Global Winning Conditions in Synthesis of Distributed Systems with Causal Memory

Bernd Finkbeiner
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Manuel Gieseking
University of Oldenburg, Oldenburg, Germany

Jesko Hecking-Harbusch
CISPA Helmholtz Center for Information Security, Saarbrücken, Germany

Ernst-Rüdiger Olderog
University of Oldenburg, Oldenburg, Germany

Abstract

In the synthesis of distributed systems, we automate the development of distributed programs and hardware by automatically deriving correct implementations from formal specifications. For synchronous distributed systems, the synthesis problem is well known to be undecidable. For asynchronous systems, the boundary between decidable and undecidable synthesis problems is a long-standing open question. We study the problem in the setting of Petri games, a framework for distributed systems where asynchronous processes are equipped with causal memory. Petri games extend Petri nets with a distinction between system places and environment places. The components of a distributed system are the players of the game, represented as tokens that exchange information during each synchronization. Previous decidability results for this model are limited to local winning conditions, i.e., conditions that only refer to individual components.

In this paper, we consider global winning conditions such as mutual exclusion, i.e., conditions that refer to the state of all components. We provide decidability and undecidability results for global winning conditions. First, we prove for winning conditions given as bad markings that it is decidable whether a winning strategy for the system players exists in Petri games with a bounded number of system players and one environment player. Second, we prove for winning conditions that refer to both good and bad markings that it is undecidable whether a winning strategy for the system players exists in Petri games with at least two system players and one environment player. Our results thus show that, on the one hand, it is indeed possible to use global safety specifications like mutual exclusion in the synthesis of distributed systems. However, on the other hand, adding global liveness specifications results in an undecidable synthesis problem for almost all Petri games.

2012 ACM Subject Classification Theory of computation

Keywords and phrases Synthesis, distributed systems, reactive systems, Petri games, decidability

Digital Object Identifier 10.4230/LIPIcs.CSL.2022.20

Related Version *Full Version*: `https://arxiv.org/abs/2107.09280` [12]

Funding Supported by the German Research Foundation (DFG) Grant Petri Games (392735815) and Collaborative Research Center "Foundations of Perspicuous Software Systems" (TRR 248, 389792660), and by the European Research Council (ERC) Grant OSARES (683300).

1 Introduction

The *synthesis problem* probes whether there exists an implementation for a formal specification and derives such an implementation if it exists. This approach automates the creation of systems. Engineers can think on a more abstract level about *what* a system should achieve instead of *how* the system should achieve its goal. The synthesis problem for a system

consisting of one component interacting with its environment is often encoded as a two-player game with complete observation between the *system* player and the *environment* player (cf. [24, 4, 2, 7, 29, 25, 23]). The system player tries to satisfy the winning condition of the game while the environment player tries to violate it. A winning strategy for the system player is a correct implementation as it encodes the system's reaction to all environment behaviors.

The synthesis problem for *distributed systems* aims to derive a correct implementation for every concurrent component of a distributed system. Each component can interact with its environment. Distributed systems can be differentiated depending on whether the components progress *synchronously* or *asynchronously*. For synchronous distributed systems, the synthesis problem is well known to be undecidable, as observed by Pnueli and Rosner [34]. For asynchronous distributed systems with causal memory, the boundary between decidable and undecidable synthesis problems is a long-standing open question [30, 15]. For the synthesis of asynchronous distributed systems, the memory model changes compared to the two-player game and the number of players increases to encode the different components of the system. In distributed systems, components observe only their local surroundings. This can be encoded by *causal memory* [16, 27, 17]: Two players share no information while they run concurrently; during every synchronization, however, they exchange their entire local histories, including all of their previous synchronizations with other players.

In this paper, we consider *reactive systems*, i.e., the components continually interact with their environment. *Control games* [17] based on *asynchronous automata* [39] and *Petri games* [15] are formalisms for the synthesis of asynchronous distributed reactive systems with causal memory. We focus on Petri games. Here, several system players play against several environment players in a Petri net. Tokens represent players and places either belong to the system or to the environment, resulting in a distribution of system and environment players. Deciding the existence of a winning strategy for the system players is EXPTIME-complete for Petri games with a bounded number of system players, one environment player, and bad places as local winning condition [15]. This also holds for Petri games with a bounded number of environment players, one system player, and bad markings as global winning condition [14]. Local winning conditions cannot express global properties like mutual exclusion.

We consider global winning conditions and contribute decidability and undecidability results regarding the synthesis of asynchronous distributed reactive systems with causal memory. In the first part of this paper, we prove that it is decidable whether a winning strategy for the system players exists in Petri games with a bounded number of system players, one environment player, and *bad markings* as global winning condition. Bad markings are a *safety* winning condition in the sense that they define markings as bad that the system players have to avoid in order to win the Petri game. Decidability is achieved by a reduction to a two-player game with complete observation and a Büchi winning condition. In the two-player game, it is encoded that transitions with the environment player fire as late as possible, i.e., transitions *without* the environment player fire before transitions *with* it. This order of transitions encodes causal memory [15]. For every sequential play of the two-player game, we need to check that no bad marking is reached for the different orders of fired concurrent transitions. The causal history of system players can grow infinitely large. We show that the finite causal history of each system player until its last synchronization with the environment player can be stored finitely and suffices to find bad markings.

In the second part of this paper, we investigate whether global winning conditions beyond bad markings are decidable. We report on two undecidability results to further underline the significance of our decidability result: We prove that it is undecidable whether a winning strategy exists for the system players in Petri games with at least two system players, one

environment player, and *good and bad markings* as winning condition. For this winning condition, no bad marking should be reached *until* a good marking is reached, which can be expressed in linear-time temporal logic (LTL) [33]. Notice that it is not required to terminate in a good marking. Good markings can be used to simulate the undecidable synchronous setting of Pnueli and Rosner [34] in the asynchronous setting of Petri games. This is realized by identifying executions as good if players deviate too much from the synchronous setting. Next, we prove that it is undecidable whether a winning strategy exists for the system players in Petri games with *good markings* and at least three players, out of which one is an environment player and each of the other two can change between being a system and an environment player. Good markings are a *liveness* winning condition in the sense that they define markings as good, one of which the system players have to reach in order to win the Petri game. Here, bad markings from the first undecidability result are encoded by repeatedly changing all players to environment players. With these results, we obtain an overview regarding decidability and undecidability for global winning conditions.

Related Work A formal connection exists between Petri games and *control games* [17] based on *asynchronous automata* [39]: Petri games can be translated into control games and vice versa, at an exponential blow-up in each direction [1]. This translates decidability in acyclic communication architectures [17], originally obtained for control games, to Petri games, and decidability in single-process systems [14], originally obtained for Petri games, to control games. Further decidability results exist for control games with acyclic communication architectures [31]. Decidability has also been obtained for restrictions on the dependencies of actions [16] or on the synchronization behavior [26, 27] and for decomposable games [19].

The decidability result of this paper does not transfer to control games because the translation in [1] produces Petri games with as many system *and* environment players as there are processes in the control game. The undecidability results of this paper transfer to control games. System players in Petri games correspond to processes with only controllable actions in control games; environment players correspond to processes with only uncontrollable actions [1]. Bad markings from the first undecidability result can be simulated by additional uncontrollable actions for all processes preventing the reaching of good markings afterward.

For Petri games with several system and environment players, bounded synthesis is a semi-decision procedure to find winning strategies for the system players [8, 22, 21]. Bounded synthesis and the reduction for bad places are implemented in the tool AdamSYNT [13, 9, 18].

2 Motivating Example

We introduce the intuition behind Petri games and bad markings with the example in Fig. 1. There, we search for a strategy for two power plants, which should react to the energy production of renewable sources based on the weather forecast. A Petri game differentiates the places of a Petri net as *system* places (depicted in gray) and as *environment* places (depicted in white). For example, p is a system place whereas *forecast* is an environment place. The players of a Petri game are represented by tokens. The type of the place, where a token is residing, dictates whether the token represents a system or an environment player.

