The Synthesis Problem for Repeatedly Communicating Petri Games

Paul Hannibal(✉) and Ernst-Rüdiger Olderog

Carl von Ossietzky University Oldenburg, 26129 Oldenburg, Germany
paul.jonathan.hannibal1@uni-oldenburg.de,
ernst.ruediger.olderog@informatik.uni-oldenburg.de

Abstract. Petri games are a multi-player game model for the automatic synthesis of distributed systems, where the players are represented as tokens on a Petri net and grouped into environment players and system players. As long as the players move in independent parts of the net, they do not know of each other; when they synchronize at a joint transition, each player gets informed of the entire causal history of the other players.

We present a subclass of Petri games, for which the synthesis problem is decidable, with finitely many sources of nondeterminism, which are caused by the finitely many environment players, and with finitely many system players. All players satisfy a synchronisation condition guaranteeing that they know within a bounded number of own moves what each other player's next (non)deterministic move has been. This differs from existing approaches that limit the number of the system players or environment players. We show that for Petri games in this subclass deciding the existence of a winning strategy for the system players with a global safety condition is in EXPTIME.

Keywords: Synthesis · Distributed systems · Concurrent systems · Petri nets · Petri games · Unfolding

1 Introduction

A game can be interpreted as a formal specification of a reactive system. If the system is distributed over several processes, a multi-player game is appropriate for its specification. In a multi-player game one distinguishes between environment and system players. A system player can control or choose which move it takes next. An environment player is uncontrollable for the system players; they have to react to all options of the environment players. A strategy for the system players decides all choices that they have to make during a play, which is a possibly infinite sequences of moves. A strategy is *winning* if it fulfills a given winning condition against all behaviors of the environment. Thus, an implementation of a winning strategy can be seen as a correct implementation of a reactive system. The *synthesis problem* asks whether there exists a winning strategy for the system players and calls for the automatic generation of such a strategy if it

L. Bernardinello and L. Petrucci (Eds.): PETRI NETS 2022, LNCS 13288, pp. 236–257, 2022.
https://doi.org/10.1007/978-3-031-06653-5_13

exists. Such an automatic generation is useful for implementation tasks, which are prone to errors.

In this paper, we consider Petri games as a formalisation of multi-player games. A Petri game extends a Petri net by dividing its places into system and environment places. A token on a system place represents a system player and a token on an environment place represents an environment player. Specific to Petri games is the notion of informedness of the players. As long as the players move in independent parts of the net, they do not know of each other; when they synchronize at a joint transition, each player gets informed of the entire causal history of the other players. Petri games are equipped with a global safety condition, which is formalized as a set of 'bad' markings, i.e., sets of places which must never be reached simultaneously by the players.

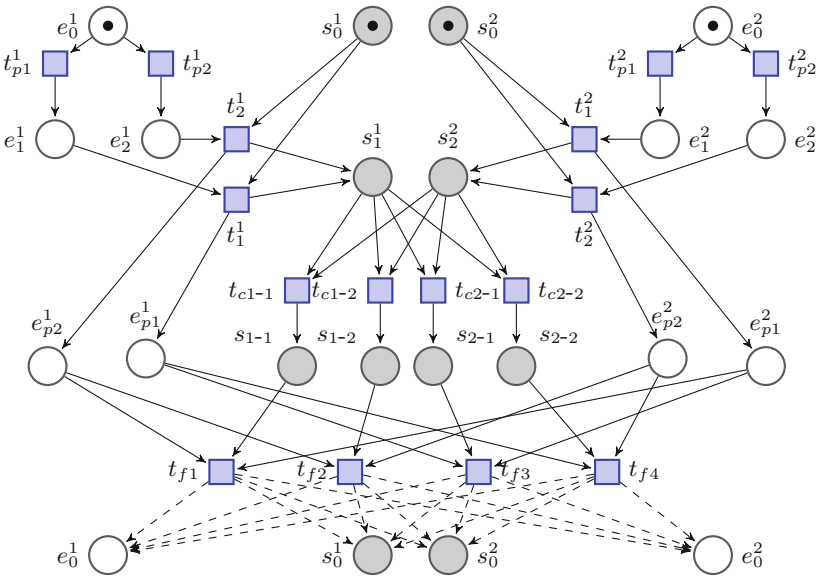

Fig. 1. A Petri game: the grey places belong to the system players and the white places to the environment players. The two environment players have two ports each to choose via t_{p1}^i and t_{p2}^i, $i = 1, 2$. The goal for the system players is to connect the chosen ports such that the environment players can communicate; every marking containing a place $s_{i\text{-}j}$ and a place $e_{p_k}^1$ with $k \neq i$ or a place $e_{p_l}^2$ with $l \neq j$ is a bad marking. The outgoing edges of the transitions t_{fi}, $i = 1, \ldots, 4$, putting the tokens back on the initial places are shown with dotted lines to keep the Petri net overseeable; those are the same places.

Petri games have been introduced in [13], where it has been shown that, limiting the environment to one player, the synthesis problem is EXPTIME-complete [13]. The dual case, limiting the system to one player, is also EXPTIME-complete [12]. For Petri games with unboundedly many players the synthesis problem is undecidable in general [13]. An approach of limiting the size of the winning strategies to search for is presented as *bounded synthesis* in [9].

In this paper, we introduce a subclass of Petri games with finitely many system players and finitely many environment players that satisfy a synchronisation condition which ensures that all players hear from each other directly or indirectly within a bounded number of own transitions, or that the Petri net is acyclic. Games with an arbitrary number of system and environment players are key to specifying many realistic distributed reactive systems. We show that for Petri games in this subclass the synthesis problem is decidable in EXPTIME.

Figure 1 shows a Petri game in an abstract communication setting, where two environment players have two ports each to connect to via transitions. The system players have to link the correct ports after knowing the choice of each environment player. If done so, the environment players communicate over the connected channel and every token is put back to the initial places. The Petri game in Fig. 1 satisfies the synchronisation condition as all tokens hear from each other in the joint transition t_{fi}, $i = 1, \ldots, 4$.

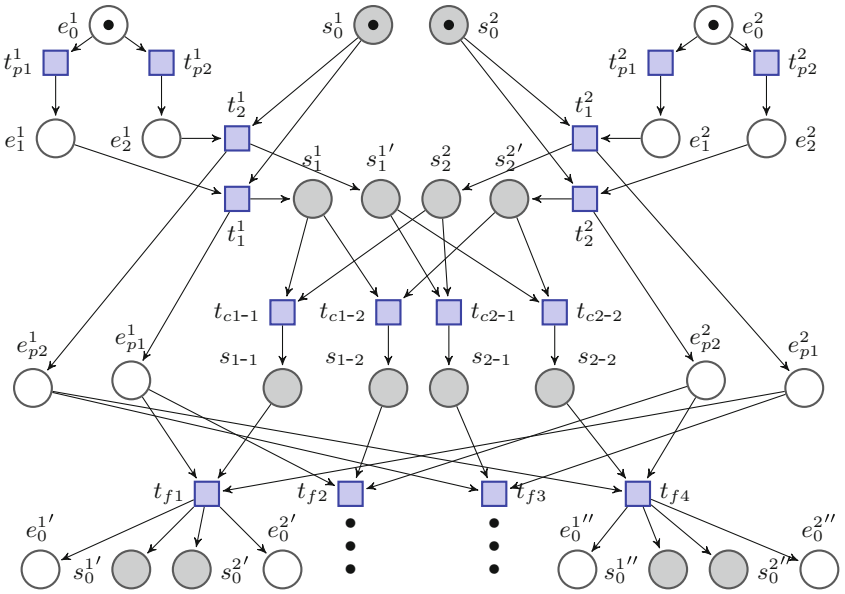

Fig. 2. An initial part of a winning strategy of the Petri game in Fig. 1. This is a part of the unfolding of the Petri net. The places $e_0^{1\prime}$, $s_0^{1\prime}$, $s_0^{1\prime}$, $s_0^{1\prime}$ are new instances of the initial places.

Figure 2 shows an initial part of the described winning strategy. The winning strategy is a part of the unfolding of the Petri net, which is itself a Petri net, where every flow of tokens through the net is represented by distinct new places and transitions: the token initially on place s_0^1 can distinguish its causality on the places s_1^1 and $s_1^{1\prime}$: on place s_1^1 the system players knows that the environment player is connected to port one and in $s_1^{1\prime}$ to port two. It does not know the port

the other environment player connected to. From there on, the system players can choose from four transitions to connect the ports. In the winning strategy, only the correct option is chosen based on the causality.

The presented solution of the synthesis problem of the described subclass of Petri games is a reduction to a two-player graph game with a safety condition. The idea for the system player is to plan so far ahead that no information, that is not yet available in the Petri net unfolding, is used for the decisions made in the winning strategy.

