Bounded Synthesis for Petri Games

Bernd Finkbeiner(✉)

Universität des Saarlandes, Saarbrücken, Germany
finkbeiner@cs.uni-saarland.de

Abstract. Petri games, introduced in recent joint work with Ernst-Rüdiger Olderog, are an extension of Petri nets for the causality-based synthesis of distributed systems. In a Petri game, each token is a player in a multiplayer game, played between the "environment" and "system" teams. In this paper, we propose a new technique for finding winning strategies for the system players based on the bounded synthesis approach. In bounded synthesis, we limit the size of the strategy. By incrementally increasing the bound, we can focus the search towards small solutions while still eventually finding every finite winning strategy.

1 Introduction

The ambition to translate formal *specifications* into executable *programs*, and to do so *automatically*, without a human programmer, dates back to the early beginnings of computer science [1,2,10,19]. In the area of *reactive systems*, which includes hardware circuits, communication protocols, and generally all systems that interact continuously with their environment, the first formalization of the problem is generally attributed to Alonzo Church at the Summer Institute of Symbolic Logic in 1957 at Cornell University:

> Given a requirement which a circuit is to satisfy [...]. The synthesis problem is then to find recursion equivalences representing a circuit that satisfies the given requirement (or alternatively, to determine that there is no such circuit) [2].

Synthesis algorithms have the potential to dramatically simplify the development of complex systems. Instead of manually writing a program, one only needs to specify the actions available to the system and the objective, or *winning condition*, that one would like the system to guarantee against all possible behaviors of the system's environment. A strategy that achieves the winning condition by reacting to the actions of the environment with appropriate system actions is then constructed automatically.

Over the years, Church's synthesis problem has been studied in many variations. Of particular interest for the purposes of this paper is the problem of synthesizing *distributed* systems [6–9,13,15,16,18,20]. Many modern reactive

This research was partially supported by the German Research Council (DFG) in the Transregional Collaborative Research Center SFB/TR 14 AVACS.

R. Meyer et al. (Eds.): Olderog-Festschrift, LNCS 9360, pp. 223–237, 2015.
DOI: 10.1007/978-3-319-23506-6_15

systems are distributed in the sense that they consist of multiple processes with individual inputs, which they may or may not share with other processes. A key challenge in the design of such systems is to decide how the processes should interact so that each process obtains the information needed to carry out its functionality.

Computationally, the synthesis of distributed systems is a very hard problem, often with nonelementary lower bounds on the complexity [9]. An encouraging observation is, however, that practical specifications often have reasonably small implementations. This observation is exploited by the *bounded synthesis* approach [7,20], which restricts the space of potential solutions by some (iteratively growing) bound. Despite the uncomfortable lower bounds on the worst-case behavior of the synthesis algorithms, bounded synthesis is often able to find an implementation in reasonable time, as long as the solution is small with respect to a suitable parameter such as the number of states of the implementation.

In this paper, we propose a first bounded synthesis approach for Petri games. *Petri games* [5], introduced in recent joint work with Ernst-Rüdiger Olderog, are an extension of Petri nets for the causality-based synthesis of distributed systems. In a Petri game, each token is a player in a multiplayer game, played between the "environment" and the "system" team. In the tradition of Zielonka's automata [21], Petri games model distributed systems with *causal memory*, i.e., the processes memorize their causal histories and learn about each other's histories when synchronizing. The environment tokens represent independent sources of input, such as different users of the system. The system players represent the processes of the system. Each system player is only allowed to act on information it actually knows, either through direct interaction with the environment, or indirectly, through synchronization with other system players. Since the different system players have the same objective but different knowledge about the system state, a winning strategy usually involves an active synchronization between the system players to ensure that every player has the knowledge needed to win the game. A good example for the type of system that can be constructed with Petri games is the distributed burglar alarm system discussed in detail in [5]: a break-in may occur at one of several locations, and the alarm system at that location must inform the other distributed components about this, so that the alarm can be activated in all locations. Petri games have also been used to synthesize controllers for robots in production plants [4], where the Petri net is used to capture the concurrency, usage constraints, and uncertain availability of machines in the plant. The winning condition is to accomplish certain tasks, such as to process a certain number of orders on a certain number of machines, despite the actions of the hostile environment, which may declare a subset of the machines to be defect.

