element of the postcondition, namely the system place. In contrast to $\mathit{Graph}(\mathcal{G})$, these edges only allow system transitions if the responsibility multiset is included in the precondition, i.e., if the system player would directly learn about all previously scheduled transitions by taking this system transition.

▶ **Theorem 2.** *If there is a winning strategy for $\mathit{Graph}(\mathcal{G})$, there exists a winning strategy for $\mathit{Graph}'(\mathcal{G})$.*

Proof sketch (detailed in full version [11, B.3]). $\mathit{Graph}'(\mathcal{G})$ only reduces Player 1's options. ◀

We now translate a winning strategy T_σ for $\mathit{Graph}'(\mathcal{G})$ back into a winning, deadlock-avoiding strategy for the Petri game. Without loss of generality, we assume T_σ to be memoryless. We traverse the strategy tree in breadth-first order and inductively build the Petri game strategy $\beta_\sigma = (\mathcal{N}^\sigma, \lambda)$. Simultaneously, we map each node of the tree to a nonempty set of cuts. We call these cuts the *associated cuts* of the node. These cuts can be reached from In^σ by firing λ-preimages of transitions corresponding to the edges of types

(E'2) and (E'3) on the path from the root to this node. In particular, every such cut $\mathcal{C}$ will satisfy $\lambda[\mathcal{C}] = \mathcal{M}$, where $\mathcal{M}$ is the marking found in the label of the node.

We begin by mapping the root of the tree to a single cut In^σ, i.e., a fresh set of places such that $\lambda[In^\sigma] = In$. Then, we traverse T_σ and distinguish between the different kinds of edges in the graph game by which the vertex of the currently visited node has been reached from its predecessor.

- (E'1): Do not modify β_σ and map the new node to the same cuts as its parent.
- (E'2) or (E'3): Let $\mathcal{C}$ be one of the cuts associated with the parent node. Let $\boldsymbol{t}$ be a transition that could have been used in the definition of (E'2) or (E'3) to justify the existence of the edge. Finally, let B be any subset of $\mathcal{C}$ with $\lambda[B] = pre(\boldsymbol{t})$. Such a subset always exists because $\boldsymbol{t}$ is enabled in $\lambda[\mathcal{C}]$. If it already exists, let $t \in \mathcal{T}^\sigma$ be a transition with $pre(t) = B$ and $\lambda(t) = \boldsymbol{t}$. Else, create a new such transition and a fresh set of places as its postcondition such that $\lambda[post(t)] = post(\boldsymbol{t})$. Choose $\mathcal{C}'$ such that $\mathcal{C}\ |t\rangle\ \mathcal{C}'$. We map the new node to all cuts $\mathcal{C}'$ that can be constructed from suitable $\mathcal{C}$, $\boldsymbol{t}$ and B in this way.

We need to show that β_σ is a strategy. First, we can easily see that the construction ensures all requirements of an occurrence net. Furthermore, β_σ is an initial branching process because λ is an initial homomorphism and because we only add a new transition if no other transition with the same label and precondition exists.

Before we can prove that β_σ satisfies the four axioms of a winning, deadlock-avoiding strategy, we need to show that the responsibility multiset construction works as intended. First, we show that the construction prevents illegal information flows. Whenever the system player moves in the graph game, she directly learns about all previously scheduled transitions. Formally, nodes labeled with player-0 vertices are only mapped to cuts $\mathcal{C}$ that are the *last known cuts* of their respective system place $s_\mathcal{C}$. The last known cut of a place $x \in \mathcal{P}^\sigma$ is defined as $LKC(x) := \{p \in \mathcal{P}^\sigma \mid p \not< x \wedge \forall t \in pre(p).\ t < x\}$. In the terminology of [6], this cut is the *mapping cut* of $past(x) \cap \mathcal{T}$, i.e., the cut reached by firing all transitions in the past of x. The last known cut of x has the special property that, for every cut $\mathcal{C}$ with $x \in \mathcal{C}$, the last known cut of x lies in $past(\mathcal{C})$ [11, Lemma 20].