After transition *sunny* fires to indicate a sunny forecast, there are two system players in place p (each representing one power plant) and one environment player in place s. Causal memory implies that both system players know what the weather forecast predicts. They do not know whether the actual energy production is high (indicated by s_h firing) or low (indicated by s_l firing) producing three or two units of energy in place w. Nevertheless, each

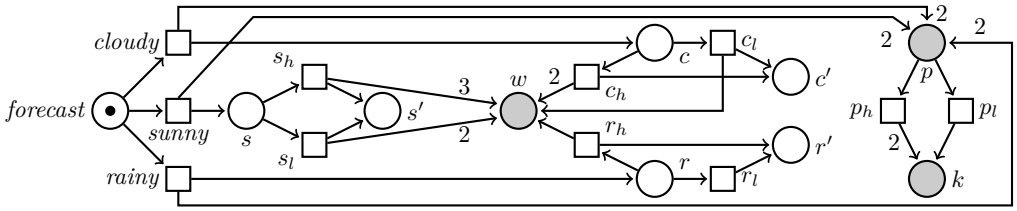


Figure 1 Two power plants observe if *sunny*, *cloudy*, or *rainy* weather is *forecast*. Depending on the actual weather, renewable sources produce up to three units of energy. The power plants produce one or two units of energy each and have to maintain the total energy production between four and five units of energy as all final markings with different energy production are bad markings.

power plant has to decide whether to produce two or one unit of energy in place k by p_h or p_l firing. The two power plants should produce together with the renewable sources either four or five units of energy. Therefore, any final marking resulting in a different energy production is a bad marking, i.e., the set of bad markings is $\{M : \mathcal{P} \to \mathbb{N} \mid (M(k) + M(w) < 4 \vee M(k) + M(w) > 5) \wedge \exists x \in \{s', c', r'\} : (M(x) = 1 \wedge \forall y \in \mathcal{P} \setminus \{x, k, w\} : M(y) = 0)\}$. The second conjunct ensures the marking being final by requiring that the only environment player is in one of the three environment places s', c', and r' and the system players are only in system places k and w. We assume that players always choose one of their successors. In Sec. 4, the finer notion of strategies being *deadlock-avoiding* is presented.

A winning strategy for the system players produces one unit of energy at both power plants for a sunny forecast, two units of energy at one power plant and one unit of energy at the other for a cloudy forecast, and two units of energy at both power plants for a rainy forecast. The specification is expressible with the local winning condition of bad places by having transitions from each bad marking leading to a bad place. This is only so as the example has no infinite behavior. For Petri games with infinite behavior and one environment player, the global winning condition of bad markings can specify losing behavior between players without requiring their synchronization which is impossible for local winning conditions.

3 Petri Nets

A *Petri net* [32, 37] $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, In)$ consists of the disjoint finite sets of *places* $\mathcal{P}$ and of *transitions* $\mathcal{T}$, the *flow relation* $\mathcal{F}$ as multiset over $(\mathcal{P}\times\mathcal{T})\cup(\mathcal{T}\times\mathcal{P})$, and the *initial marking* In as multiset over $\mathcal{P}$. For a place p, the *precondition* is the set $pre(p) = \{t \in \mathcal{T} \mid \mathcal{F}(t,p) > 0\}$ and the *postcondition* is the set $post(p) = \{t \in \mathcal{T} \mid \mathcal{F}(p,t) > 0\}$. For a transition t, the *precondition* is the multiset over $\mathcal{P}$ defined by $pre(t)(p) = \mathcal{F}(p,t)$ for all $p \in \mathcal{P}$ and the *postcondition* is the multiset over $\mathcal{P}$ defined by $post(t)(p) = \mathcal{F}(t,p)$ for all $p \in \mathcal{P}$. States of Petri nets are represented by multisets over $\mathcal{P}$, called *markings*. A marking M puts $M(p)$ *tokens* in every place $p \in \mathcal{P}$. A transition t is *enabled* in a marking M if $pre(t) \subseteq M$. If no transition is enabled in a marking M, then M is called *final*. An enabled transition t can *fire* in a marking M resulting in the successor marking $M' = M - pre(t) + post(t)$ (written $M[t\rangle M'$). For markings M and M', we write $M[t_0, \ldots, t_{n-1}\rangle M'$ if there exist markings $M_0, \ldots, M_n$ such that $M_0 = M$, $M_n = M'$, and $M_i[t_i\rangle M_{i+1}$ for $0 \leq i < n$. The set of *reachable markings* of $\mathcal{N}$ is defined as $\mathcal{R}(\mathcal{N}) = \{M \mid \exists n \in \mathbb{N}, t_0, \ldots, t_{n-1} \in \mathcal{T} : In[t_0, \ldots, t_{n-1}\rangle M\}$. A net $\mathcal{N}'$ is a *subnet* of $\mathcal{N}$ (written $\mathcal{N}' \sqsubseteq \mathcal{N}$) if $\mathcal{P}' \subseteq \mathcal{P}$, $\mathcal{T}' \subseteq \mathcal{T}$, $In' = In$, and $\mathcal{F}' = \mathcal{F} \restriction (\mathcal{P}'\times\mathcal{T}')\cup(\mathcal{T}'\times\mathcal{P}')$. We enforce $In' = In$ to maintain all players when later defining strategies for Petri games.

We call elements in $\mathcal{P}\cup\mathcal{T}$ *nodes*. For nodes x and y, we write $x \lessdot y$ if $x \in pre(y)$. With $\leq$, we denote the reflexive, transitive closure of $\lessdot$. The *causal past* of x is $past(x) = \{y \mid y \leq x\}$.

Nodes x and y are *causally related* if $x \leq y \vee y \leq x$. They are *in conflict* (written $x \sharp y$) if, for a place $p \in \mathcal{P}$, there are distinct transitions $t_1, t_2 \in post(p)$ with $t_1 \leq x \wedge t_2 \leq y$. Node x is in *self-conflict* if $x \sharp x$. We call x and y *concurrent* if they are neither causally related nor in conflict.

An *occurrence net* is a Petri net where the pre- and postcondition of transitions are sets, the initial marking coincides with places without ingoing transitions, other places have exactly one ingoing transition, no infinite path starting from any given node and following the inverse flow relation exists, and no transition is in self-conflict. A *homomorphism* maps nodes from $\mathcal{N}_1$ to $\mathcal{N}_2$ preserving the type of nodes and the pre- and postcondition of transitions.

A branching process [5, 28, 6] describes parts of possible behaviors of a Petri net. We use the *individual token semantics* [20]. A *branching process* of a Petri net $\mathcal{N}$ is a pair $\iota = (\mathcal{N}^\iota, \lambda^\iota)$ where $\mathcal{N}^\iota$ is an occurrence net and $\lambda^\iota : \mathcal{P}^\iota \cup \mathcal{T}^\iota \to \mathcal{P} \cup \mathcal{T}$ is a homomorphism from $\mathcal{N}^\iota$ to $\mathcal{N}$ that is injective on transitions with the same precondition. Intuitively, whenever a node can be reached on two distinct paths in a Petri net $\mathcal{N}$, it is split up in the branching process of $\mathcal{N}$. λ^ι labels the nodes of $\mathcal{N}^\iota$ with the original nodes of $\mathcal{N}$. The injectivity condition avoids additional unnecessary splits. The *unfolding* $\iota_U = (\mathcal{N}^U, \lambda^U)$ of $\mathcal{N}$ is a maximal branching process: Whenever there is a set of pairwise concurrent places C such that $\lambda^U[C] = pre^{\mathcal{N}}(t)$ for some transition $t \in \mathcal{T}$, then there exists $t' \in \mathcal{T}^U$ with $\lambda^U(t') = t$ and $pre^U(t') = C$.

4 Petri Games and Büchi Games

A *Petri game* [15, 14] is a tuple $\mathcal{G} = (\mathcal{P}_S, \mathcal{P}_E, \mathcal{T}, \mathcal{F}, In, \mathcal{W})$. The places of the underlying Petri net $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, In)$ are partitioned into *system places* $\mathcal{P}_S$ and *environment places* $\mathcal{P}_E$. We call tokens on system places *system players* and tokens on environment places *environment players*. The game is played by firing transitions in $\mathcal{N}$. Players *synchronize* when a joint transition fires. Intuitively, a strategy controls the behavior of system players by deciding which transitions to allow. Environment players are uncontrollable and transitions only dependent on environment players cannot be restricted. The *winning condition* is given by $\mathcal{W}$ as the set of *bad places* $\mathcal{P}_B \subseteq \mathcal{P}$, *bad markings* $\mathcal{M}_B \subseteq \mathcal{R}(\mathcal{N})$, or *good markings* $\mathcal{M}_G \subseteq \mathcal{R}(\mathcal{N})$ or the pair of disjoint sets of *good and bad markings* $(\mathcal{M}_G, \mathcal{M}_B) \in \mathbb{P}(\mathcal{R}(\mathcal{N})) \times \mathbb{P}(\mathcal{R}(\mathcal{N}))$. We depict Petri games as Petri nets and color system places gray and environment places white. A Petri game has *a bounded number of system players and one environment player* if, for a bound $k \in \mathbb{N}$, every system place contains at most k tokens for all reachable markings of $\mathcal{N}$ and the sum of tokens in all environment places is exactly one for all reachable markings of $\mathcal{N}$.