Strategies in a Petri game are, in general, infinite objects, because they are defined in terms of the (infinite) unfolding of the net. The reason for this construction is that different places in the unfolding reflect different causal histories; this enables the strategy to act depending on the individual history of a player. In practice we are, however, mostly interested in winning strategies that can be represented by a finite net. For the special case of a single environment token,

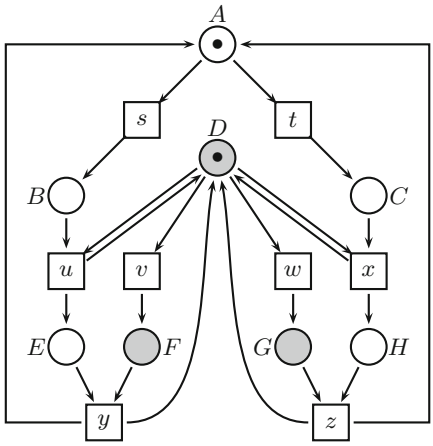

Fig. 1. Example Petri game. Places belonging to the system player are shown in gray, all other places belong to the environment player. The set of bad markings consists of the markings $\{B, G\}$ and $\{C, F\}$, where the net has reached a deadlock. The system player can win the game by waiting in place D until a synchronization with the environment token in places B or C becomes possible. The move after the synchronization then depends on whether the synchronization was via transition u or via transition x. In case of u, the system player takes transition v, thus avoiding $\{B, G\}$; in case of x, the system player takes w, thus avoiding $\{C, F\}$.

the existence of a winning strategy can be decided, and a finite representation of the winning strategy can be constructed in single-exponential time via a reduction to two-player games over finite graphs [5]. This synthesis algorithm has been implemented as a BDD-based fixed point iteration in the tool Adam [4]. The bounded synthesis approach of the present paper complements the symbolic algorithm of Adam with a satisfiability-based approach. We bound the size of the solutions of interest by setting a bound on the number of instances of each place of the Petri game. The existence of a winning strategy is then encoded as a quantified boolean formula, where the choices of the strategy appear as existentially quantified variables. We use a QBF solver to extract a satisfying assignment to these variables, which defines a winning strategy that meets the specified bounds.

The remainder of the paper is structured as follows. We begin in Section 2 with a review of the main notions for Petri games from [5]. In Section 3, we define *bounded* strategies. Section 4 then presents an encoding of the existence of winning strategies as a quantified boolean formula. In Section 5 we discuss the analysis of trade-offs, in particular the trade-off between the memory allocated to different players, and the trade-off between memory and proof complexity.

2 Petri Games

Petri games were introduced in [5] as an extension of Petri nets. In the following we briefly review the main definitions. To simplify the presentation, we consider in this paper only *safe* nets, where each place can be marked with at most one token. The definitions for the general case are given in [5].

Figure 1 shows a simple Petri game, which will serve as our running example in the following. If we ignore the distinction between gray and white places for the moment, then the net shown in the figure is a standard Petri net. A *Petri net* $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, In)$ consists of disjoint sets $\mathcal{P}$ of *places* and $\mathcal{T}$ of transitions, a *flow relation* $\mathcal{F} \subseteq (\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P})$, and an *initial marking* $In \subseteq \mathcal{P}$. We depict places as circles and transitions as rectangles. Note that the flow relation defines a bipartite graph, i.e., the flow relation connects places with transitions and transitions with places. Places and transitions are generically called *nodes*. A *finite* Petri net is a Petri net with finitely many nodes. For nodes x, y we write $x \mathcal{F} y$ for $(x, y) \in \mathcal{F}$.

The presence of a flow between two nodes models a causal dependency. The *preset* of a node x, denoted by ${}^\bullet x$, is the set $\{y \mid y \mathcal{F} x\}$. The *postset* of x, denoted by $x^\bullet$, is the set $\{y \mid x \mathcal{F} y\}$.