The remainder of this paper is organized as follows. In Sect. 2 we introduce the notions of Petri games and graph games. In Sect. 3 we define the reduction to graph games and show its correctness. Related work and conclusions are presented in Sect. 4 and Sect. 5.

2 Foundations

In this section, we define branching processes and unfoldings as they are defined in [7]. Also, we define Petri games and their winning strategies.

The *power set* for a set A is denoted as $2^A = \{B \mid B \subseteq A\}$. The *set of nonempty finite subsets* for a set A is denoted as $2^A_{nf} = \{B \mid B \subseteq A \wedge B \text{ is nonempty and finite}\}$ and the *set of finite subsets* as 2^A_f. A *Petri net* is a 5-tuple $(\mathcal{P}, \mathcal{T}, pre, post, In)$, where $\mathcal{P}$ is the set of *places*, $\mathcal{T}$ is the set of *transitions*, *pre* and *post* are flow mappings, *In* is the *initial marking* and the following properties hold: $\mathcal{P} \cap \mathcal{T} = \emptyset$, $pre :\mapsto 2^{\mathcal{P}}_{nf}$, $post : \mathcal{T} \mapsto 2^{\mathcal{P}}_f$, and $In \subseteq \mathcal{P}$ is the initial marking. A Petri net is called finite if $\mathcal{P} \cup \mathcal{T}$ is finite. The flow mappings *pre* and *post* are extended for places as usual: $\forall p \in \mathcal{P} : pre(p) = \{t \in \mathcal{T} \mid p \in post(t)\}$ and $\forall p \in \mathcal{P} : post(p) = \{t \in \mathcal{T} \mid p \in pre(t)\}$. The *flow relation* $\mathcal{F}$ is defined as $\mathcal{F} = \{(p,t) \in \mathcal{P} \times \mathcal{T} \mid p \in pre(t)\} \cup \{(t,p) \in \mathcal{T} \times \mathcal{P} \mid p \in post(t)\}$. A *marking* M of a Petri net $\mathcal{N}$ is a multi-set over $\mathcal{P}$. In particular, *In* is a marking. A Petri net N is called *safe*, if for all reachable markings $M(p) \leq 1$ for all $p \in \mathcal{P}$ holds. Then, M is a subset of $\mathcal{P}$.

A transition $t \in \mathcal{T}$ is *enabled* in marking M, if $pre(t) \subseteq M$. If t is enabled, the transition t can be *fired*, such that the new marking is $M' = M \setminus pre(t) \cup post(t)$. This is denoted as $M|t\rangle M'$. The marking M' is also denoted by $M|t\rangle$. This notation is extended to sequences of enabled transitions $M|t_1 \dots t_n\rangle M'$ and $M|t_1 \dots t_n\rangle$, respectively A marking M is *reachable*, if there exists a sequence of enabled transitions $(t_k)_{k=\{1,\dots,n\}}$ and $In|t_1 \dots t_n\rangle M$. This sequence can be empty. The set of all reachable markings is denoted as $\mathcal{R}(N)$, where N is a Petri net. We call a *node* x a place or a transition $x \in \mathcal{P} \cup \mathcal{T}$.

The following definitions are essential for understanding this paper and are also taken from [7]. A node $x \in \mathcal{P} \cup \mathcal{T}$ is a *causal predecessor* of y, denoted as $x \leq y$, if there is a sequence $(x_i, y_i)_{i=1,\dots,n}$, $(x_i, y_i) \in \mathcal{F}$, where $x_1 = x$, $y_n = y$ and $y_i = x_{i+1}$ forall $i = 1, \dots, n-1$. Furthermore, $x \leq x$ holds for all $x \in \mathcal{P} \cup \mathcal{T}$. Two nodes $x, y \in \mathcal{P} \cup \mathcal{T}$ are *causally related*, if and only if $x \leq y$ or $y \leq x$ holds. We say x is a *causal successor* of y, if and only if $y \leq x$ holds. We define

the *past* of a node in an occurrence net as the set of all causal predecessors $Past(x) = \{y \in \mathcal{P} \cup \mathcal{T} \mid y \leq x\}$ as this is used in later definitions.

Two nodes $x_1, x_2 \in \mathcal{P} \cup \mathcal{T}$ are *in conflict*, denoted $x_1 \# x_2$, if there exist two transitions $t_1, t_2 \in \mathcal{T}$, $t_1 \neq t_2$ with $pre(t_1) \cap pre(t_2) \neq \emptyset$ and $t_i \leq x_i$, $i = 1, 2$. A node $x \in \mathcal{P} \cup \mathcal{T}$ is in self-conflict, if $x \# x$. Informally speaking, two nodes are in conflict if two transitions exist that share some place in their presets and each node is a causal successor of one of those transitions. Two nodes $x, y \in \mathcal{P} \cup \mathcal{T}$ are *concurrent*, denoted $x || y$, if they are neither causally related nor in conflict.

A Petri net N is *finitely preceded*, if for every node $x \in \mathcal{P} \cup \mathcal{T}$ the set $Past(x)$ is finite. A Petri net N is *acyclic*, if the directed graph $(\mathcal{P} \cup \mathcal{T}, \mathcal{F})$ is acyclic. The two following definitions lead to the definition of a branching process, which represents several runs of the underlying Petri net taken together. A run is represented by a (possibly infinite) firing sequence of transitions. An *occurrence net* is a Petri net $N = (\mathcal{P}, \mathcal{T}, pre, post, In)$ with the following properties: N is acyclic, finitely preceded, $\forall p \in \mathcal{P} : |pre(p)| \leq 1$, no transition $t \in \mathcal{T}$ is in self-conflict, and $In = \{p \in \mathcal{P} | \, pre(p) = \emptyset\}$.

A homomorphism from one Petri net to another maps each node to a node such that the preset and postset relations are preserved including the initial marking. Let $N_1 = (\mathcal{P}_1, \mathcal{T}_1, pre_1, post_1, In_1)$ and $N_2 = (\mathcal{P}_2, \mathcal{T}_2, pre_2, post_2, In_2)$ be two Petri nets. A *homomorphism* from N_1 to N_2 is a mapping $h : \mathcal{P}_1 \cup \mathcal{T}_1 \to \mathcal{P}_2 \cup \mathcal{T}_2$ with following properties: $h(\mathcal{P}_1) \subseteq \mathcal{P}_2$ and $h(\mathcal{T}_1) \subseteq \mathcal{T}_2$, for all transitions $t \in \mathcal{T}_1$, h restricted to $pre_1(t)$ is a bijection between $pre_1(t)$ and $pre_2(h(t))$, for all transitions $t \in \mathcal{T}_1$, h restricted to $post_1(t)$ is a bijection between $post_1(t)$ and $post_2(h(t))$, and the restriction of h to In_1 is a bijection between In_1 and In_2. An *isomorphism* is an bijective homomorphism.

The previous definitions now lead to the definition of a branching process of a Petri net resembling multiple runs of the Petri net.

Branching Process. Let $N_0 = (\mathcal{P}_0, \mathcal{T}_0, pre_0, post_0, In_0)$ be a Petri net. A *branching process* of N_0 is a pair $B = (N, \pi)$, where $N = (\mathcal{P}, \mathcal{T}, pre, post, In)$ is an occurrence net and π a homomorphism from N to N_0 such that:

(*) For all $t_1, t_2 \in \mathcal{T}$: if $pre(t_1) = pre(t_2)$ and $\pi(t_1) = \pi(t_2)$, then $t_1 = t_2$.

The notion of the set of all reachable markings of a branching process $B = (N, \pi)$ is extended to $\mathcal{R}(B) = \mathcal{R}(N)$.

The property $(*)$ of the definition of a branching process ensures that every run of the Petri net is represented at most once. Informally speaking, a run only consists of concurrent and causally related nodes and a node can be part of multiple runs. Nodes that are in conflict, cannot belong to the same run.

Homomorphism on Branching Processes. Given two branching processes $B_1 = (N_1, \pi_1)$ and $B_2 = (N_2, \pi_2)$ of a Petri net N_0. A *homomorphism* from B_1 to B_2 is a homomorphism h from N_1 to N_2 such that $\pi_2 \circ h = \pi_1$. It is called an *isomorphism* if h is an isomorphism. The branching processes B_1 and B_2 are *isomorphic* if there exists an isomorphism from B_1 to B_2.

As the names of the nodes of isomorphic branching processes may differ we define the set of canonical names such that every branching process is isomorph to a branching process with canonical names.

Canonical Names. Let $N = (\mathcal{P}, \mathcal{T}, pre, post, In)$ be a Petri net, the set of *canonical Names* CAN is the smallest set such that, if $x \in \mathcal{P} \cup \mathcal{T}$ and A is a finite subset of CAN, then $(x, A) \in CAN$.