A *strategy* for $\mathcal{G}$ is a branching process $\sigma = (\mathcal{N}^\sigma, \lambda^\sigma)$ of $\mathcal{N}$ satisfying *justified refusal*: If there is a set of pairwise concurrent places C in $\mathcal{N}^\sigma$ and a transition $t \in \mathcal{T}$ with $\lambda^\sigma[C] = pre^{\mathcal{N}}(t)$, then there either is a transition $t' \in \mathcal{T}^\sigma$ with $\lambda^\sigma(t') = t$ and $C = pre^\sigma(t')$ or there is a system place $p \in C \cap (\lambda^\sigma)^{-1}[\mathcal{P}_S]$ with $t \notin \lambda^\sigma[post^\sigma(p)]$. Justified refusal enforces that only system places can prohibit transitions based on their causal past: From every situation in the game, a transition possible in the underlying net is either in the strategy or there is a system place that never allows it. A strategy is a restriction of possible transitions in the Petri game because it is a branching process which describes subsets of the behavior of a Petri net. We further require σ to be *deterministic*: For every reachable marking M of σ and system place $p \in M$, there is at most one transition from $post^\sigma(p)$ enabled in M. Notice that $post^\sigma(p)$ can contain more than one transition as long as at most one of them is enabled in the same reachable marking. This allows that the environment player decides between different branches of the Petri game and the system player later on reacts to every decision.

A strategy is *deadlock-avoiding* if, for every final, reachable marking M in the strategy, $\lambda^\sigma[M]$ is final as well. A strategy σ is *winning for bad places* $\mathcal{W} = \mathcal{P}_B$ if it is deadlock-avoiding

and no reachable marking in σ contains a place corresponding to a bad place. A strategy σ is *winning for bad markings* $\mathcal{W} = \mathcal{M}_B$ if it is deadlock-avoiding and no reachable marking in σ corresponds to a bad marking. One branching process $\iota^1 = (\mathcal{N}^1, \lambda^1)$ is a *subprocess* of another $\iota^2 = (\mathcal{N}^2, \lambda^2)$ if $\mathcal{N}^1 \sqsubseteq \mathcal{N}^2$ and $\lambda^1 = \lambda^2 \restriction (\mathcal{P}^1 \cup \mathcal{T}^1)$. A *play* $\pi = (\mathcal{N}^\pi, \lambda^\pi)$ is a subprocess of a strategy $\sigma = (\mathcal{N}^\sigma, \lambda^\sigma)$ with $\forall p \in \mathcal{P}^\pi : |post(p)| \leq 1$. It is *maximal* if, for each set of pairwise concurrent places C in $\mathcal{N}^\pi$ with $C = pre^\sigma(t)$ for some $t \in \mathcal{T}^\sigma$, a place $p \in C$ and a transition $t' \in \mathcal{T}^\pi$ exist with $t' \in post^\pi(p)$. A *complete firing sequence* of a play π is a possibly infinite sequence of fired transitions such that each transition of π occurs. A strategy σ is *winning for good markings* $\mathcal{W} = \mathcal{M}_G$ if, for all complete firing sequences $t_0 t_1 t_2 \ldots$ of all maximal plays π of σ with $M_0 = In^\pi$ and $M_0[t_0\rangle M_1[t_1\rangle M_2[t_2\rangle \ldots$, there exists $i \geq 0$ with $\lambda^\pi[M_i] \in \mathcal{M}_G$. A strategy σ is *winning for good and bad markings* $\mathcal{W} = (\mathcal{M}_G, \mathcal{M}_B)$ if, for all complete firing sequences $t_0 t_1 t_2 \ldots$ of all maximal plays π of σ with $M_0 = In^\pi \wedge M_0[t_0\rangle M_1[t_1\rangle M_2[t_2\rangle \ldots$, there exists $i \geq 0$ with $\lambda^\pi[M_i] \in \mathcal{M}_G \wedge \forall 0 \leq j < i : \lambda^\pi[M_j] \notin \mathcal{M}_B$. Terminating in a final marking as winning condition is different from reaching a good marking as players are not required to terminate in a good marking and can reach a bad marking afterward.

A *Büchi game* has two players: *Player 0* represents the *system*, *Player 1* the *environment*. Both act on complete information about the game arena and the play so far. To win, Player 0 has to ensure that an accepting state is visited infinitely often. A winning strategy for Player 0 corresponds to a correct implementation of the encoded synthesis problem. Deciding the existence of a winning strategy can be done in polynomial time [3].

Formally, a *Büchi game* $\mathbb{G} = (V, V_0, V_1, I, E, F)$ consists of the finite set of *states* V partitioned into the disjoint sets of states V_0 of Player 0 and of states V_1 of Player 1, the *initial state* $I \in V$, the *edge relation* $E \subseteq V \times V$, and the set of *accepting states* $F \subseteq V$. We assume that all states in a Büchi game have at least one outgoing edge. A *play* is a possibly infinite sequence of states which is constructed by letting Player 0 choose the next state from the successors in E whenever the game is in a state from V_0 and by letting Player 1 choose otherwise. An *initial play* is a play that starts from the initial state. A play is *winning* for Player 0 if it visits at least one accepting state infinitely often. Otherwise, the play is *winning* for Player 1. A *strategy* for Player 0 is a function $f : V^* \cdot V_0 \to V$ that maps plays ending in states of Player 0 to one possible successor according to E. A play *conforms* to a strategy f if all successors of states in V_0 are chosen in accordance with f. A strategy f is winning for Player 0 if all initial plays that conform to f are winning for Player 0.

5 Decidability in Petri Games with Bad Markings

We present a reduction from Petri games with a bounded number of system players, one environment player, and bad markings to Büchi games. In the following, we give an intuition for the main concepts of the reduction, before presenting the structure of the Büchi game in the remainder of this section. More details are in [12] and a running example is in Fig. 2.

Petri games use unfoldings, which can be of infinite size, to encode the causal memory of players. By contrast, Büchi games have two players with complete information and a finite number of states. To overcome these differences when encoding Petri games, states in the corresponding Büchi games consist of a representation of the current marking and some additional information. Edges in the Büchi game mostly correspond to a transition firing in the Petri game. We say that a transition fires in the Büchi game when it fires in the encoded Petri game. Concurrency between transitions in the Petri game is encoded by having most possible interleavings in the Büchi game. Some interleavings are left out to encode causal memory of the players in Petri games: Causal memory is simulated in Büchi games

by transitions with an environment place in their precondition firing as late as possible at *mcuts* [15]. An mcut is a situation in the Petri game where all system players have progressed maximally, i.e., the environment player can choose between all remaining possible transitions. Mcuts can only be defined for Petri games with at most one environment player. Player 1 in the Büchi game makes decisions only at states corresponding to mcuts.

We make two key additions to ensure that the idea of firing transition with the environment player only at mcuts can be lifted from local winning conditions to global winning conditions: First, we add *backward moves* to detect bad markings and nondeterministic decisions. Intuitively, backward moves allow us to rewind transitions with only system players participating. They are realized by each system player remembering its history until its last synchronization with the environment player. In every state of the Büchi game, it is checked whether the backward moves of all system players allow us to rewind the game in such a way that a bad marking is reached or a nondeterministic decision is found.

Second, we add the so-called *NES-case* to handle system players playing infinitely without synchronizing with the environment player directly in the Büchi game. The abbreviation NES stands for *no more environment synchronization* and is necessary when some system players play infinitely but without synchronization with the environment player. In [15], this situation is called the type-2 case and can be handled as a preprocessing step, because only the local winning condition of bad places is considered. This is impossible for the global winning condition of bad markings considered in this paper. Throughout this paper, the NES-case can be disregarded by adding the restriction that each system player either terminates or synchronizes infinitely often with the environment player.

For the NES-case, every system player has a three-valued flag. As long as the system player will terminate or will synchronize with the environment player in the future, the flag should be set to *negative NES-status*. When system players can play infinitely without synchronizing with the environment player, they should set their flags to *positive NES-status*. After the NES-case, participating system players obtain an *ended NES-status*, which excludes them from the remaining Büchi game. A positive NES-status triggers the NES-case. Here, the system players with positive NES-status have to prove that they can play infinitely without synchronizing with the environment player. Therefore, the usual order of all transitions without the environment player being possible until reaching an mcut is interrupted. Instead, only system players with positive NES-status are considered until their proof of playing infinitely without the environment player is successful. If the system players with positive NES-status make a mistake in their proof, then Player 0 immediately loses the Büchi game.