The behavior of a Petri net is defined in terms of its *markings*, which are subsets of the places. A transition t is *enabled* in a marking $M \subseteq \mathcal{P}$ if ${}^\bullet t \subseteq M$. A marking M' is *reachable* from a marking M *in one step*, denoted by $M \rightarrow M'$ if there is a transition t that is enabled in M and $M' = (M \setminus {}^\bullet t) \cup t^\bullet$. A sequence $M_1 M_2 M_3 \dots M_n$ such that $M_i \rightarrow M_{i+1}$ for all $i \in \{1, \dots, n-1\}$ is a *firing sequence* of $\mathcal{N}$. The set of *reachable markings* is defined as $\mathcal{R}(\mathcal{N}) = \{M' \mid In \rightarrow^* M'\}$ where $\rightarrow^*$ is the reflexive and transitive closure of $\rightarrow$.

In the example of Fig. 1, the initial marking $\{A, D\}$ (which is depicted by the black dots on places A and D) has four enabled transitions, namely s, t, v and w, which result in the successor markings $\{B, D\}, \{C, D\}, \{A, F\}$ and $\{A, G\}$, respectively. An example for a firing sequence is the infinite repetition of the sequence $\{A, D\}\{B, D\}\{E, D\}\{E, F\}\{A, D\}$. Note that there is no reachable marking that contains both B and C. Places B and C are in conflict: transition s, which adds a token to B, also removes the token from A, which is needed to enable t. We say that two nodes x and y are *in conflict*, denoted by $x \sharp y$, if there exists a place $p \in \mathcal{P}$, different from x and y, from which one can reach x and y via the transitive closure $\mathcal{F}^+$ of $\mathcal{F}$, exiting p by different transitions.

A *Petri game* $\mathcal{G} = (\mathcal{P}_S, \mathcal{P}_E, \mathcal{T}, \mathcal{F}, In, \mathcal{B})$ is a finite Petri net where the set of places has been partitioned into a subset $\mathcal{P}_S$ belonging to the *system* and a subset $\mathcal{P}_E$ belonging to the *environment*; additionally, the Petri game identifies a set $\mathcal{B} \subseteq 2^{\mathcal{P}}$ of *bad* markings (from the point of view of the system), which indicate a victory for the environment[1]. We call the Petri net $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, In)$ with $\mathcal{P} = \mathcal{P}_S \cup \mathcal{P}_E$ the *underlying* Petri net of $\mathcal{G}$.

[1] In [5], the bad markings are given as a set of bad places that must be avoided by the system.

We view each token as a player in a multiplayer game. Informally, the tokens on the environment places show the complete, unrestricted behavior, while the tokens on the system places restrict the behavior strategically by forbidding a subset of the transitions in the postset of the currently occupied place.

In the example game of Fig. 1, places belonging to the system are shown in gray, all other places belong to the environment. Let the set of bad markings consist of the markings $\{B, G\}$ and $\{C, F\}$, where the net has reached a deadlock. The example has two tokens, one that moves on environment places, one on system places. We will refer to these two tokens as the system player and the environment player, respectively. Note that, in general, there may be more than one token in a team, which means that these players have the same objective, but not necessarily the same knowledge about decisions that have been made by the players of the other team.

Initially, in place D, the system player must choose a subset of the transitions u, v, w and x. To avoid the bad markings, it is crucial that the system player stays in place D until it is clear whether the environment player has played transition s or t. If the system player proceeds with, for example, transition v, and the environment player turns out to be in place C, then the bad marking $\{C, F\}$ is reached. To win the game, the system player must forbid transitions v and w and thus stay in place D until a synchronization with the environment token in places B or C becomes possible. After the synchronization via u or x, the system player knows whether the environment player is in place E or H and can, correspondingly, enable v if the synchronization was via transition u or w if the synchronization was via transition x. Transition v thus leads to the marking $\{E, F\}$, transition w to the marking $\{G, H\}$ and the bad markings $\{B, G\}$ and $\{C, F\}$ are avoided.

The example illustrates that information about other players can be obtained through synchronization. The system player does *not* know whether the environment player went from place A via s to B or via t to C *until* the players synchronize via transitions u or x. The formalization of this idea is based on the notions of occurrence nets, branching processes, and unfoldings.