▶ **Lemma 3.** *Let a node in T_σ be labeled with a vertex belonging to Player 0 and let $\mathcal{C}$ be one of its associated cuts. Then, $\mathcal{C} = LKC(s_\mathcal{C})$.*

Proof in full version [11, B.4]. ◀

Second, we need to show that the responsibility multiset construction does not overly restrict the scheduling. For certain schedulings of purely environmental transitions, the responsibility multiset prevents a system transition from being fired even though it is enabled and in the commitment. If, since the Player 0's last move, Player 1 had skipped firing all transitions that do not help to enable this system transition, the transition could be fired. Therefore, the Petri game strategy contains all system transitions wherever they are not refused. This is formally stated and proved in [11, Lemma 21].

▶ **Lemma 4** (safety). *Let $\mathcal{C}$ be a cut in $\mathcal{N}^\sigma$. Then, $\lambda[\mathcal{C}] \notin \mathcal{B}$.*

Proof. Consider the node n for which $s_\mathcal{C}$ was inserted into the strategy. This node must be labeled with a $\mathcal{V}_0$ vertex and must have $LKC(s_\mathcal{C})$ as one of its associated cuts by Lemma 3. Since $LKC(s_\mathcal{C}) \subseteq past(\mathcal{C})$, there is a sequence of purely environmental transitions leading from $LKC(s_\mathcal{C})$ to $\mathcal{C}$ [11, Lemma 16]. Thus, from n's unique successor, we can follow a corresponding sequence of type-(E'2) edges to a node n' with $\mathcal{C}$ as one of its associated cuts. If $\lambda[\mathcal{C}]$ were a bad marking, n' would be labeled with a bad vertex of type (X'1). Since T_σ is a winning strategy, this is not the case. ◀

For the proofs of justified refusal (Lemma 24), determinism (Lemma 25) and deadlock avoidance (Lemma 26), we refer the reader to the full version [11]. As an immediate consequence, β_σ is a winning, deadlock-avoiding strategy, which concludes the claimed equivalence:

▶ **Theorem 5.** *If Graph′(𝒢) has a winning strategy for Player 0, there exists a winning, deadlock-avoiding strategy for 𝒢.*

5 Synthesis in distributed environments is EXPTIME-complete

▶ **Theorem 6.** *For fixed $k \geq 1$, k-bounded Petri games with one system player and an arbitrary number of environment players can be decided in exponential time.*

Proof. Our reduction allows to decide such Petri games $\mathcal{G}$ in exponential time: The number of vertices in $Graph(\mathcal{G})$ is bounded by $k^{|\mathcal{P}|} \cdot (2^{|\mathcal{T}|} + 1)$ and its local structure can be computed efficiently. Since graph games with such safety winning conditions can be solved in linear time in the size of the game [1, pp. 78–79], this requires exponential time in the size of the Petri game.

In [11, App. C.1], we describe an algorithm that evaluates the commitments symbolically and uses a SAT solver to speed up solving the game in practice. If we solve the SAT instances through a naïve enumeration, we have an explicit EXPTIME algorithm, whose complexity is analyzed in [11, App. C.2]. ◀

▶ **Theorem 7.** *Deciding k-bounded Petri games with one system player and an arbitrary number of environment players is EXPTIME-hard for any $k \geq 1$.*

Proof sketch (detailed in full version [11, B.7]). We show hardness through a reduction from the EXPTIME-complete combinatorial game G_5 from [20]. This reduction is similar to the one given in [12] for the fragment with one environment player. In G_5, two players, P_S and P_E, take turns in switching the truth values of a finite set of Boolean variables, one at a time. Alternatively, they are allowed to pass. The players operate on disjoint subsets of the variables. Initially, the variables have predefined values. If, at a certain point, a formula ϕ over the variables becomes satisfied, P_E wins; else, P_S wins.

For an instance of this game, we build a Petri game such that there is a winning, deadlock-avoiding strategy iff P_S has a winning strategy in the original game. Without loss of generality, let ϕ be given in negation normal form. An example for the reduction is illustrated in [11, Fig. 4]. Each variable is represented by an environment token moving between two places, indicating the variable's truth value. An additional environment token keeps track of the current turn. If it is P_E's turn, this token synchronizes with one of the environment variables and switches its position. If it is P_S's turn, the token first informs the single system token of the previous moves and then enables the transitions for switching a system variable, from which the system token chooses one.