5.1 States and Initial State in the Büchi Game

Decision tuples represent players of the Petri game in states in the Büchi game. A *decision tuple* for a player consists of an *identifier*, a *position*, a *NES-status*, a *decision*, and a *representation of the last mcut*. The identifier uniquely determines the player. The position gives the current place of the player. System players with negative NES-status *false* claim that they will terminate or fire a transition with an environment place in its precondition and are not part of the NES-case. In the NES-case, system players go from positive NES-status *true* to ended NES-status *end* as described previously.

The decision is either $\top$ or the set of *allowed transitions* by the player. For system players, $\top$ indicates that a decision for a set of allowed transitions is missing and has to be chosen. The representation of the last mcut encodes the last known position of the environment player. There can be at most as many different such positions as there are system players. Thus, a number suffices to identify the last known mcut. Let $\max_S$ be the

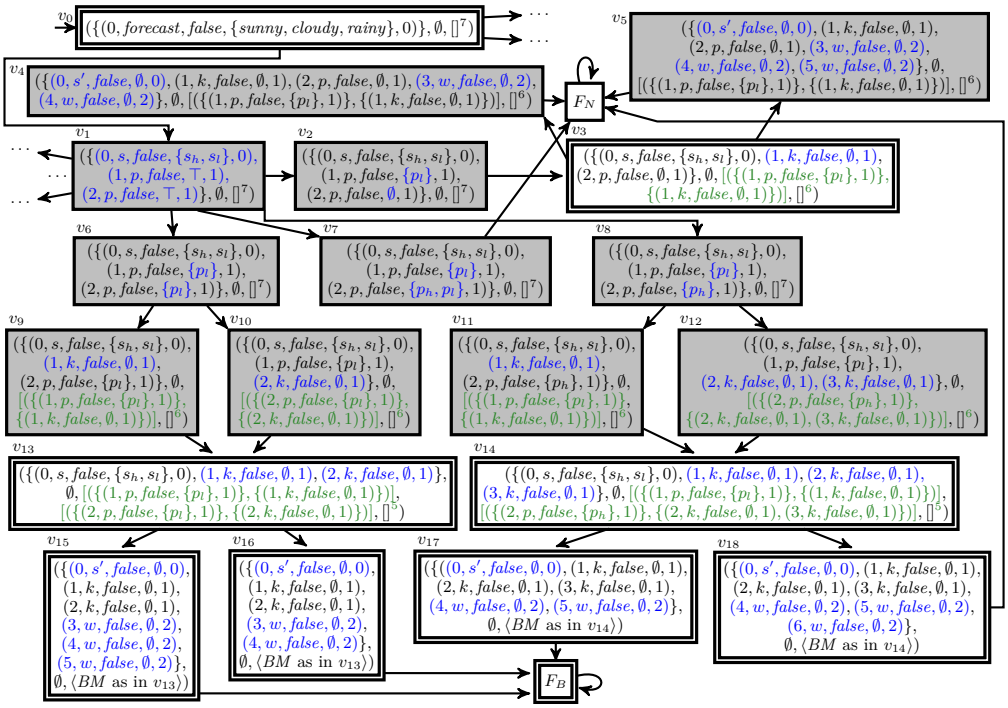
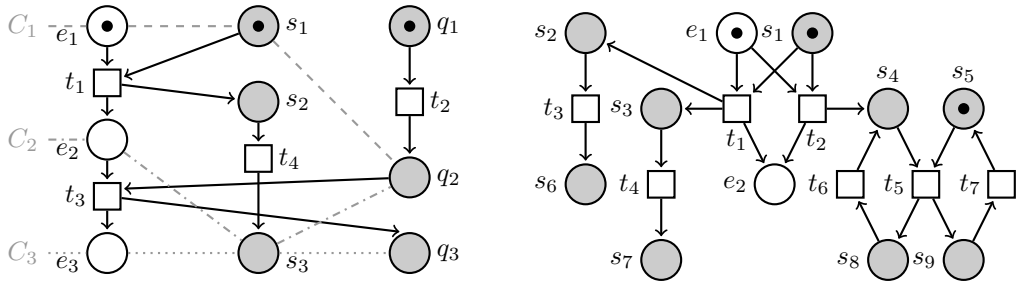


Figure 2 Part of the Büchi game for the Petri game in Fig. 1 is given. States of Player 0 are gray, states of Player 1 white. Most states are labeled for identification. Double squares are accepting states. Changes from previous states are blue for decision tuples and green for backward moves.

maximal number of system players in the Petri game which are visible at the same time. The set of *system decision tuples* is $\mathcal{D}_S = \{(id, p, b, T, K) \mid id, K \in \{1, \ldots, \max_S\} \wedge p \in \mathcal{P}_S \wedge b \in \{false, true, end\} \wedge (T = \top \vee T \subseteq post(p))\}$, the set of *environment decision tuples* is $\mathcal{D}_E = \{(0, p, false, post(p), 0) \mid p \in \mathcal{P}_E\}$, and the set of all *decision tuples* is $\mathcal{D} = \mathcal{D}_S \cup \mathcal{D}_E$.

▶ **Example 1.** In Fig. 2, a branch of the Büchi game for the Petri game in Fig. 1 is shown. States with decision tuples with positive NES-status are omitted because no infinite behavior occurs. The initial state v_0 has one decision tuple for the environment player in place *forecast* and empty information for the NES-case and the backward moves. After Player 1 plays the edge for transition *sunny* firing, state v_1 with three decision tuples is reached. The decision tuples for the two system players in place p have $\top$ as decision. There are 16 combinations of decisions by the two system players, out of which four are shown. The first system player always allows transition p_l and the second system player allows no transition in v_2, only one of the two transitions p_l and p_h in v_6 and v_8, or both transitions in v_7.

Almost all states in the Büchi game contain decision tuples and additional information for the NES-case and for backward moves. The *states in the Büchi game* are defined as $V = V_{BN} \cup \mathscr{D} \times (\mathcal{P}_S \to \{0, \ldots, k\}) \times (\mathcal{B}^*)^{\max_S}$ with $V_{BN} = \{F_B, F_N\}$. Finite winning and losing behavior in the Petri game is represented in the Büchi game by the two unique states F_B and F_N in V_{BN}. A *decision marking* is a set of decision tuples corresponding to a reachable marking in the Petri game such that each identifier occurs at most once. $\mathscr{D}$ is the set of all such decision markings. The next element stores the underlying multiset over system places of the decision marking from the start of the NES-case restricted to system players with positive NES-status. In the NES-case, repeating this multiset proves that the



(a) The mcuts of the unfolding are C_1, C_2, and C_3. **(b)** Markings containing s_2 and s_7 are bad markings.

Figure 3 Two Petri games illustrate mcuts, backward moves, and the NES-case.

system players with positive NES-status can play infinitely without firing a transition with an environment place in its precondition. This element is the empty multiset if not in the NES-case. More details are in Sec. 5.5. $\mathcal{B} : \mathbb{P}(\mathcal{D}_S) \times \mathbb{P}(\mathcal{D}_S)$ is the set of backward moves to detect states corresponding to a bad marking or a nondeterministic decision. The remaining elements are $\max_S$ sequences of backward moves. Each identifier in a decision tuple maps to the position of a sequence of backward moves. More details are in Sec. 5.4.

The *initial state in the Büchi game* has as many decision tuples with unique identifier, NES-status *false*, $\top$ as decision, and last mcut 1 as there are tokens in system places in *In* of the Petri game and one decision tuple with identifier 0, NES-status *false*, the postcondition of p_E as decision, and last mcut 0 for the one environment place p_E with one token in *In*. The other parts are the empty multiset or the empty sequence of backward moves.

5.2 States of Player 0, States of Player 1, and Accepting States

Causal memory in Petri games is encoded in Büchi games by letting Player 0 fix the decisions of allowed transitions for system players as early as possible and having Player 1 fire transitions with an environment place in their precondition as late as possible at mcuts. *Cuts* are markings in unfoldings. An *mcut* is a cut where all enabled transitions have an environment place in their precondition, i.e., all system players progressed maximally on their own. With Fig. 3a, we illustrate mcuts. The initial cut $\{e_1, s_1, q_1\}$ is *not* an mcut as the enabled transition t_2 has only the system place q_1 in its precondition. After t_2 fires, the cut $C_1 = \{e_1, s_1, q_2\}$ is an mcut as the only enabled transition t_1 has environment place e_1 in its precondition. Analog arguments lead to $\{e_2, s_2, q_2\}$ not being an mcut and $C_2 = \{e_2, s_3, q_2\}$ being an mcut. The final cut $C_3 = \{e_3, s_3, q_3\}$ is an mcut as there are no enabled transitions.