A canonical name of a node in a branching process is composed of the label of the node and the set of the canonical names of all nodes in its preset. The label of a node x is the name of the node in the underlying Petri net, denoted as $\pi(x)$.

Canonical Coding. Let $B = (N, \pi)$ be a branching process of N_0 and $N = (\mathcal{P}, \mathcal{T}, pre, post, In)$. Then the *canonical coding* of B is a mapping $cod_B : \mathcal{P} \cup \mathcal{T} \to CAN$ with $cod_B(x) = (\pi(x), cod_B(pre(x)))$. Note that $pre(x)$ is a set and thus $cod_B(pre(x)) = \{cod_B(y) | \, y \in pre(x)\}$. Consequently, a branching process is called *canonical* if all nodes have canonical names.

Canonical Branching Process. A branching process $B = (N, \pi)$ is called *canonical*, if $\mathcal{P} \cup \mathcal{T} \subseteq CAN$ and $cod_B(x) = x$ for all $x \in \mathcal{P} \cup \mathcal{T}$.

Note that the definition of a canonical branching process requires the initial marking of the branching process to be a set rather than a multi set.

A natural partial order on branching processes is defined in the following.

Subprocess Relation of Branching Processes. Let B_1 and B_2 be two branching processes of a Petri net $N = (\mathcal{P}, \mathcal{T}, pre, post, In)$. Then B_1 approximates B_2, denoted $B_1 \leq B_2$, if there exists an injective homomorphism denoted $h_{\leq}$ from B_1 to B_2.

Note that this partial order is independent of a branching processes being canonical. Restricting this partial order to canonical branching processes results in a partial order, too. Now we define the maximal canonical branching process as the unfolding of a Petri net.

Unfolding. A branching process $B = (N, \pi)$ with $N = (\mathcal{P}, \mathcal{T}, pre, post, In)$ is isomorphic to the unfolding $unf(N_0)$ of the underlying Petri net $N_0 = (\mathcal{P}_0, \mathcal{T}_0, pre_0, post_0, In_0)$ if and only if the following holds: For all transitions $t_0 \in \mathcal{T}_0$ and all sets $C \subseteq \mathcal{P}$ of pairwise concurrent places exists $t \in \mathcal{T}$ with $pre(t) = C$ and $\pi(t) = t_0$, if the restriction of π to C is a bijection between C and $pre_0(t_0)$. The notation $unf(N_0)$ denotes the canonical unfolding. We refer to the components of the unfolding as $\mathcal{T}_{unf(N_0)}$, $\mathcal{P}_{unf(N_0)}$, $pre_{unf(N_0)}$, $post_{unf(N_0)}$, and $In_{unf(N_0)}$.

As we only consider Petri games of finite and safe Petri nets in this paper, these properties are part of the definition of a Petri game. Note that we allow tokens to be generated or deleted. Also, we allow tokens to transition from a system place to an environment place and vice versa.

Definition 1 (Petri game). *A* Petri-game *of an underlying finite and safe Petri net N is a tuple $G = (\mathcal{P}^S, \mathcal{P}^E, \mathcal{T}, pre, post, In, \mathcal{B})$, where the places are two disjoint sets $\mathcal{P}^S$, called the* system places *and $\mathcal{P}^E$, called the* environment

places *with* $\mathcal{P}^S \cup \mathcal{P}^E = P$. *The sets* $\mathcal{P}$ *and* $\mathcal{T}$ *are finite.* $\mathcal{B}$ *is the set of* bad markings.

We assume the underlying Petri net of a Petri game to be safe for formal simplicity of the canonical branching processes. Note that a finite and safe Petri net has a bounded number of places in every reachable marking.

Now we can define a winning strategy as a branching process of the underlying Petri net, that satisfies four properties.

Definition 2 (Winning strategy). *A* winning strategy *of a Petri-game* $G_P = (\mathcal{P}_0^S, \mathcal{P}_0^E, \mathcal{T}_0, pre_0, post_0, In_0, \mathcal{B})$ *with underlying Petri-net* $N_0 = (\mathcal{P}_0, \mathcal{T}_0, pre_0, post_0, In_0)$ *is a branching process* $B = (N, \pi)$ *of* N_0 *with* $N = (\mathcal{P}, \mathcal{T}, pre, post, In)$ *and the following properties.*

1. ***Justified refusal****: Let* $C \subseteq \mathcal{P}$ *be a set of pairwise concurrent places and* $t \in T_0$ *a transition with* $\pi(C) = pre_0(t)$. *If no* $t' \in \mathcal{T}$ *with* $\pi(t') = t$ *and* $pre(t') = C$ *exists, then there exists a place* $p \in C$ *with* $\pi(p) \in \mathcal{P}_0^S$, *such that* $t \notin \pi(post(p))$.
2. ***Safety****: For all reachable markings* M *in* N *holds* $\pi(M) \notin \mathcal{B}$.
3. ***Determinism****: For all* $p \in \mathcal{P}$ *with* $\pi(p) \in \mathcal{P}_0^S$ *and for all reachable markings* M *in* N *with* $p \in M$ *exists at most one transition* $t \in post(p)$, *which is enabled in* M.
4. ***Deadlock avoiding****: For all reachable markings* M *in* N *exists an enabled transition, if a transition is enabled in* $\pi(M)$ *in the underlying Petri-net* N_0.

We fix the notations $G_P = (\mathcal{P}_0^S, \mathcal{P}_0^E, \mathcal{T}_0, pre_0, post_0, In_0, \mathcal{B})$ *of the Petri game* G_P *and* $N_0 = (\mathcal{P}_0, \mathcal{T}_0, pre_0, post_0, In_0)$ *of the underlying Petri net* N_0.

The four properties of a winning strategy can be interpreted as follows: The *justified refusal* property forces the system player in each place to allow all instances of an outgoing transition t or no instance at all. This enables the representation of the decisions of the system player as commitment sets in its places: each transition is allowed for every possible instance or it is forbidden at all. The *safety* property ensures that no bad markings are reachable. The *determinism* property ensures that for each system place at most one transition is enabled in every reachable marking. The *deadlock avoiding* property ensures that the system allows at least one transition in every reachable marking if an enabled transition exists in that marking.

2.1 Graph Games

A graph game is a two player game with perfect information played on a directed graph, called an arena. The vertices in the arena divide into system vertices and environment vertices. The system player, who chooses the next move in system vertices, is referred to as player 0 and the environment player, who chooses the next move in environment vertices, is player 1.

An *arena* $A = (V, V_0, V_1, E)$ consists of a finite set V of vertices, disjoint subsets $V_0, V_1 \subseteq V$ with $V = V_0 \cup V_1$ denoting the vertices of player 0 and player

1, a set $E \subseteq V \times V$ of (directed) edges such that every vertex has at least one outgoing edge, i.e., $\{v_0 \mid (v, v_0) \in E\}$ is non-empty for every $v \in V$. The size of A, denoted by $|A|$, is defined to be $|V|$. A *play* in an arena $A = (V, V_0, V_1, E)$ is an infinite sequence of vertices $\mu = v_1 v_2 v_3 \ldots \in V^\omega$ such that $(v_n, v_{n+1}) \in E$ holds for every $n \in \mathbb{N}$. We say μ starts in the vertex v_1. The set of plays in A is denoted by $Plays(A)$, the set of all plays starting in v by $Plays(A, v)$, and we define $Plays(A, V') = \bigcup_{v \in V'} Plays(A, v)$ for every $V' \subseteq V$.

A strategy of a player determines the next vertex of a play, if the current vertex belongs to the player. A *strategy* for Player $i \in \{0, 1\}$ in an arena (V, V_0, V_1, E) is a function $\sigma : V^* V_i \mapsto V$ such that $\sigma(wv) = v'$ implies $(v, v') \in E$ for every $w \in V^*$ and every $v \in V_i$. The set of plays obtained by following a strategy is the set of consistent plays. A play $v_1 v_2 v_3 \ldots$ in an arena $A = (V, V_0, V_1, E)$ is consistent with a strategy σ for Player i in A if $v_{n+1} = \sigma(v_1 \ldots v_n)$ for every $n \in \mathbb{N}$ with $v_n \in V_i$. Given a vertex v, we denote the set of plays that are consistent with σ and start in v with $Plays(A, v, \sigma)$. Finally, we define $Plays(A, V', \sigma)$ for $V' \subseteq V$ by $Plays(A, V', \sigma) = \bigcup_{v \in V'} Plays(A, v, \sigma)$.

The winning condition in a safety graph game is to remain in safe vertices. The set of safe vertices is a subset of all vertices. A *safety graph game* $G = (A, S)$ consists of an arena A with vertex set V and a set of safe vertices $S \subseteq V$. We call a sequence μ winning for Player 0 if, and only if $Occ(\mu) \subseteq S$. $Occ(\mu)$ denotes all vertices occurring in μ: $Occ(\mu) := \{v \in V \mid \exists n \in \mathbb{N} : v_n = v\}$. A strategy is winning, if all consistent plays remain in safe vertices at all time: A strategy σ of a safety graph game $G = (A, S)$ is called winning in a vertex $v \in V$, if all consistent plays $\mu \in Plays(A, v, \sigma)$ are winning.