Informally, the unfolding of a Petri net is constructed by splitting any places with multiple incoming flows into separate copies that each only have a single incoming flow. Loops are unrolled into an infinite structure. Figure 2 shows the unfolding of the example game from Fig. 1. Note that every place in the unfolding has a unique causal history. Each place thus captures precisely the knowledge of the player when the token reaches the place in a play of the game. This correspondence between nodes and knowledge is exploited in the definition of strategies: a strategy fixes for each place in the unfolding the set of transitions that are not forbidden by any of the players. In Fig. 2, the strategy discussed above is depicted with thick lines. Note that, because there are multiple instances of place D, the system token in D can choose to forbid different transitions depending on whether transition u, transition x, or neither u or x has occurred in the present round of the game.

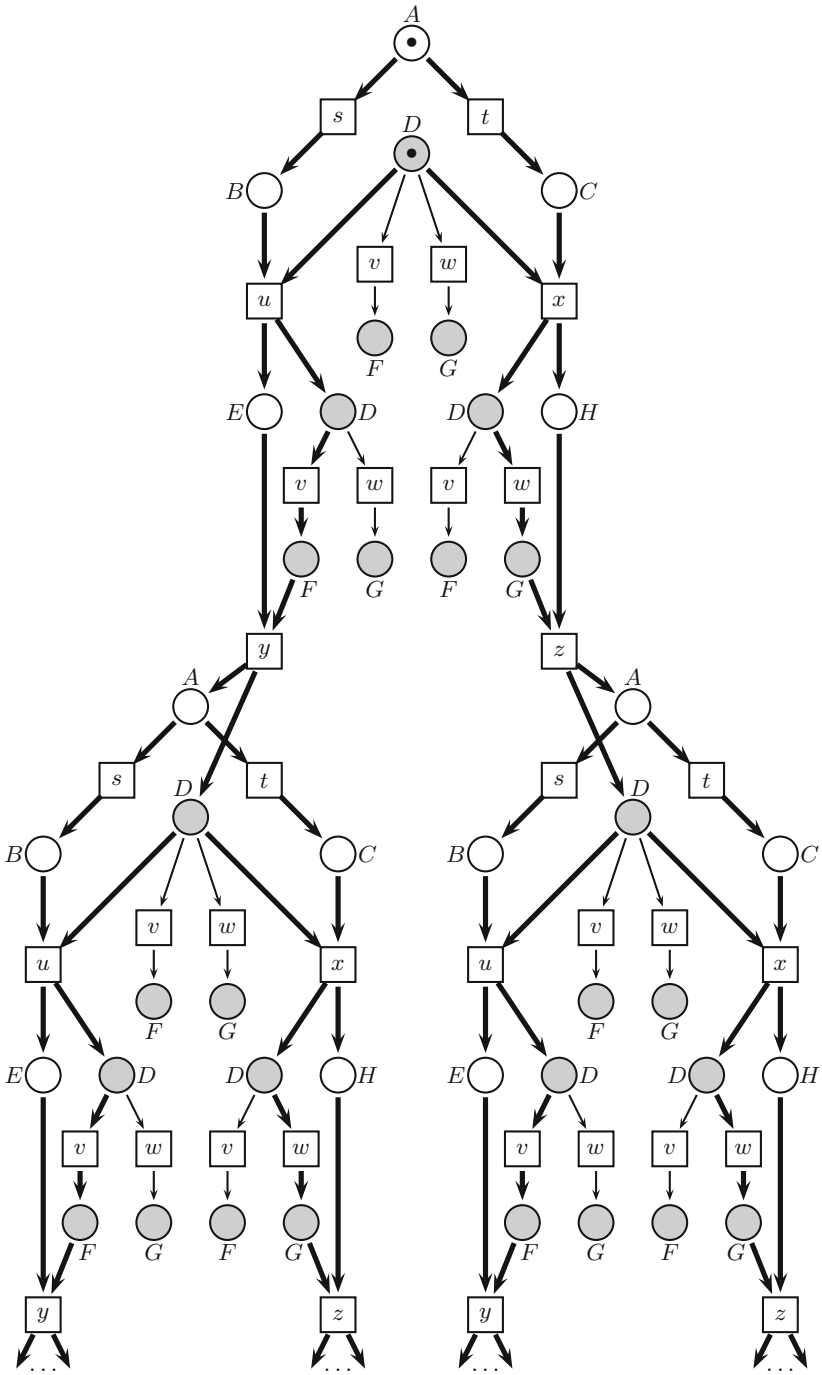

Fig. 2. Unfolding and winning strategy of the Petri game from Fig. 1. The winning strategy is shown with thick lines.