A decision marking $\mathbb{D}$ in the states in the Büchi game *corresponds to an mcut* when no $\top$ and no positive NES-status are part of $\mathbb{D}$ and every transition with only system places in its precondition is not enabled or not allowed by a participating system player in $\mathbb{D}$. A state in the Büchi game can correspond to an mcut although the cut in the unfolding of the Petri game is not an mcut as the decisions of the system players in the Büchi game can disallow transitions. *States of Player 1* are F_B, F_N, and states corresponding to an mcut. *States of Player 0* are all other states. *Accepting states* are F_B and states corresponding to an mcut.

▶ **Example 2.** The Petri game from Fig. 1 has $\{forecast\}$, $\{\{e, k : i\} \mid e \in \{s, c, r\} \wedge 2 \leq i \leq 4\}$, and $\{\{s', w : w_{s'}, k : i\}, \{c', w : w_{c'}, k : i\}, \{r', w : w_{r'}, k : i\} \mid 2 \leq w_{s'} \leq 3 \wedge 1 \leq w_{c'} \leq 2 \wedge 0 \leq w_{r'} \leq 1 \wedge 2 \leq i \leq 4\}$ as mcuts, i.e., the initial cut, cuts where the power plants produced energy while the energy production by renewable sources was not selected, and all final, reachable cuts. In the Büchi game in Fig. 2, the eight states v_0, v_3, and v_{13} to v_{18} of Player 1

have decision markings that correspond to an mcut. For states v_0, v_{13}, and v_{14}, all enabled transitions have an environment place in their precondition. For states v_{15} to v_{18}, each decision marking corresponds to a final cut. The decision marking of state v_3 corresponds to an mcut as the second system player in p decided to not allow any of its outgoing transitions.

5.3 Edges in the Büchi Game

Edges in the Büchi game mostly connect states $\mathcal{V} = (\mathbb{D}, M_{T2}, BM_1, \ldots, BM_{\max_S})$ and $\mathcal{V}' = (\mathbb{D}', M'_{T2}, BM'_1, \ldots, BM'_{\max_S})$ where $\mathbb{D}$ is a decision marking, M_{T2} is a marking, and $BM_1, \ldots, BM_{\max_S}$ are as many sequences of backward moves as the maximum number $\max_S$ of system players in the Petri game. There are five sets of edges *TOP*, *SYS*, *NES*, *MCUT*, and *STOP*. In the following description of the five sets of edges, not mentioned elements of the connected states stay the same. The formal definitions can be found in [12].

(1) Edges from *TOP* occur from states where at least one decision tuple in $\mathbb{D}$ has $\top$ as decision. To obtain $\mathbb{D}'$, Player 0 replaces each $\top$ in the decision tuples of system players with a set of allowed transitions and can change the NES-status of decision tuples for system players from *false* to *true*. The underlying marking of decision tuples with positive NES-status *true* is stored in M'_{T2} when a NES-status changes.

(2) Edges from *SYS* occur from states where all decision tuples in $\mathbb{D}$ have negative NES-status and at least one transition with only system places in its precondition is enabled and allowed by the decision tuples in $\mathbb{D}$. To get $\mathbb{D}'$, Player 0 simulates one such transition t firing by removing decision tuples $\mathbb{D}_{pre}$ for the precondition of t and adding decision tuples $\mathbb{D}_{post}$ for the postcondition of t. For $\mathbb{D}_{post}$, the last mcut of all participating players is the maximum of their previous values and Player 0 picks the decisions and can change the NES-status as in (1). Marking M'_{T2} is obtained as in (1). Backward move $(\mathbb{D}_{pre}, \mathbb{D}_{post})$ is added to BM_{id} of all participating players with identifier *id* to get BM'_{id}.

(3) Edges from *NES* are the NES-case and occur from states where a decision tuple in $\mathbb{D}$ has positive NES-status. To obtain $\mathbb{D}'$, Player 0 fires a transition as in (2) but only from decision tuples with positive NES-status resulting in new decision tuples with positive NES-status. This includes the storage of backward moves. The NES-case is successful if the marking M_{T2} is reached again and all players in it moved. Then, decision tuples with NES-status *true* are set to NES-status *end* and M'_{T2} becomes the empty marking.

(4) Edges from *MCUT* occur from states where all enabled and allowed transitions have an environment place in their precondition. To get $\mathbb{D}'$, Player 1 fires one such transition. Decision tuples for the precondition of the transition are removed, decision tuples for the postcondition are added. Added decision tuples for system players have NES-status *false*, $\top$ as decision, an empty sequence of backward moves, and the highest last mcut. As backward moves store the past of system players until their last mcut, backward moves for system players that are part of the transition are removed. If backward moves become never applicable by firing the transition, they are removed from the successor state.

(5) Edges from *STOP* occur from states with no transition enabled or corresponding to losing behavior. They replace other outgoing edges for losing behavior. States corresponding to termination lead to the winning state F_B. States corresponding to a deadlock but not termination lead to the losing state F_N. If backward moves detect a bad marking or a nondeterministic decision, the state leads to F_N. In the NES-case, a synchronization of decision tuples with positive and negative NES-status or a deadlock or vanishing of decision tuples with positive NES-status leads to F_N. Decision tuples with positive NES-status can vanish when transitions with empty postcondition fire.

▶ **Example 3.** In Fig. 2, outgoing edges of state v_1 are in *TOP*. Outgoing edges of states v_2, v_6, and v_8 to v_{12} are in *SYS*. Outgoing edges of states v_0, v_3, v_{13}, and v_{14} are in *MCUT*. Other edges are in *STOP*. No edges in *NES* exist in the depicted part. Outgoing edges of states v_4 and v_5 represent the deadlock of the second system player in p disallowing both outgoing transitions while only they are enabled. The outgoing edge of state v_7 encodes a nondeterministic decision of the second system player, which allows two enabled transitions. Such a decision is only useful if another player ensures that at most one of the transitions becomes enabled. Outgoing edges of states v_{15} to v_{17} represent termination. The outgoing edge of state v_{18} represents a bad marking for six produced units of energy.

When, as in our construction, (I) Player 0 immediately resolves $\top$ to the decisions of system players, (II) Player 0 decides which transitions with only system places in their precondition fire following the decisions of system players, and (III) Player 1 decides as late as possible at mcuts which transitions with an environment place in their precondition fire following the decisions of system players, then the corresponding Büchi games encode causal memory [15]. Allowed transitions with only system places in their precondition fire in an order determined by Player 0 until an mcut is reached. There, Player 1 decides for the environment player which allowed transition to fire. Afterward, this process repeats itself.

5.4 Backward Moves in the Büchi Game

In the Büchi game, Player 0 can avoid markings by picking the firing order for transitions with only system places in their precondition. In Fig. 3b, the two system players in s_2 and s_3 are reached after t_1 fires. One can fire t_3, the other t_4. This results in the firing sequences $t_1t_3t_4$ and $t_1t_4t_3$. If s_2 and s_7 are in a bad marking, then Player 0 can decide for edges corresponding to the first firing sequence and the bad marking is missed. We introduce backward moves to avoid such problems. A *backward move* is a pair of decision markings. It stores the change to the decision tuples by edges from *SYS* and *NES*. For every such edge from $\mathcal{V} = (\mathbb{D}, M_{T2}, BM_1, \ldots, BM_{\max_S})$ to $\mathcal{V}' = (\mathbb{D}', M'_{T2}, BM'_1, \ldots, BM'_{\max_S})$, we obtain $\mathbb{D}_{pre}$ and $\mathbb{D}_{post}$ with $\mathbb{D}' = (\mathbb{D} \setminus \mathbb{D}_{pre}) \cup \mathbb{D}_{post}$ and add backward move $(\mathbb{D}_{pre}, \mathbb{D}_{post})$ to the end of BM_{id} of all participating players with identifier id.

For every state $\mathcal{V}'$ in the Büchi game, it is checked with backward moves if $\mathcal{V}'$ is losing due to a bad marking or a nondeterministic decision. The decision marking $\mathbb{D}'$ and all decision markings that are reachable via backward moves are checked. Therefore, it is checked whether backward moves $(\mathbb{D}_{pre}, \mathbb{D}_{post})$ are *applicable* to $\mathbb{D}'$, i.e., whether $\mathbb{D}_{post} \subseteq \mathbb{D}'$ and $(\mathbb{D}_{pre}, \mathbb{D}_{post})$ is the last backward move of all participating players. In this case, the backward move is removed from the end of the sequences of backward moves of all participating players and $\mathbb{D} = (\mathbb{D}' \setminus \mathbb{D}_{post}) \cup \mathbb{D}_{pre}$ results from the application of the backward move. The underlying marking of $\mathbb{D}$ is checked to not be a bad marking and $\mathbb{D}$ is checked to have only deterministic decisions. This is repeated recursively from $\mathbb{D}$ for all applicable backward moves until no backward move is applicable. If a decision marking corresponding to a bad marking or a nondeterministic decision is detected, the current state $\mathcal{V}'$ only has an edge to F_N.