A strategy σ for Player i in an arena (V, V_0, V_1, E) is *positional* if $\sigma(wv) = \sigma(v)$ for all $w \in V^*$ and $v \in V_i$. Safety graph games are determined with positional winning strategies. These strategies are called positional because they do not need any memory of the vertices visited so far.

Safety graph games can be solved with the standard attractor construction in linear time in the number of edges of the underlying arena [2].

3 Reduction of Petri Games to Graph Games

In this section, we reduce Petri games, where the underlying Petri net satisfies a synchronisation condition, to safety graph games. We call the synchronisation condition non-simultaneous synchronisation condition as the players do not need to take one joint transition. They do need to be causally dependent from each other directly or indirectly after they take a bounded number of transitions. The only exception occurs, if there are only finite firing sequences of transitions in the Petri net meaning the Petri net is acyclic.

Informally speaking the non-simultaneous synchronisation condition defined in the following expresses that every token in the Petri net has to take a transition within firing at most n transitions or no transition is enabled anymore after $n - 1$ transitions. This ensures that every token hears from every other token directly or indirectly after firing finitely many transitions, if there are still enabled transitions in the Petri net.

Definition 3 (Non-simultaneous synchronisation condition). *A Petri net N satisfies the non-simultaneous synchronisation condition if and only if there exists a bound $n \in \mathbb{N}$ such that*

$$\forall M \in \mathcal{R}(N) : \forall s \in M : \neg \exists t_1 \dots t_n : M | t_1 \dots t_n \rangle M' \wedge \forall k = 1, \dots n : s \notin pre(t_k).$$

This bound is unrelated to the number of tokens in the Petri net. An equivalent characterisation of Petri nets that satisfy this synchronisation condition is to bound the number of concurrent transitions for every place in the unfolding of a Petri net. A Petri net satisfies the non-simultaneous synchronisation if and only if there exists a bound $m \in \mathbb{N}$ such that for all places $p \in unf(N)$ the set of concurrent transitions in the unfolding is bounded by m, i.e. $|\{t \in \mathcal{T}_{unf(N)} \mid p||t\}| \leq m$.

The remainder of this section is structured as follows: in Subsect. 3.1 we introduce further definitions for Petri games such that we can express a winning strategy in a Petri game to be planned part by part in a graph game. In Subsect. 3.2 we define the graph game to which a Petri game is reduced. In Subsect. 3.3 we show how to construct a winning Petri game strategy if the graph game has a winning strategy and vice versa.

3.1 Extended Petri Game Semantics

In the following, we introduce another synchronisation condition, the local synchronisation condition, that is defined for the nodes in a branching process as opposed to the non-simultaneous synchronisation condition that is defined for a Petri net.

The reduction from Petri games to graph games, where both players have perfect information, needs to ensure that the decisions of the system player in the graph game do not rely on information it would not have in the Petri game. Therefore, we introduce a local synchronisation condition that ensures that the system player has sufficient information in the graph game. The idea is that the system player plans so far ahead until it meets the local synchronisation condition for all tokens such that it cannot abuse information in the graph game it would not have in the Petri game. We define the set of enabled transitions in a marking of a Petri net and the local synchronisation condition dependent on this set as follows. The set of the enabled transitions in a marking M of a Petri net is defined as $T_M^{En} = \{t \in \mathcal{T} \mid t \text{ is enabled in } M\}$. The notation T_B^{En} is used for a branching process B. This is the set of enabled transitions in the initial marking of B.

Definition 4 (Local synchronisation condition (abbreviated LSC)). *For a branching process $B = (N, \pi)$ of a Petri net N_0 the* local synchronisation condition*, abbreviated LSC, is defined for a node $x \in \mathcal{P} \cup \mathcal{T}$ as follows: x satisfies the LSC if*

$$\forall t \in T_B^{En} : t \in Past(x) \vee x \# t.$$

The local synchronisation condition is called local because the causal past of a node determines if it is satisfied: a node satisfies the local synchronisation condition if it has information of all enabled transitions in the initial marking whether they have been fired or not. The information that one of those transitions was not fired is equal to having a node in conflict to that transition in the past of the node that satisfies the local synchronisation condition.

In the graph game, the system player and environment player take alternate turns. The system player plans a branching process, i.e. the system player determines its strategy part by part. Then, the environment player chooses a set of enabled and pairwise concurrent transitions that are fired. Afterwards, the system player has to plan further ahead, and so on. We define firing transitions in a branching process which results in a branching process containing all remaining nodes that were not in conflict to a fired transition or a causal predecessor of fired transition. As the initial marking of this new branching process changed and therefore it might not be a branching process by definition, we extend the set of branching process to those starting in an arbitrary reachable marking of the underlying Petri net.

Definition 5 (Extended set of branching processes of a Petri net). *An extended branching process $B = (N, \pi)$ of a safe Petri net $N_0 = (\mathcal{P}_0, \mathcal{T}_0, pre_0, post_0, In_0)$ is a branching process of a Petri net $N_0' = (\mathcal{P}_0, \mathcal{T}_0, pre_0, post_0, M)$, where $M \in \mathcal{R}(N_0)$. The set of all extended branching processes of a Petri net N_0 or a Petri Game G_P is denoted as $EB(N_0)$ and $EB(G_P)$, respectively. The notion of the unfolding of N_0' as an extended branching process is added as $unf_M(N_0)$.*

In the following, we define firing a set of pairwise concurrent transitions in an extended branching process.

Definition 6 (Firing transitions in extended branching processes). *Let $B = (N, \pi)$ be an extended branching process with $N = (\mathcal{P}, \mathcal{T}, pre, post, In)$ and $T_f \subseteq T_B^{En}$ a set of pairwise concurrent transitions. Then $B|T_f\rangle B'$ denotes the firing of all transitions in T_f in an arbitrary order, resulting in the extended branching process*

$$B' = ((\mathcal{P} \setminus \{s \in \mathcal{P} \mid s \in pre(T_f) \vee \exists t' \in T_f : t' \# s\},$$
$$\mathcal{T} \setminus \{t \in \mathcal{T} \mid t \in T_f \vee \exists t' \in T_f : t' \# t\}, pre\restriction_{\mathcal{T}'}, post \restriction_{\mathcal{T}'}, M|T_f\rangle), \pi \restriction_{\mathcal{T}' \cup \mathcal{P}'}),$$

where the components of B' are referred to as $\mathcal{P}'$, $\mathcal{T}'$, pre', $post'$, In' and π', and $pre\restriction_{\mathcal{T}'}$ and $post \restriction_{\mathcal{T}'}$ are the restrictions of pre and post to the transitions $\mathcal{T}'$ of B', and $\pi \restriction_{\mathcal{T}' \cup \mathcal{P}'}$ the restriction of π to $\mathcal{T}' \cup \mathcal{P}'$.

The branching process B' is also denoted as $B|T_f\rangle$. We extend these notations for sequences of sets of concurrent transitions as $B|T_{f_1} \dots T_{f_n}\rangle B'$ and $B|T_{f_1} \dots T_{f_n}\rangle$ respectively. Here, $M|T_f\rangle$ denotes the marking reached after firing all transitions in T_f in an arbitrary order. Let $\tilde{B}' \cong B'$ denote the canonical branching process, which is isomorph to B' via an isomorphism $\Phi : B' \mapsto \tilde{B}'$.

To ensure that the decisions of the system player which transitions are allowed to be fired in each place are final, such that the system does not change its decisions when it has more information, we define a commitment set mapping for an extended branching process, which states for every place the set of transitions which are allowed to be fired. This ensures the *justified refusal* property of a winning strategy in a Petri game. For environment places the allowed transitions are not restricted. In the graph game, the system player has to choose a commitment set for every system place, when it is added, and this commitment set is kept the same from there on.

Definition 7 (Commitment set mapping). *Given a Petri game G_P, a* commitment set mapping *CS of an extended branching process $B = (N, \pi) \in EB(G_P)$, where $N = (\mathcal{P}, \mathcal{T}, pre, post, M)$, is a mapping with the following properties:*

$$\begin{aligned} CS :& \mathcal{P} \to 2^{\mathcal{T}_0} \\ & \forall s \in \mathcal{P} : CS(s) \subseteq post_0(\pi(s)) \\ & \wedge (\pi(s) \in \mathcal{P}_0^E \Rightarrow CS(s) = post_0(\pi(s))) \end{aligned}$$

For example, the commitment set of the system place s_1^1 in Fig. 2 would be $\{t_{c1\text{-}1}, t_{c1\text{-}2}\}$ as it needs to connect port one of the first environment player to either port one or port two of the other environment player.