Instead of letting a player move, the turn token can permanently freeze the variables and prove that ϕ is satisfied. For this, we have an additional environment token for every subformula, each with two places. The turn token can move these tokens to their second place to prove that the subformula is satisfied. For literals, the turn token needs to synchronize with the respective variable in the correct place. For disjunctions, it must synchronize with the token of one of the subformulas, which must have been proved before. For conjunctions, synchronization with both subformula tokens is required. The bad markings are exactly those in which the entire formula ϕ is proved. This game is 1-bounded, thus k-bounded. ◀

6 Sparse Petri games

The nets produced by our EXPTIME-hardness reduction contain a high number of environment tokens. Because of this, the number of reachable markings grows exponentially and computational cost with it. To study other sources of algorithmic hardness, we analyze the complexity of the problem for a fixed maximum number p of environment players. Then, we can bound the number of reachable markings by the polynomial $(|\mathcal{P}|+1)^{(p+1)}$ instead of by $(k+1)^{|\mathcal{P}|}$. For a fixed p, the problem is in NP: We nondeterministically guess a commitment for every $\mathcal{V}_0$ vertex and verify in polynomial time that no bad vertices are reachable.

▶ **Theorem 8.** *For a fixed $p \geq 3$, deciding Petri games with one system player and p environment players is NP-complete.*

Proof sketch (detailed in full version [11, B.8]). The upper bound has already been established. Show the lower bound by a reduction from the Boolean satisfiability problem with 3-clauses (3SAT). For a given instance, construct a Petri game with three environment players and a single system player. For every clause, the single system player must allow at least one transition corresponding to a satisfied literal in the clause. Deadlock avoidance forces the system player to allow at least one such transition per clause. Nondeterminism prevents the system player from allowing two transitions corresponding to complementary literals. ◀

▶ **Theorem 9.** *Petri games with one system player and at most two environment players can be decided in polynomial time.*

Proof sketch (detailed in full version [11, B.9]). We adapt the algorithm in [11, App. C.1], which evaluates commitments symbolically with a SAT solver. Due to the special structure of the SAT instances generated, we can add pre- and postprocessing steps such that the SAT queries only contain 2-clauses. Since 2SAT can be solved in polynomial time [2], this yields a polynomial-time decision procedure. ◀

7 Conclusions

In this paper, we have developed algorithms for the synthesis of reactive systems in distributed environments. We have studied the problem in the setting of Petri games. Previously, the decidability of Petri games was only known for non-distributed environments, i.e., for games with a single environment token [12]. Our algorithms solve Petri games with one system token and an arbitrary number of environment tokens. We have shown that the synthesis problem can be solved in polynomial time for nets with up to two environment tokens. For an arbitrary but fixed number of three or more environment tokens, the problem is NP-complete. If the number of environment tokens grows with the size of the net, the problem is EXPTIME-complete.

An intriguing question for future work is whether our results, which scale to an arbitrary number of environment tokens, can be combined with the results of [12], which scale to an arbitrary number of system tokens. This would allow us to synthesize "distributed systems in distributed environments." With the algorithm presented in this paper, we can already synthesize individual components in such distributed systems, by treating the other components as adversarial (cf. [13]). The approach of [12] would additionally allow us to analyze the cooperation between the system components.

References

1 Krzysztof R. Apt and Erich Grädel. *Lectures in game theory for computer scientists*. Cambridge University Press, 2011.

2 Bengt Aspvall, Michael F. Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Inf. Process. Lett.*, 8(3):121–123, 1979. doi:10.1016/0020-0190(79)90002-4.

3 Aaron Bohy, Véronique Bruyère, Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Acacia+, a tool for LTL synthesis. In P. Madhusudan and Sanjit A. Seshia, editors, *Computer Aided Verification - 24th International Conference, CAV 2012, Berkeley, CA, USA, July 7-13, 2012 Proceedings*, volume 7358 of *Lecture Notes in Computer Science*, pages 652–657. Springer, 2012. doi:10.1007/978-3-642-31424-7_45.

4 J. Richard Buchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969. doi:10.2307/1994916.

5 Rüdiger Ehlers. Unbeast: Symbolic bounded synthesis. In Parosh Aziz Abdulla and K. Rustan M. Leino, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 17th International Conference, TACAS 2011, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2011, Saarbrücken, Germany, March 26-April 3, 2011. Proceedings*, volume 6605 of *Lecture Notes in Computer Science*, pages 272–275. Springer, 2011. doi:10.1007/978-3-642-19835-9_25.