The identifier of players in decision tuples is used to map the decision tuple to the corresponding sequence of backward moves, i.e., for each system player in the Petri game, the Büchi game collects a sequence of backward moves. Edges from *MCUT* empty the sequence of backward moves of decision tuples when their system place is in the precondition of the fired transition. This removal can make backward moves not applicable because some participating players do not have the backward move as their last one anymore.

The sequence of backward moves can grow infinitely long when system players play

infinitely without the environment player and without the NES-case. This would result in a Büchi game with infinitely many states. To avoid this, the Büchi game becomes losing for Player 0 when it plays in a way that corresponds to a strategy with a variant of useless repetitions [19] for the system players in the Petri game. Our variant of useless repetitions identifies the repetition of a loop consisting only of transitions without the environment player in their precondition such that the last mcut of the system players does not change, i.e., the system players repeat a loop in which they do not exchange any new information about the environment player. Thus, winning strategies have to avoid playing a useless repetition more than once between the successor of an mcut and the next mcut. This can be achieved either by continuing to the next mcut or by setting some players to NES-status *true* and completing the NES-case, i.e., playing infinitely without the environment player.

▶ **Example 4.** In Fig. 2, we include the collection of backward moves. State v_{13} represents each power plant producing one unit of energy after a sunny weather forecast. It is reached from state v_6 either via state v_9 or v_{10} depending on which power plant produces energy first. State v_{13} has a backward move for each power plant: $(\{(1, p, \mathit{false}, \{p_l\}, 1)\}, \{(1, k, \mathit{false}, \emptyset, 1)\})$ and $(\{(2, p, \mathit{false}, \{p_l\}, 1)\}, \{(2, k, \mathit{false}, \emptyset, 1)\})$. Because the three markings $\{s, k : 2\}$ (underlying marking of v_{13}), $\{s, p, k\}$ (applying one backward move), and $\{s, p : 2\}$ (applying both backward moves) are no bad markings and all decisions are deterministic, state v_{13} continues with edges for the transitions of the environment place s instead of having an edge to F_N.

5.5 Encoding the NES-Case Directly in the Büchi Game

We handle the *NES-case* where system players play infinitely without firing a transition with an environment place in its precondition directly in the Büchi game as players in the NES-case might be in a bad marking. This is in contrast to the reduction for bad places [15].

In the Büchi game, Player 0 has to reach an accepting state infinitely often in order to win the game. Only F_B and states corresponding to an mcut are accepting states. Transitions with only system places in their precondition are fired between successors of mcuts and the following mcut. Thus, if the system players can fire transitions with only system places in their precondition infinitely often, eventually a useless repetition is reached which is losing. To overcome this, we give Player 0 the possibility to change the NES-status for decision tuples of system players from negative to positive. The underlying marking of this change is stored and afterward only transitions from decision tuples with positive NES-status can be fired. Firing these transitions maintains the positive NES-status for new decision tuples. Instead of firing infinitely many transitions, the NES-case is ended if the stored marking is reached again and all players in the marking have moved. In this case, the NES-status of all decision tuples with positive NES-status is changed to ended NES-status and the Büchi game continues with the remaining decision tuples with negative NES-status. The requirement to move is necessary as otherwise too many players could get ended NES-status. Decision tuples with ended NES-status are maintained as backward moves can be applicable to them, i.e., backward moves store the NES-status and allow us to reverse it in search for a bad marking. We can thus ensure that continuing with the case where all decision tuples have negative NES-status avoids bad markings that span the NES-case.

Player 0 has to disclose decision tuples with positive NES-status if system players fire infinitely many transitions with only system places in their precondition. Otherwise, they lose the game as no accepting state is reached infinitely often. It is losing if system players with positive and negative NES-status synchronize, if players with positive NES-status deadlock, if one such player is not moved and the marking from the start of the NES-case is repeated, if

all such players vanish, or if another marking is repeated. Notice that at most one NES-case is necessary per branch in the strategy tree of the Büchi game. For a safety winning condition, possible NES-cases after the first successful one can simply terminate.

▶ **Example 5.** A Petri game with necessary NES-case in the encoding Büchi game is shown in Fig. 3b. After Player 0 allows transition t_2 and Player 1 fires it, a state is reached where the decision tuples for s_4 and s_5 can be set to positive NES-status by Player 0. After transitions t_5, t_6, and t_7 fire, the marking $\{s_4, s_5\}$ is repeated and the NES-case is successful, proving that t_5, t_6, and t_7 can fire infinitely often.

5.6 Decidability Result

We analyze the properties of the constructed Büchi game. Detailed proofs are in [12].

▶ **Lemma 6** (From Büchi game to Petri game strategies). *If Player 0 has a winning strategy in the Büchi game, then there is a winning strategy for the system players in the Petri game.*

Proof Sketch. From the tree T_f representing the winning strategy f for Player 0 in the Büchi game, we inductively build a winning strategy σ for the system players in the Petri game. Each cut in σ is associated with a node in T_f, transitions are added following the edges in T_f, and the associated cut is updated if needed. This strategy σ for the system players in the Petri game is winning as it visits equivalent cuts to the reachable states in f. ◀

▶ **Lemma 7** (From Petri game to Büchi game strategies). *If the system players have a winning strategy in the Petri game, then there is a winning strategy for Player 0 in the Büchi game.*

Proof Sketch. We skip unnecessary NES-cases and useless repetitions in the winning strategy σ for the system players in the Petri game. We replace $\top$ based on the post-condition of system places, disclose necessary NES-cases, fire enabled transitions with only system places in their precondition in an arbitrary but fixed order between states after an mcut and the next mcut, and add all options at mcuts. This strategy for Player 0 in the Büchi game is winning as it visits equivalent states to the reachable cuts in σ. ◀

▶ **Theorem 8** (Game solving). *For Petri games with a bounded number of system players, one environment player, and bad markings, the question of whether the system players have a winning strategy is decidable in* 2-EXPTIME. *If a winning strategy for the system players exists, it can be constructed in exponential time.*

Proof Sketch. The complexity is based on the double exponential number of states in the Büchi game and polynomial solving of Büchi games. There are exponentially many states in the size of the Petri game to represent decision tuples and each of these states has to store sequences of backward moves of at most exponential length in the size of the Petri game. This transfers to the size of the winning strategy because it can be represented finitely. ◀

▶ Remark 9. In the presented construction, Player 0 in the Büchi game decides both the decisions of the system players in the Petri game and the order in which concurrent transitions with only system places in their precondition are fired. The question might arise whether it is possible that Player 1 representing the environment determines the order in which concurrent transitions with only system places in their precondition are fired. This is not possible because the system players can make different decisions depending on the order of transitions decided by the environment player. We present a detailed counterexample where this change would result in a different winner of a Petri game in the full version [12].

6 Undecidability in Petri Games with Good Markings

We prove that it is undecidable if a winning strategy exists for the system players in Petri games with at least two system and one environment player and good and bad markings by enforcing an undecidable synchronous setting in Petri games. For this winning condition, no bad marking should be reached *until* a good marking is reached, which can be expressed in LTL. Notice also that, after a good marking has been reached, it is allowed to reach a bad marking. Afterward, we prove that it is undecidable if a winning strategy exists for the system players in Petri games with only good markings and at least three players, out of which one is an environment player and each of the other two changes between system and environment player. Bad markings from the previous result are encoded by system players repeatedly changing to environment players and back. More details can be found in [12]. The underlying main idea of the first construction is also used in other settings [26, 34, 36].

6.1 Petri Game for the Post Correspondence Problem

We recall that a strategy σ is winning for good and bad markings $\mathcal{W} = (\mathcal{M}_G, \mathcal{M}_B)$ if, for all complete firing sequences $t_0 t_1 t_2 \dots$ of all maximal plays π of σ with $M_0 = In^\pi \wedge M_0[t_0\rangle M_1[t_1\rangle M_2[t_2\rangle \dots$, there exists $i \geq 0$ with $\lambda^\pi[M_i] \in \mathcal{M}_G \wedge \forall 0 \leq j < i : \lambda^\pi[M_j] \notin \mathcal{M}_B$. The undecidability proof uses the Post correspondence problem [35]. The *Post correspondence problem* (PCP) is to determine, for a finite alphabet Σ and two finite lists $r_0, r_1, \dots, r_n$ and $v_0, v_1, \dots, v_n$ of non-empty words over Σ, if there exists a non-empty sequence $i_1, i_2, \dots, i_l \in \{0, 1, \dots, n\}$ such that $r_{i_1} r_{i_2} \dots r_{i_l} = v_{i_1} v_{i_2} \dots v_{i_l}$. This problem is undecidable.