In the following definition we extend the subprocess relation $\leq$ to a subprocess relation $\leq_{CS'}^{CS}$ respecting given commitment set mappings of extended branching processes. This means that the commitment sets are preserved under the subprocess homomorphism $h_{\leq}$.

Definition 8 (Subprocess relation of extended branching processes with commitment set mapping). *Let B_1 and B_2 be extended branching processes of a safe Petri net N_0 with commitment set mappings CS_1 and CS_2. B_1 approximates B_2 with respect to the commitment set mappings CS_1 and CS_2, denoted $B_1 \leq_{CS_2}^{CS_1} B_2$, if and only if $B_1 \leq B_2$ and for all $p \in \mathcal{P}_1$ for the commitment set $CS_1(p) = CS_2(h_{\leq}(p))$ holds.*

Note that this defines a partial order on tuples of extended branching processes and their commitment set mappings.

Now, we can define planning segments as tuples of extended branching processes and a commitment set mapping, such that every node necessary to satisfy the local synchronisation condition for every maximally progressed place is added and no more nodes are included. We call a place maximally progressed, if and only if it satisfies the *LSC* or has an empty postset. Thus, either the *LSC* is satisfied or no further transitions are allowed in the commitment set mapping. Exactly those transitions which are allowed in the commitments set are added until the *LSC* is satisfied.

Definition 9 (Planning segment). *A tuple (B, CS) consisting of an extended branching process $B = (N, \pi)$ with $N = (\mathcal{P}, \mathcal{T}, pre, post, M)$ of a Petri game G_P*

with underlying Petri net N_0 and a commitment set mapping CS of B is a planning segment*, if and only if the following holds:*

$$\forall t \in \mathcal{T} : \forall p \in pre(t) : \pi(t) \in CS(p) \\ \wedge \exists t' \in \mathcal{T} : (t \leq t' \wedge \exists s \in pre(t') : s \text{ does not satisfy the } LSC)) \quad (1)$$

$$\neg\exists (B', CS') : (B', CS') \text{ satisfies } (1) \wedge B \leq^{CS}_{CS'} B' \quad (2)$$

Let $PSeg(G)$ denote the set of all planning segments of a Petri game G. The notation T_v^{En} is used for a planning segment $v \in PSeg(G)$. This is the set of enabled transitions in the initial marking of its branching process B.

Note: The second part of property (1): $\exists t' \in \mathcal{T} : (t \leq t' \wedge \exists s \in pre(t') : s$ does not satisfy $LSC)$ ensures that every added transitions is necessary to satisfy the LSC. Thereby, it allows transitions, that are causal predecessors of other transitions, where the preset does not satisfy the local synchronisation condition, to be added despite the possibility that all places in the preset of the transition itself already satisfy the local synchronisation condition. The property (2) of such a tuple ensures that all such transitions are added by requiring the tuple to be maximal.

3.2 Corresponding Graph Game

The set of all planning segments is used as the set of vertices of the environment player in the graph game. In the following, we define decision sets that are the possible decisions for the environment player in one of its vertices. This means that every decision set corresponds to an outgoing edge from an environment vertex in the graph game. A decision set consists of a set of concurrent transitions that are enabled in the current marking. The environment player chooses to fire these transitions.

Definition 10 (Decision sets). *For a Petri game G_P and a planning segment $(B, CS) \in PSeg(G)$ with initial marking In of B, the set of decision sets for the environment player is defined as follows:*

$$DSets((B, CS)) = \{DS \subseteq T_{In}^{En} \mid \forall t_i, t_j \in DS, t_i \neq t_j : t_i || t_j\}$$

An example of a branching process of a planning segment is shown in Fig. 2, if the places in the post sets of the transitions t_{f2} and t_{f3} are added, which are indicated with vertical dots. The places after firing a transition t_{fi}, $i = 1, \ldots, 4$, satisfy the local synchronisation condition as they know which port each environment player chose. The branching process in Fig. 2 is a planning segment, if a suitable commitment set mapping is added.

In the following, we define deterministic and deadlock avoiding planning segments. The intuition here is quite similar to the determinism and deadlock avoidance of a winning strategy in a Petri game: eventually, assuming exactly those transitions allowed in the commitment sets are added to the branching process the definition is equal for the places and transitions that are actually in the branching process. The definitions are not only for planning segments but for extended branching processes with a commitment set mapping.

Definition 11 (Deterministic and deadlock avoiding planning segments). *Let G_P be a Petri game. An extended branching process with a commitment set mapping (B, CS), where $B = (N, \pi)$ and $N = (\mathcal{P}, \mathcal{T}, pre, post, In)$, is called* deterministic *in a reachable marking $M \in \mathcal{R}(B)$, if for all $p \in M$ with $\pi(p) \in \mathcal{P}^S$ exist at most one transition $t \in CS(p)$, where $pre_0(t) \subseteq \pi(M)$ and for all $p \in \pi \restriction_M^{-1} (pre_0(t)) : t \in CS(p)$ holds.*

An extended branching process with a commitment set mapping (B, CS) is called deadlock avoiding *in a reachable marking $M \in \mathcal{R}(B)$ if and only if the following holds: If a transition $t \in \mathcal{T}_0$ is enabled in $\pi(M)$, then there exists a transition $t' \in \mathcal{T}_0$, where for all $p \in \pi \restriction_M^{-1} (pre_0(t')) : t' \in CS(p)$ holds.*

The set of bad planning segments is defined as expected.

Definition 12 (Bad planning segments). *Let G_P be a Petri game and let $PSeg(G_P)$ denote the set of all planning segments. The set of* bad planning segments *is defined as:*

$$\begin{aligned} PSeg_{bad} = \{&(B, CS) \in PSeg \mid \exists M \in \mathcal{R}(B) : \pi(M) \in \mathcal{B} \\ &\vee (B, CS) \text{ is not deterministic or not deadlock avoiding in } M.\} \end{aligned}$$

Now, turn to the graph game. We define the starting vertices of the system player in the graph game. A winning strategy has to be winning starting in one of those vertices to construct a winning strategy in the Petri game later on. A starting vertex of the graph game consists of the branching process with just the places of the initial marking, their arbitrary commitment sets and no transitions. Note the pre-image of *pre* and *post* are empty.

Definition 13 (Starting vertices). *The set of starting vertices in the graph game for a Petri game $G_P = (\mathcal{P}_0^S, \mathcal{P}_0^E, \mathcal{T}_0, pre_0, post_0, In_0, \mathcal{B})$ is defined as:*

$$\begin{aligned} Start = \{&(B = ((In_0, \emptyset, pre, post, In_0), \pi), CS) \mid \\ &CS \text{ is a commitment set mapping of } B\} \end{aligned}$$

The reduction of a Petri game to a graph game is now defined as follows: the set of environment vertices V_1 is the set of all planning segments. The set of system vertices V_0 is the union of the set of starting vertices and the set of extended branching processes with commitment set mapping, that are reached by firing a decision set of an arbitrary planning segment. The commitment sets remain the same for the remaining places. The system vertices are denoted as pairs with a 0 in the second component to distinguish them from the environment vertices; the first component of a system vertex might be a planning segment. The set of directed edges consists of three sets: First, firing a decision set resembles an edge from the planning segment to a system vertex. Second, the system player has to plan further ahead to meet the local synchronisation condition resulting in an edge from a system vertex to a planning segment. Third, the graph game loops if no transition is enabled. All edges preserve a subprocess relation in their direction. The commitment sets for all those places kept due to the subprocess relation remain the same. The set of all safe vertices are all vertices that are not bad planning segments.