Formally, an *occurrence net* is a Petri net where (ON1) each place has at most one incoming transition; (ON2) the inverse flow relation $\mathcal{F}^{-1}$ is well-founded, i.e., starting from any node of $\mathcal{N}$ there does not exist an infinite path following the flow relation backwards; (ON3) no transition $t \in \mathcal{T}$ is in *self-conflict*, i.e., $t \sharp t$ does not hold for any transition t, and (ON4) the initial marking is the set of places without incoming transitions.

Two nodes x, y of an occurrence net are *causally related* if $x\,\mathcal{F}^*\,y$ or $y\,\mathcal{F}^*\,x$, where $\mathcal{F}^*$ denotes the reflexive and transitive closure of $\mathcal{F}$. They are *concurrent* if they are neither causally related nor in conflict. A *homomorphism* from a Petri net $\mathcal{N}_1$ to a Petri net $\mathcal{N}_2$ is a mapping $h : \mathcal{P}_1 \cup \mathcal{T}_1 \to \mathcal{P}_2 \cup \mathcal{T}_2$ that preserves the type of the elements, i.e., $h(\mathcal{P}_1) \subseteq \mathcal{P}_2$ and $h(\mathcal{T}_1) \subseteq \mathcal{T}_2$, and the pre- and postsets of the transitions, i.e., for all transitions $t \in \mathcal{T}_1 : h \downarrow {}^\bullet t$ is a bijection from ${}^\bullet t$ onto ${}^\bullet h(t)$ and $h \downarrow t^\bullet$ is a bijection from $t^\bullet$ onto $h(t)^\bullet$, where $h \downarrow D$ denotes the restriction of h to the domain D. If additionally the restriction $h \downarrow In_1$ is a bijection from In_1 onto In_2, then h is called *initial*.

A *branching process* of a net $\mathcal{N}$ is a pair $\beta = (\mathcal{N}^B, \lambda^B)$, where $\mathcal{N}^B$ is an occurrence net and λ^B is a homomorphism from $\mathcal{N}^B$ to $\mathcal{N}$ that is injective on transitions with the same preset, i.e., for all transitions t_1 and t_2 of the branching process, if ${}^\bullet t_1 = {}^\bullet t_2$ and $\lambda^B(t_1) = \lambda^B(t_2)$, then t_1 and t_2 must be the same transition. If λ^B is initial, β is called an *initial branching process*.

The *unfolding of a net* $\mathcal{N}$ is an initial branching process $\beta_U = (\mathcal{N}^U, \lambda^U)$ that is *complete* in the sense that every transition of the net is recorded in the unfolding, i.e., for every transition t and every set C of concurrent places, if $\lambda^U \downarrow C$ is a bijection from C onto ${}^\bullet t$, then there exists a transition $t^U \in \mathcal{T}^U$ such that ${}^\bullet t^U = C$ and $\lambda^U(t^U) = t$. The *unfolding of a game* $\mathcal{G}$ is the unfolding of the underlying net $\mathcal{N}$.

A branching process β_1 is a *subprocess* of a branching process β_2 if the identity on the nodes of β_1 is an initial homomorphism from β_1 to β_2. A *strategy* for the system players is a subprocess $\sigma = (\mathcal{N}^\sigma, \lambda^\sigma)$ of the unfolding $\beta_U = (\mathcal{N}^U, \lambda)$ of $\mathcal{N}$ subject to the following conditions: (S1) σ is deterministic in all system places, i.e., for all reachable markings $M \in \mathcal{R}(\mathcal{N}^\sigma)$ and all system places p, there is at most one transition $t \in \mathcal{T}^\sigma$ such that $p \in {}^\bullet t$ and ${}^\bullet t \subseteq M$; (S2) the strategy does not restrict local transitions of the environment, i.e., if, for a transition $t \in \mathcal{T}^U$, ${}^\bullet t \subseteq \mathcal{P}_E^\sigma$, then $t \in \mathcal{T}^\sigma$; and (S3) if an instance t of a transition is forbidden by σ there exists a place $p \in {}^\bullet t$ where σ uniformly forbids all instances t' of this transition.