To simulate the PCP in a Petri game, we use one environment and two system players. The three players are *independent* as they cannot communicate with each other. Each system player outputs a solution to the PCP. By firing a transition, a player *outputs the label* of the transition. The output of the first system player is $i_1 r_{i_1} \tau i_2 r_{i_2} \tau \dots i_l r_{i_l} \tau \#_1$ and the output of the second one is $j_1 v_{j_1} \tau j_2 v_{j_2} \tau \dots j_m v_{j_m} \tau \#_2$ for $i_1, \dots, i_l, j_1, \dots, j_m \in \{0, 1, \dots n\}$. Both system players output *indices* followed by the word from the index position of the respective list and τ, and end symbol $\#_1$ or $\#_2$ at the end of the sequence. Words r_i for $i \in \{i_1, \dots, i_l\}$ and v_j for $j \in \{j_1, \dots, j_m\}$ are output letter-by-letter. A correct solution to the PCP fulfills $l > 0$, $m > 0$, $l = m$, $i_1 = j_1$, $i_2 = j_2$, $\dots$, $i_l = j_m$, and $r_{i_1} r_{i_2} \dots r_{i_l} = v_{j_1} v_{j_2} \dots v_{j_m}$.

We ensure that strategies for the two system players can only win by outputting the same sequence of indices at both players. This permits to decide for these strategies if a good marking is reached where both system players have output a correct solution. Using the independence of the three players and depending on a choice by the environment player, we either check the equality of the output sequences of indices or of the letter-by-letter output sequences of words. Therefore, the strategy for the system players has to behave as if both is tested. With good markings, we restrict the asynchronous setting of Petri games to turn-taking firing sequences on the output indices or letters. Thus, we consider equivalent firing sequences to the synchronous setting and can check the conditions for a correct solution to the PCP after both system players have output the end symbol. With bad markings, we identify when output indices or output letters do not match. System players can only output the end symbol after outputting at least one index and word to ensure non-empty solutions.

6.2 Linear Firing Sequences via Good Markings

We use MOD-3 counters to restrict the asynchronous setting of Petri games to firing sequences equivalent to the synchronous setting of Pnueli and Rosner [34, 38]. The main idea is that

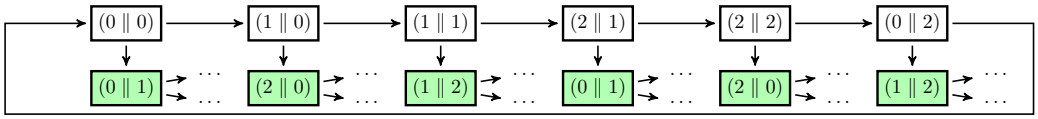


Figure 4 The reachability graph for the two system players is depicted when only considering either the values of their MOD-3 index counters or of their MOD-3 letter counters and differentiating markings depending on if a good marking is reached before. Good markings are colored green. All behavior after a good marking (including reaching a bad marking) is winning by definition. To compare output indices or letters, only the specific firing sequence in white has to be considered.

we are just interested in runs where the first system player is only zero or one step ahead of the second system player. When the second system player is ahead of the first one or the first system player is two or more steps ahead of the second one, then a good marking is reached and the possible reaching of bad markings afterwards does not matter. Hence, bad markings are only checked for runs where the first system player is zero or one step ahead of the second system player until they reach a good marking for giving an answer to the PCP.

Formally, for each system player, we introduce two *MOD-3 counters* to count the number of output indices and of output letters modulo three. When a player outputs an index, the respective index counter is increased by one, and accordingly for output letters and the letter counter. If a counter would reach value three, it is reset to zero. We define good markings based on the two MOD-3 index counters and the two MOD-3 letter counters. In a *linear firing sequence for indices (letters)*, the two system players output the indices (letters) alternately with the first system player preceding the second one at each turn. We ensure that the environment player first decides that either the output indices or letters are checked for equality. Afterward, a good marking is reached when a firing sequence is not a linear firing sequence for indices or letters, depending on the decision by the environment player.

In Fig. 4, we visualize the reachability graph for the two system players when only considering either the values of their MOD-3 counters for indices or letters. Markings are differentiated in the reachability graph depending on if a good marking is reached before, e.g., position $(0 \parallel 1)$ does not lead to position $(1 \parallel 1)$ as the path to $(1 \parallel 1)$ does not include a good marking. With linear firing sequences, we only consider firing sequences where the first system player outputs the first index or letter before the second system player as the opposite cases are good markings. For firing sequences not reaching a good marking, equality of output indices or letters is checked at positions $(0 \parallel 0)$, $(1 \parallel 1)$, and $(2 \parallel 2)$. Thereby, equality of output indices or letters at the same position can be checked without storing all outputs and it is ensured that solutions have the same length.

Notice that linear firing sequences for indices do not restrict the order in which the two system players output letters between two indices, and vice versa. Also, we at least need a MOD-3 counter. For a MOD-2 counter, the good marking $(2 \parallel 0)$ is replaced by $(0 \parallel 0)$, implying that all firing sequences contain a good marking. A MOD-3 counter prevents that one player overtakes the other. Thus, indices or letters at different positions are not compared, i.e., output indices or letters at position $(0 \parallel 3)$ (not modulo three) can be different.

6.3 Preventing Untruthful Termination

The good markings to only consider linear firing sequences introduce new possibilities for the system players to be winning. These possibilities arise when the system players can enforce all firing sequences to reach a good marking. They occur when a system player terminates without the end symbol ($\#_1$ or $\#_2$) and are called *untruthful termination*. Untruthful

termination is prevented by letting the environment player decide which system player it believes to not terminate with the end symbol or that everything is okay. This decision happens together with the initial choice of the environment player between checking equality of indices or letters. Due to the independence of the players, each system player has to behave as if the environment player is anticipating it to untruthfully terminate and has to output the end symbol to avoid this. Therefore, no untruthful termination can occur.

6.4 Undecidability Results

A winning strategy exists in the Petri game iff there exists a solution to the instance of the PCP. The only strategy with a chance to be winning for the two system players is to output the same solution to the PCP and we can translate solutions between both cases. We obtain:

▶ **Theorem 10.** *For Petri games with good and bad markings and at least two system and one environment player, the question if the system players have a winning strategy is undecidable.*

Bad markings can be encoded by system players repeatedly changing to environment players and back. Players commit to transitions and then system players become environment players. Environment players either follow the committed transition and fire a transition returning to the respective system and environment players or fire a transition with all other environment players after which no good markings are reachable to encode a bad marking. We obtain:

▶ **Theorem 11.** *For Petri games with good markings and at least three players, out of which one is an environment player and each of the other two changes between system and environment player, the question if the system players have a winning strategy is undecidable.*

7 Conclusion

We have investigated global winning conditions for the synthesis of asynchronous distributed reactive systems with causal memory. The general decidability or undecidability of the synthesis problem for these systems is a long-standing open question [30, 15]. We encode the synthesis problem for these systems by Petri games. For global winning conditions, we achieve a clear picture regarding decidability and undecidability.

From our decidability result and previous work [14], we obtain for bad markings as global winning condition that the question of whether the system players have a winning strategy is decidable for Petri games where the number of system players or the number of environment players is at most one and the number of players of the converse type can be bounded by some arbitrary number. For bad markings as global winning condition, this leaves the case of Petri games with two or more system players *and* two or more environment players open.

From our undecidability results, we obtain for good markings as global winning condition that the question of whether the system players have a winning strategy is undecidable for Petri games with two or more system players and three or more environment players. For good markings as global winning condition, this only leaves the corner case of Petri games with at most one system player and at most two environment players open.

Thus, for the synthesis of asynchronous distributed reactive systems with causal memory, global safety winning conditions are decidable for a large class of such systems, whereas global liveness winning conditions are undecidable for almost all classes of such systems. In the future, we plan to combine the decidability results for bad markings as global safety winning condition with local liveness specifications per player as in Flow-LTL [10, 11].

References

1 Raven Beutner, Bernd Finkbeiner, and Jesko Hecking-Harbusch. Translating asynchronous games for distributed synthesis. In *30th International Conference on Concurrency Theory, CONCUR*, volume 140 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. doi:10.4230/LIPIcs.CONCUR.2019.26.

2 Aaron Bohy, Véronique Bruyère, Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. Acacia+, a tool for LTL synthesis. In *Computer Aided Verification - 24th International Conference, CAV*, volume 7358 of *Lecture Notes in Computer Science*. Springer, 2012. doi:10.1007/978-3-642-31424-7_45.