Definition 14 (Corresponding graph game). *For a Petri game* $G_P = (\mathcal{P}_0^S, \mathcal{P}_0^E, \mathcal{T}_0, pre_0, post_0, In_0, \mathcal{B})$ *we define a safety graph game* $G_{Graph} = (\mathcal{A}, S)$, $\mathcal{A} = (V, V_0, V_1, E)$ *as follows:*

$$\boldsymbol{V_1} = PSeg(G_p)$$

$$\boldsymbol{V_0} = \{((\tilde{B}', CS'), 0) \mid \exists (B, CS) \in V_1 : \exists DS \in DSets((B, CS)) : \tilde{B}' = \Phi(B|DS\rangle) \wedge CS' = CS \restriction_{\mathcal{P}'} \circ \Phi^{-1}\} \cup Start \times \{0\}$$

$$\boldsymbol{E} = \{(((B, CS), 0), (B', CS') \in V_0 \times V_1 \mid B \leq_{CS'}^{CS} B'\}$$
$$\cup \{((B, CS), ((\tilde{B}', CS'), 0)) \in V_1 \times V_0 \mid \exists DS \in DSets((B, CS)) : \tilde{B}' = \Phi(B|DS\rangle) \wedge CS' = CS \restriction_{\mathcal{P}'} \circ \Phi^{-1}\}$$
$$\cup \{((B, CS), (B, CS) \in V_1 \times V_1 \mid T_B^{En} = \emptyset\}$$

$$\boldsymbol{S} = V_1 \cup V_0 \setminus PSeg_{bad}(G_p)$$

The sets $\mathcal{P}$ *and* $\mathcal{P}'$ *denote the places of the branching processes* B *and* B'*, respectively.* Φ *denotes the isomorphism from the branching process* B' *to the canonical branching process* $\tilde{B}'$*. We refer to this graph game as the* corresponding *graph game of* G*.*

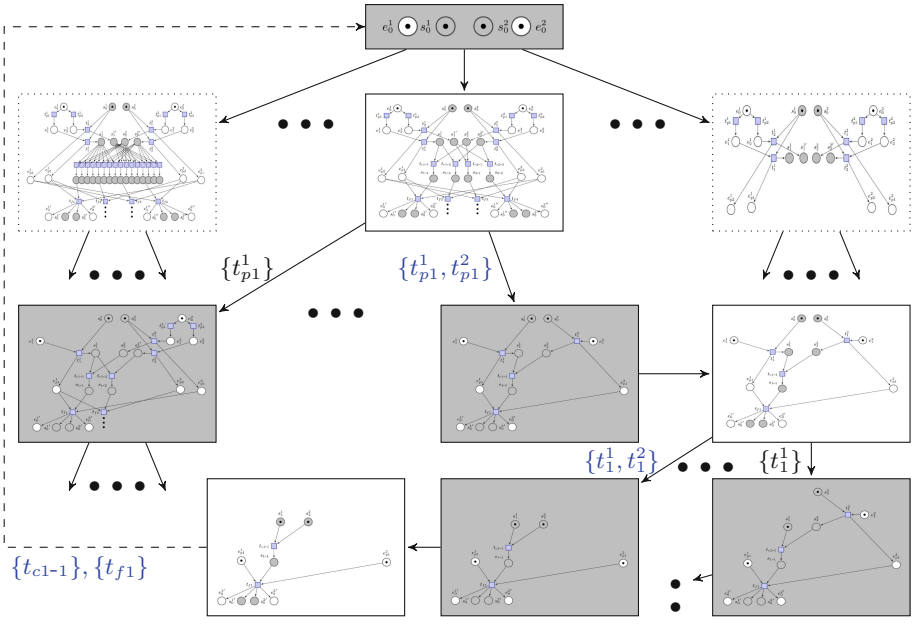

Fig. 3. A part of the corresponding safety graph game of the Petri game in Fig. 1 starting in the vertex, where the commitment set mapping allows all transitions. The commitment set mappings are implicit within the branching processes. White vertices belong to the environment player and gray to the system player. The dotted white vertices are bad planning segments. Edges are annotated with their decision sets. Decision sets with unique planning are blue, later defined in Definition 17.

The graph game is almost bipartite: As long as $\mathcal{T}_{In}^{En} \neq \emptyset$, vertices of the system player and the environment player alternate. This is not necessary when the extended branching process with its commitment set mapping of a vertex of the system player is already a planning segment again. The graph game is kept bipartite in that case for formal simplicity reasons.

In Fig. 3, a part of the corresponding graph game starting in the vertex, that allows all transitions in the initial marking, is shown. The planning segments represent all possible combinations of commitment sets, from the unfolding on the left over the initial part of a winning strategy in Fig. 2 to not allowing any transition on the right. The edges are annotated with decision sets. A blue decision set belongs to a play with unique planning, later defined in Definition 17. The play following the blue decision sets is winning. Note that the *LSC* is always satisfied after a transition t_{fi}.

Lemma 1 (Corresponding graph game has exponential size). *The corresponding graph game of a Petri game has exponential size.*

Proof (Proof sketch).[1] The corresponding graph game has exponential size as the *LSC* ensures that every token has to hear from every other token within taking n own transitions. The combinations of commitment sets in the system places result in exponentially many planning segments, which results in an exponential size of the corresponding graph game.

3.3 From Graph Games to Petri Games and Vice Versa

First in this section, we prove the existence of a winning strategy in a Petri game if a winning strategy exists in its corresponding graph game. Starting with a winning strategy in the graph game the idea of the construction of the Petri game strategy is simple: Every node added in some branching process of a consistent play with unique planning is added to the Petri game strategy and no more.

In the following, we fix the Petri game $G_P = (\mathcal{P}_0^S, \mathcal{P}_0^E, \mathcal{T}_0, pre_0, post_0, In_0, \mathcal{B})$ and the underlying Petri net of a Petri game always satisfies the non-simultaneous synchronisation condition from Definition 3, $G = (\mathcal{A}, S)$ is the corresponding graph game from Definition 14, and σ is a strategy of G.

The next two definitions lead to the definition of unique planning that prevents different planning by varying the order of a firing sequence. A play in the graph game gets annotated with the decision sets chosen by the environment:

Definition 15 (Annotation of plays). *We annotate a play of the graph game* $\mu = v_1 v_2 v_3 \ldots$ *with its decision sets:*

If $(v_n, v_{n+1}) = ((B, CS), ((\tilde{B}', CS'), 0)) \in V_1 \times V_0$ *and* $DS \in DSets(B, CS)$: $\tilde{B}' = \Phi(B|DS\rangle) \wedge CS' = CS \restriction_{\mathcal{P}' \circ \Phi^{-1}}$ *we write* $v_n \xrightarrow{DS} v_{n+1}$.

[1] Full proofs will appear in an extended version of this paper on arXiv.

We define the canonical branching process of a play in the graph game as the branching process yielded by sequentially attaching the planning segments of the play. As every planning segment itself has canonical names, the names in the branching process of a play are not the same as in the planning segments in general. A subprocess homomorphism $h_{\leq}$ for each planning segment maps its places and transitions to its part of the branching process of the play. The commitment set mapping of the branching process of a play is compounded by the commitment set mappings of the planning segments.

Definition 16 (Branching process and commitment set mapping of a play). *Let $\mu = v_1 v_2 v_3 \dots$ be a play or a prefix of a play of G and $DS_1 DS_2 \dots$ the sequence of annotated decision sets. The branching process of the play μ, $B_\mu = (N_\mu, \pi_\mu)$, is defined as the smallest canonical branching process with respect to $\leq$ such that the sequence of annotated decision sets can be fired in B_μ. We refer to the components of N_μ as $\mathcal{P}_\mu, \mathcal{T}_\mu, pre_\mu, post_\mu$, and In_μ.*

For all prefixes $v_1 v_2 \dots v_n$ with $v_n = (B_{v_n}, CS_{v_n}) \in V_1$ and its sequence of annotated decision sets $DS_1 DS_2 \dots DS_m$, we define a part of the planning segment v_n in B_μ, denoted $\tilde{v}_n$, as follows: $\tilde{v}_n = (B_{\tilde{v}_n}, CS_{\tilde{v}_n})$, where $B_{\tilde{v}_n} = h_{\leq}(B)$ and B is the maximal canonical branching process with respect to $\leq$ such that $B \leq B_\mu | DS_1 DS_2 \dots DS_m \rangle$ with the subprocess homomorphism $h_{\leq} : B \mapsto B_\mu | DS_1 DS_2 \dots DS_m \rangle$ and $B \leq B_{v_n}$. The components of $B_{\tilde{v}_n}$ are denoted as $\mathcal{P}_{\tilde{v}_n}, \mathcal{T}_{\tilde{v}_n}, pre_{\tilde{v}_n}$, $post_{\tilde{v}_n}$ and $In_{\tilde{v}_n}$. For all $p \in \mathcal{P}_{\tilde{v}_n}$, the commitment set mapping $CS_{\tilde{v}_n}(p)$ is defined as $CS_{v_n}(p_n)$ if $h_{\leq}(p_n) = p$. The commitment set mapping $CS_\mu(p)$ of B_μ for all $p \in \mathcal{P}_\mu$ is defined as $CS_\mu(p) = CS_{\tilde{v}_n}(p)$, if $p \in \mathcal{P}_{\tilde{v}_n}$.

Note that the compound commitment set mapping CS_μ is well defined as already existing commitment sets are preserved by the edges in the graph game.

Now we define the set of plays with unique planning. Firing the same set of transitions in a different order can result in differently planned branching processes in the graph game. Those different branching processes might not result in a winning strategy for the Petri game when they are added together. We only consider those plays in the graph game, where every transition is fired as soon as possible or never. This means that every transition chosen in a decision set must not be enabled in an earlier planning segment.