6 Javier Esparza. Model checking using net unfoldings. *Sci. Comput. Program.*, 23(2-3):151–195, 1994. doi:10.1016/0167-6423(94)00019-0.

7 Peter Faymonville, Bernd Finkbeiner, Markus N. Rabe, and Leander Tentrup. Encodings of bounded synthesis. In Axel Legay and Tiziana Margaria, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I*, volume 10205 of *Lecture Notes in Computer Science*, pages 354–370, 2017. doi:10.1007/978-3-662-54577-5_20.

8 Bernd Finkbeiner. Bounded synthesis for petri games. In Roland Meyer, André Platzer, and Heike Wehrheim, editors, *Correct System Design - Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday, Oldenburg, Germany, September 8-9, 2015. Proceedings*, volume 9360 of *Lecture Notes in Computer Science*, pages 223–237. Springer, 2015. doi:10.1007/978-3-319-23506-6_15.

9 Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. Symbolic vs. bounded synthesis for Petri games. In Dana Fisman and Swen Jacobs, editors, *6th Workshop on Synthesis (SYNT 2017)*. EPTCS, 2017. URL: https://www.react.uni-saarland.de/publications/FGHO17.pdf.

10 Bernd Finkbeiner, Manuel Gieseking, and Ernst-Rüdiger Olderog. Adam: Causality-based synthesis of distributed systems. In Daniel Kroening and Corina S. Pasareanu, editors, *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, volume 9206 of *Lecture Notes in Computer Science*, pages 433–439. Springer, 2015. doi:10.1007/978-3-319-21690-4_25.

11 Bernd Finkbeiner and Paul Gölz. Synthesis in distributed environments. *CoRR*, abs/1710.05368, 2017. arXiv:1710.05368.

12 Bernd Finkbeiner and Ernst-Rüdiger Olderog. Petri games: Synthesis of distributed systems with causal memory. *Inf. Comput.*, 253:181–203, 2017. doi:10.1016/j.ic.2016.07.006.

13 Bernd Finkbeiner and Sven Schewe. Semi-automatic distributed synthesis. In Doron A. Peled and Yih-Kuen Tsay, editors, *Automated Technology for Verification and Analysis,*

Third International Symposium, ATVA 2005, Taipei, Taiwan, October 4-7, 2005, Proceedings, volume 3707 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 2005. doi:10.1007/11562948_21.

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

15 Swen Jacobs, Roderick Bloem, Romain Brenguier, Ayrat Khalimov, Felix Klein, Robert Könighofer, Jens Kreber, Alexander Legg, Nina Narodytska, Guillermo A. Pérez, Jean-François Raskin, Leonid Ryzhyk, Ocan Sankur, Martina Seidl, Leander Tentrup, and Adam Walker. The 3rd reactive synthesis competition (SYNTCOMP 2016): Benchmarks, participants & results. In Ruzica Piskac and Rayna Dimitrova, editors, *Proceedings Fifth Workshop on Synthesis, SYNT@CAV 2016, Toronto, Canada, July 17-18, 2016.*, volume 229 of *EPTCS*, pages 149–177, 2016. doi:10.4204/EPTCS.229.12.

16 Barbara Jobstmann, Stefan J. Galler, Martin Weiglhofer, and Roderick Bloem. Anzu: A tool for property synthesis. In Werner Damm and Holger Hermanns, editors, *Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, July 3-7, 2007, Proceedings*, volume 4590 of *Lecture Notes in Computer Science*, pages 258–262. Springer, 2007. doi:10.1007/978-3-540-73368-3_29.

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

18 Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II*, pages 746–757. IEEE Computer Society, 1990. doi:10.1109/FSCS.1990.89597.

19 Stuart Russell and Peter Norvig. *Artificial Intelligence – A Modern Approach*. Pearson, 3rd edition, 2009.

20 Larry J. Stockmeyer and Ashok K. Chandra. Provably difficult combinatorial games. *SIAM J. Comput.*, 8(2):151–174, 1979. doi:10.1137/0208013.

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