3 Krishnendu Chatterjee and Monika Henzinger. An $O(n^2)$ time algorithm for alternating Büchi games. In *Twenty-Third Annual ACM-SIAM Symposium on Discrete Algorithms, SODA*. SIAM, 2012. doi:10.1137/1.9781611973099.109.

4 Rüdiger Ehlers. Unbeast: Symbolic bounded synthesis. In *Tools and Algorithms for the Construction and Analysis of Systems - 17th International Conference, TACAS*, volume 6605 of *Lecture Notes in Computer Science*. Springer, 2011. doi:10.1007/978-3-642-19835-9_25.

5 Joost Engelfriet. Branching processes of Petri nets. *Acta Informatica*, 28(6), 1991. doi:10.1007/BF01463946.

6 Javier Esparza and Keijo Heljanko. *Unfoldings - A Partial-Order Approach to Model Checking*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2008. doi:10.1007/978-3-540-77426-6.

7 Peter Faymonville, Bernd Finkbeiner, Markus N. Rabe, and Leander Tentrup. Encodings of bounded synthesis. In *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS*, volume 10205 of *Lecture Notes in Computer Science*, 2017. doi:10.1007/978-3-662-54577-5_20.

8 Bernd Finkbeiner. Bounded synthesis for Petri games. In *Correct System Design - Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday*, volume 9360 of *Lecture Notes in Computer Science*. 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 *Sixth Workshop on Synthesis, SYNT@CAV*, volume 260 of *EPTCS*, 2017. doi:10.4204/EPTCS.260.5.

10 Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. Model checking data flows in concurrent network updates. In *Automated Technology for Verification and Analysis - 17th International Symposium, ATVA*, volume 11781 of *Lecture Notes in Computer Science*. Springer, 2019. doi:10.1007/978-3-030-31784-3_30.

11 Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. AdamMC: A model checker for Petri nets with transits against Flow-LTL. In *Computer Aided Verification - 32nd International Conference, CAV*, volume 12225 of *Lecture Notes in Computer Science*. Springer, 2020. doi:10.1007/978-3-030-53291-8_5.

12 Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. Global winning conditions in synthesis of distributed systems with causal memory (Full Version). *CoRR*, abs/2107.09280, 2021. arXiv:2107.09280.

13 Bernd Finkbeiner, Manuel Gieseking, and Ernst-Rüdiger Olderog. Adam: Causality-based synthesis of distributed systems. In *Computer Aided Verification - 27th International Conference, CAV*, volume 9206 of *Lecture Notes in Computer Science*. Springer, 2015. doi:10.1007/978-3-319-21690-4_25.

14 Bernd Finkbeiner and Paul Gölz. Synthesis in distributed environments. In *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS*, volume 93 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.FSTTCS.2017.28.

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

16 Paul Gastin, Benjamin Lerman, and Marc Zeitoun. Distributed games with causal memory are decidable for series-parallel systems. In *FSTTCS: Foundations of Software Technology and Theoretical Computer Science*, volume 3328 of *Lecture Notes in Computer Science*. Springer, 2004. doi:10.1007/978-3-540-30538-5_23.

17 Blaise Genest, Hugo Gimbert, Anca Muscholl, and Igor Walukiewicz. Asynchronous games over tree architectures. In *Automata, Languages, and Programming - 40th International Colloquium, ICALP*, volume 7966 of *Lecture Notes in Computer Science*. Springer, 2013. doi:10.1007/978-3-642-39212-2_26.

18 Manuel Gieseking, Jesko Hecking-Harbusch, and Ann Yanich. A web interface for Petri nets with transits and Petri games. In *Tools and Algorithms for the Construction and Analysis of Systems - 27th International Conference, TACAS*, volume 12652 of *Lecture Notes in Computer Science*. Springer, 2021. doi:10.1007/978-3-030-72013-1_22.

19 Hugo Gimbert. On the control of asynchronous automata. In *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS*, volume 93 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017. doi:10.4230/LIPIcs.FSTTCS.2017.30.

20 Ursula Goltz and Wolfgang Reisig. The non-sequential behavior of Petri nets. *Inf. Control.*, 57(2/3), 1983. doi:10.1016/S0019-9958(83)80040-0.

21 Jesko Hecking-Harbusch and Niklas O. Metzger. Efficient trace encodings of bounded synthesis for asynchronous distributed systems. In *Automated Technology for Verification and Analysis - 17th International Symposium, ATVA*, volume 11781 of *Lecture Notes in Computer Science*. Springer, 2019. doi:10.1007/978-3-030-31784-3_22.

22 Jesko Hecking-Harbusch and Leander Tentrup. Solving QBF by abstraction. In *Ninth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF*, volume 277 of *EPTCS*, 2018. doi:10.4204/EPTCS.277.7.

23 Swen Jacobs, Roderick Bloem, Maximilien Colange, Peter Faymonville, Bernd Finkbeiner, Ayrat Khalimov, Felix Klein, Michael Luttenberger, Philipp J. Meyer, Thibaud Michaud, Mouhammad Sakr, Salomon Sickert, Leander Tentrup, and Adam Walker. The 5th reactive synthesis competition (SYNTCOMP 2018): Benchmarks, participants & results. *CoRR*, abs/1904.07736, 2019. arXiv:1904.07736.

24 Barbara Jobstmann, Stefan J. Galler, Martin Weiglhofer, and Roderick Bloem. Anzu: A tool for property synthesis. In *Computer Aided Verification, 19th International Conference, CAV*, volume 4590 of *Lecture Notes in Computer Science*. Springer, 2007. doi:10.1007/978-3-540-73368-3_29.

25 Michael Luttenberger, Philipp J. Meyer, and Salomon Sickert. Practical synthesis of reactive systems from LTL specifications via parity games. *Acta Informatica*, 57(1-2), 2020. doi:10.1007/s00236-019-00349-3.

26 P. Madhusudan and P. S. Thiagarajan. A decidable class of asynchronous distributed controllers. In *CONCUR - Concurrency Theory*, volume 2421 of *Lecture Notes in Computer Science*. Springer, 2002. doi:10.1007/3-540-45694-5_11.

27 P. Madhusudan, P. S. Thiagarajan, and Shaofa Yang. The MSO theory of connectedly communicating processes. In *FSTTCS: Foundations of Software Technology and Theoretical Computer Science*, volume 3821 of *Lecture Notes in Computer Science*. Springer, 2005. doi:10.1007/11590156_16.

28 José Meseguer, Ugo Montanari, and Vladimiro Sassone. Process versus unfolding semantics for place/transition Petri nets. *Theor. Comput. Sci.*, 153(1&2), 1996. doi:10.1016/0304-3975(95)00121-2.

29 Thibaud Michaud and Maximilien Colange. Reactive synthesis from LTL specification with Spot. In *7th Workshop on Synthesis, SYNT@CAV*, 2018.

30 Anca Muscholl. Automated synthesis of distributed controllers. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP*, volume 9135 of *Lecture Notes in Computer Science*. Springer, 2015. doi:10.1007/978-3-662-47666-6_2.

31 Anca Muscholl and Igor Walukiewicz. Distributed synthesis for acyclic architectures. In *34th International Conference on Foundation of Software Technology and Theoretical Computer Science, FSTTCS*, volume 29 of *LIPIcs*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014. doi:10.4230/LIPIcs.FSTTCS.2014.639.

32 Mogens Nielsen, Gordon D. Plotkin, and Glynn Winskel. Petri nets, event structures and domains, Part I. *Theor. Comput. Sci.*, 13, 1981. doi:10.1016/0304-3975(81)90112-2.

33 Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science FOCS*. IEEE Computer Society, 1977. doi:10.1109/SFCS.1977.32.

34 Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *31st Annual Symposium on Foundations of Computer Science FOCS*. IEEE Computer Society, 1990. doi:10.1109/FSCS.1990.89597.

35 Emil L. Post. A variant of a recursively unsolvable problem. *Bull. Am. Math. Soc.*, 52(4), 1946.

36 John H. Reif. The complexity of two-player games of incomplete information. *J. Comput. Syst. Sci.*, 29(2), 1984. doi:10.1016/0022-0000(84)90034-5.

37 Wolfgang Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1985. doi:10.1007/978-3-642-69968-9.

38 Sven Schewe. Distributed synthesis is simply undecidable. *Inf. Process. Lett.*, 114(4), 2014. doi:10.1016/j.ipl.2013.11.012.

39 Wieslaw Zielonka. Notes on finite asynchronous automata. *ITA*, 21(2), 1987. doi:10.1051/ita/1987210200991.