Definition 17 (Set of plays with unique planning). *Let $Plays(G, \sigma)$ be the set of consistent plays and let $DS_1^\mu DS_2^\mu \dots$ denote the sequence of annotated decision sets of a play $\mu \in Plays(G, \sigma)$. The set of consistent plays with unique planning is defined as*

$$\begin{aligned} Plays_u(G,\sigma) = \{\mu \in Plays(G,\sigma) \mid \forall t \in DS_j^\mu, j \geq 2 : t \notin \mathcal{T}^{En}_{v_{DS_{j-1}^\mu}} \\ \vee \exists t' \in DS_{j-1}^\mu : pre_{v_{DS_{j-1}^\mu}}(t') \cap pre_{v_{DS_j^\mu}}(t) \neq \emptyset\}, \end{aligned}$$

where $v_{DS_{j-1}^\mu}$ is the vertex of the play μ with outgoing annotation DS_{j-1}^μ and $pre_{v_{DS_i^\mu}}$ the preset mapping of the branching process of $v_{DS_i^\mu}$.

This definition makes use of the canonical names. An enabled transition in the initial marking has the same name for all instances of that transition. If a transition is enabled in the initial marking of a previous planning segment, we do not allow that transition in the decision set. But, we have to consider that the transition in the current planning segment can be another instance of the transition of the previous planning segment that just has the same name. This is the case, if a transition t_0 of the Petri net is enabled right again after firing itself or after firing a transition in conflict to the previous instance of t_0 such that an instance with the same name is in $\mathcal{T}^{En}_{v_{DS^\mu_{j-1}}}$, but the second instance cannot be fired earlier. The condition $\exists t' \in DS^\mu_{j-1} : pre_{v_{DS^\mu_{j-1}}}(t') \cap pre_{v_{DS^\mu_j}}(t) \neq \emptyset$ ensures that a second instance with the same name can be chosen in the decision set.

The idea of the construction of the winning strategy in the Petri game is to merge the branching processes of every possible and to the graph game strategy consistent unique play. We define the union of two branching processes componentwise. As the proof of the construction of a winning strategy in the Petri game is by induction, we define a winning prefix and its union naturally. The branching processes of prefixes of winning plays in the graph game are winning prefixes.

In the following we show three crucial properties of the branching process of a single play consistent to the winning strategy. The first property is that every transition has to be allowed in the commitment sets of its preset.

Lemma 2 (All transitions are allowed in commitment sets). *For the branching process B_μ of a play $\mu \in Plays(G, \sigma)$ it holds that for every transition $t \in \mathcal{T}_\mu$ for all $p \in pre_\mu(t) : t \in CS_\mu(p)$.*

Proof. As in the definition of a planning segment, every transition has to be in the commitment set of every place in its preset.

The second property is that every reachable marking in the branching process of a play is reachable in one of its planning segments.

Lemma 3 (Every reachable marking is reachable in a planning segment). *For the branching process B_μ of a play $\mu \in Plays(G, \sigma)$ it holds that for every reachable marking $M \in \mathcal{R}(B_\mu)$ exists a planning segment $v \in V_1$ in μ, where M is reachable in its part $B_{\tilde{v}}$ of the branching process B_μ.*

Proof (Proof sketch). We need to distinguish two cases. In the first case, a transition $t \in \mathcal{T}_\mu$ is enabled in M. Then we can take the prefix $v_1 v_2 \dots v_n$ of μ, where $v_n \in V_1$ is the first vertex, where In_{v_n} enables a transition $t' \in \mathcal{T}_\mu$, which is enabled in M. Every reachable marking in a branching process is a set of pairwise concurrent places and the maximally progressed places in v_n either satisfy the *LSC* or no further transitions are allowed. Thus, as those places are either not concurrent to the places in $pre(t')$ or no concurrent places can be added later on, M is reachable in $B_{\tilde{v}_n}$.

In the second case, a planning segment v_i is reached eventually, where M is the initial marking $In_{\tilde{v}_i}$.

The third property is that every transition allowed in the commitment sets of its preset is added in a branching process of some play with unique planning.

Lemma 4 (All allowed transitions are added). *Let $\mu = v_1 v_2 \ldots \in Plays_u(G, \sigma)$ be a consistent play with unique planning. If M is a reachable marking in v_i such that no transition in the past of a place of M could have been fired in a previous planning segment of μ and $\exists t_0 \in \mathcal{T}_0 : pre_0(t_0) \subseteq \pi_{v_i}(M) \wedge \forall p \in \pi_{v_i} \restriction_M^{-1} (pre_0(t_0)) : t_0 \in CS_{v_i}(p)$, then there exists a unique play $\mu' \in Plays_u(G, \sigma)$ with a same prefix $\mu^{|i|} = \mu'^{|i|}$, and a vertex v'_j that is reached in μ' with $j \geq i$ via the sequence of annotated decision sets $DS_1 \ldots DS_n$ from $v'_i \ldots v'_j$ such that there exists $t \in \mathcal{T}_{v'_j}$ with $pre_{v'_j}(t) = \Phi(\pi_{v_i} \restriction_M^{-1} (pre_0(t_0)))$ and $\pi_{v'_j}(t) = t_0$, where Φ is the isomorphism from $B|DS_1 \ldots DS_n\rangle$ to its canonical branching process.*

Proof. Let $t_0 \in \mathcal{T}_0$ be such a transition in v_i. Since M is a reachable marking in v_i, we construct a play $\mu' \in Plays_u(G, \sigma)$, where the environment player chooses its decision sets in a way that every transition in the past of any place of M is chosen as soon as possible and no transition in conflict to one of the places in M is chosen. Since B_μ is finitely preceded it follows that a planning segment v'_j, $j \geq i$, is reached, where a place $p \in \Phi(\pi_{v_i} \restriction_M^{-1} (pre_0(t_0)))$ is in the initial marking of v'_j, and the commitment sets of the places remained the same due to the construction of the arena A. The place p does not satisfy the LSC of v'_j and since v'_j is maximal with respect to its commitment set mapping, we have $t \in \mathcal{T}_{v'_j}$ with $pre_{v'_j}(t) = \Phi(\pi_{v_i} \restriction_M^{-1} (pre_0(t_0)))$ and $\pi_{v'_j}(t) = t_0$.

Lemma 5 (From graph games to Petri games). *If a winning strategy $\sigma : V^*V_0 \to V$ for player 0 exists in G starting in a vertex $v_1 \in Start$, then there exists a winning strategy σ' in the Petri game G_P.*

Proof (Proof sketch). We show that the union of the branching processes of all plays with unique planning is winning in the Petri game. For the maximally progressed places in a planning segment that satisfy the LSC, all decision sets, which are allowed in plays with unique planning, are distinguishable. This means that each of those places refers to exactly one such decision set. Thus, for two different decision sets the maximally progressed places are disjoint and further transitions added later in the play have disjoint presets, such that every node is added only in one branching process of a prefix of a play with unique planning. For the maximally progressed places, that do not satisfy the LSC, no transition is allowed to be added due to the commitment sets. From there on, the commitment set is kept the same in all plays with that prefix.

Now, the three shown properties for branching processes of a play with unique planning together with σ being a winning strategy ensure the winning properties of the Petri game strategy: Lemma 2 and Lemma 4 ensure the justified refusal property; Lemma 3 ensures the safety property; Lemma 2 and Lemma 3 ensure the determinism property; Lemma 3 and Lemma 4 ensure the deadlock avoiding property.

Now, we show the existence the other way round.

Lemma 6 (From Petri games to graph games). *Let G_P be a Petri game and σ a winning strategy for G_P. Then there exists a winning strategy σ' for player 0 in the safety graph game $G = (\mathcal{A}, S)$ from Definition 14.*

Proof (Proof sketch). We choose the planning segments according to the given Petri game strategy, where the commitment sets are equal to the postsets of the places in the Petri game strategy.

We conclude with the theorem that states that the synthesis problem for Petri games of the presented subclass is decidable in EXPTIME.

Theorem 1 (Synthesis). *Let G_P be a Petri game, where the underlying Petri net satisfies the non-simultaneous synchronisation condition from Definition 3, then the existence of a winning strategy, and deriving it if existent, is decidable in EXPTIME.*

Proof. The equivalence of the existence of a winning strategy follows from the two implications in Lemma 5 and Lemma 6. The corresponding graph game has exponential size dependent on the size of the Petri game. The graph game takes linear time to be solved dependent on the number of edges, which are at most square as many as vertices. This results in exponential complexity overall.

4 Related Work

There have been quite a few approaches to the synthesis problem. We distinguish between works that address a single-process synthesis problem, where the system consists of one process that has all its information accessible, and works that address a multi-process synthesis problem, where multiple processes exist that are all partially informed. One of the latter is presented in this paper. The works on Petri nets in [5,23] belong to the former.

The synthesis problem was first introduced in [6]. Pnueli and Rosner introduced a setting of synchronous processes that communicate via shared variables [22]. For a single process, this setting is known to be decidable [4,21]. For multiple processes, this setting is known to be undecidable [22]. In particular, information forks have been found to be a necessary and sufficient criterion for the undecidability in that distributed setting [14]. There have been positive decidability results on specific architectures with multiple processes, including pipelines [24], rings [16], and acyclic architectures [14]. However, all the positive results for multiple processes have non-elementary complexity. A general game model in this type of setting is introduced by Walukiewicz and Mohalik [19]. Another line of work concerns the alternating-time temporal logics, which are interpreted over concurrent game structures [1].

Petri nets are conceptually connected to event structures by their unfoldings [18,20]. As an application example, unfoldings are used to determine the set of all reachable markings in a Petri net [8]. We use net unfoldings to define strategies

on Petri games. The causal past of a node is the only available information; concurrent and future actions are invisible.

Zielonka automata are another distributed setting introduced in [25]. These are weakly bisimilar on their winning strategies to Petri games that have a local safety condition, with an exponential blow-up [3]. The synthesis problem is decidable for games on acyclic Zielonka automata [15] and on Zielonka automata with a strong synchronisation condition together with constraints on the winning condition that allow no distinction of differently ordered executions of the same trace [17]. We allow such distinctions, but the synchronisation condition in this paper is similar to the one in [17] and restricts the Petri nets to those that only allow loops including all tokens or that are acyclic.

5 Conclusions

We have presented a subclass of Petri games with an arbitrary mixture of system and environment players for which the synthesis problem is decidable in EXPTIME. Petri games use the tokens as carriers of information and link their information flow to their causality. This makes Petri games a suitable formalism to reason about distributed applications. The presented approach might seem anti-intuitive as it does not use the causal past of the players; it uses the causality to plan ahead. This subclass allows us to model distributed systems with a hierarchic communication structure, where every part has to check on its subordinated parts within a bound. Every part may consist of multiple processes itself, that communicate repeatedly within a bound. For example, we can model a control for several traffic lights geared to each other which react to the current traffic situation. A failure of a traffic light could be modeled within the Petri game. Another possible modeling is the communication structure of a round robin protocol. We cannot express exact timing constraints in Petri games.

This is the first work approaching the synthesis problem for distributed systems that allows finitely many system players and finitely many environment players while the global safety condition allows to distinguish between different interleavings of the same trace. Also, we allow tokens to be generated or deleted, which makes these Petri games a convenient way to model resource allocations and situations where processes are generated or deleted. This is not possible in the setting of Zielonka automata.

In future work, we will investigate weaker synchronisation conditions. We will also implement the presented decision procedure and compare it to existing ones in [9,11]. Another challenge is to extend the winning condition to reachability, Büchi, or parity conditions. A reachability condition is harder to check as we need to ensure that for all plays a designated marking is reached in every possible order of firing its transitions, which is not possible to check within the presented planning segments. In [10], it is shown that Petri games with a global reachability condition are undecidable.

References

1. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. J. ACM **49**(5), 672–713 (2002)
2. Alur, R., Madhusudan, P., Nam, W.: Symbolic computational techniques for solving games. Int. J. Softw. Tools Technol. Transf. **7**(2), 118–128 (2005)
3. Beutner, R., Finkbeiner, B., Hecking-Harbusch, J.: Translating asynchronous games for distributed synthesis. In: Fokkink, W.J., van Glabbeek, R. (eds.) 30th International Conference on Concurrency Theory, CONCUR 2019, Amsterdam, The Netherlands, 27–30 August 2019. LIPIcs, vol. 140, pp. 26:1–26:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019). https://doi.org/10.4230/LIPIcs.CONCUR.2019.26
4. Bloem, R., Schewe, S., Khalimov, A.: CTL* synthesis via LTL synthesis. In: Fisman, D., Jacobs, S. (eds.) Proceedings Sixth Workshop on Synthesis, SYNT@CAV 2017, Heidelberg, Germany, 22nd July 2017. EPTCS, vol. 260, pp. 4–22 (2017). https://doi.org/10.4204/EPTCS.260.4
5. Buy, U.A., Darabi, H., Lehene, M., Venepally, V.: Supervisory control of time Petri nets using net unfolding. In: 29th Annual International Computer Software and Applications Conference, COMPSAC 2005, Edinburgh, Scotland, UK, 25–28 July 2005, vol. 2. pp. 97–100. IEEE Computer Society (2005). https://doi.org/10.1109/COMPSAC.2005.148
6. Church, A.: Applications of recursive arithmetic to the problem of circuit synthesis. Summ. Summer Inst. Symb. Logic **1**, 3–50 (1957)
7. Engelfriet, J.: Branching processes of Petri nets. Acta Inf. **28**(6), 575–591 (1991)
8. Esparza, J.: Model checking using net unfoldings. Sci. Comput. Program. **23**(2), 151–195 (1994)
9. Finkbeiner, B.: Bounded synthesis for petri games. In: Meyer, R., Platzer, A., Wehrheim, H. (eds.) Correct System Design. LNCS, vol. 9360, pp. 223–237. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23506-6_15
10. Finkbeiner, B., Gieseking, M., Hecking-Harbusch, J., Olderog, E.: Global winning conditions in synthesis of distributed systems with causal memory. CoRR abs/2107.09280 (2021). https://arxiv.org/abs/2107.09280
11. Finkbeiner, B., Gieseking, M., Olderog, E.-R.: Adam: causality-based synthesis of distributed systems. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 433–439. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21690-4_25
12. Finkbeiner, B., Gölz, P.: Synthesis in distributed environments. In: Lokam, S., Ramanujam, R. (eds.) 37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017). Leibniz International Proceedings in Informatics (LIPIcs), vol. 93, pp. 28:1–28:14. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany (2018). https://doi.org/10.4230/LIPIcs.FSTTCS.2017.28. http://drops.dagstuhl.de/opus/volltexte/2018/8406
13. Finkbeiner, B., Olderog, E.R.: Petri games: synthesis of distributed systems with causal memory. Inf. Comput. **253**, 181–203 (2017)
14. Finkbeiner, B., Schewe, S.: Uniform distributed synthesis. In: Proceedings of the 20th IEEE Symposium on Logic in Computer Science (LICS 2005), Chicago, IL, USA, 26–29 June 2005, pp. 321–330 (2005). https://doi.org/10.1109/LICS.2005.53

15. Genest, B., Gimbert, H., Muscholl, A., Walukiewicz, I.: Asynchronous games over tree architectures. In: Fomin, F.V., Freivalds, R., Kwiatkowska, M., Peleg, D. (eds.) ICALP 2013. LNCS, vol. 7966, pp. 275–286. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39212-2_26
16. Kupferman, O., Vardi, M.: Synthesizing distributed systems. In: Proceedings - Symposium on Logic in Computer Science, pp. 389–398 (2001). https://doi.org/10.1109/LICS.2001.932514. https://www.scopus.com/inward/record.uri?eid=2-s2.0-0034873871&doi=10.1109
17. Madhusudan, P., Thiagarajan, P.S., Yang, S.: The MSO theory of connectedly communicating processes. In: Sarukkai, S., Sen, S. (eds.) FSTTCS 2005. LNCS, vol. 3821, pp. 201–212. Springer, Heidelberg (2005). https://doi.org/10.1007/11590156_16
18. Meseguer, J., Montanari, U., Sassone, V.: Process versus unfolding semantics for place/transition petri nets. Theoret. Comput. Sci. **153**(1), 171–210 (1996)
19. Mohalik, S., Walukiewicz, I.: Distributed games. In: Pandya, P.K., Radhakrishnan, J. (eds.) FSTTCS 2003. LNCS, vol. 2914, pp. 338–351. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-540-24597-1_29
20. Nielsen, M., Plotkin, G., Winskel, G.: Petri nets, event structures and domains, part I. Theoret. Comput. Sci. **13**(1), 85–108 (1981)
21. Pnueli, A., Rosner, R.: On the synthesis of an asynchronous reactive module. In: Ausiello, G., Dezani-Ciancaglini, M., Della Rocca, S.R. (eds.) ICALP 1989. LNCS, vol. 372, pp. 652–671. Springer, Heidelberg (1989). https://doi.org/10.1007/BFb0035790
22. Pnueli, A., Rosner, R.: Distributed reactive systems are hard to synthesize. In: 31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, 22–24 October 1990, vol. II, pp. 746–757 (1990). https://doi.org/10.1109/FSCS.1990.89597
23. Raskin, J., Samuelides, M., Begin, L.V.: Petri games are monotone but difficult to decide. Technical report, Université Libre De Bruxelles (2003)
24. Rosner, R.: Modular synthesis of reactive systems. Ph.D. thesis, Weizmann Institute of Science, Rehovot, Israel (1992)
25. Zielonka, W.: Notes in finite asynchronous automata. RAIRO - Theoret. Inform. Appl. - Informatique Théorique et Applications **21**(2), 99–135 (1987)