A strategy σ for the system players is *winning* if the bad markings are unreachable in the strategy, i.e., $\mathcal{B} \cap \mathcal{R}(\mathcal{N}^\sigma) = \emptyset$. For example, the strategy shown in Fig. 2 is a winning strategy for the system player of the Petri game in Fig. 1. To avoid trivial solutions, we look for strategies σ that are *deadlock avoiding* in the sense that for all reachable markings $M \in \mathcal{R}(\mathcal{N}^\sigma)$, if there exists an enabled transition in the unfolding, i.e., $t \in \mathcal{T}^U$ with ${}^\bullet t \subseteq M$, then there exists an enabled transition in the strategy as well, i.e., $t \in \mathcal{T}^\sigma$ with ${}^\bullet t \subseteq M$. Note that we allow the strategy to produce a deadlock if the deadlock was already present in the game. In such situations we say that the game has *terminated*. If

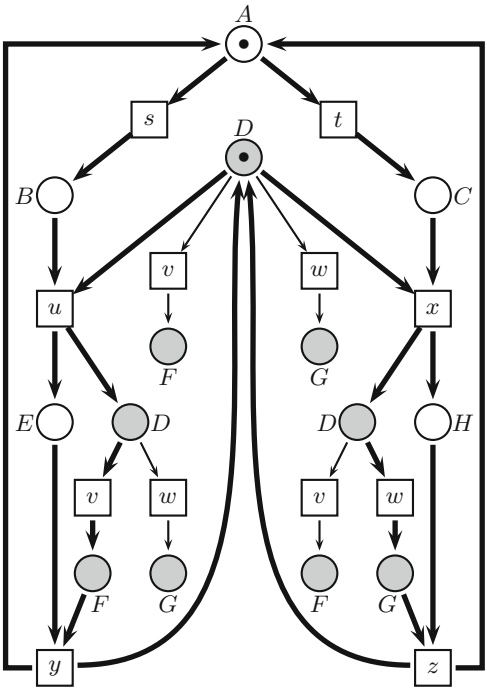

Fig. 3. Bounded unfolding und winning strategy of the Petri game from Fig. 1 for a bound b that allows for three instances of place D: $b(p) = 1$ for $p \in \{A, B, C, E, H\}$ and $b(p) = 3$ for $p \in \{D, F, G\}$. The b-bounded unfolding admits the winning strategy shown with thick lines.

termination is undesired (as in the example of Fig. 1), such markings must be explicitly included in the set of bad markings.

3 Bounded Strategies

Since strategies are subprocesses of the unfolding, they are in general infinite objects, even if the state space of an actual controller implementing the strategy turns out to be finite. A strategy $\sigma = (\mathcal{N}^\sigma, \lambda^\sigma)$ is *finitely generated* if there exists a finite net $\mathcal{N}^f$ and a homomorphism λ from $\mathcal{N}^\sigma$ to $\mathcal{N}^f$ such that $(\mathcal{N}^\sigma, \lambda)$ is an unfolding of $\mathcal{N}^f$. We say that σ is *finitely generated by* $\mathcal{N}^f$.

We search for finitely generated strategies by considering bounded unfoldings of the game. A *bound* $b : \mathcal{P} \to \mathbb{N}$ assigns to each place of the game a natural number. A pair $(\mathcal{N}^b, \lambda^b)$ consisting of a finite net $\mathcal{N}^b$ and a homomorphism λ^b from $\mathcal{N}^b$ to $\mathcal{N}$ is a *b-bounded unfolding* of the game $\mathcal{G}$ if there exists a homomorphism λ from the net $\mathcal{N}^U$ of the unfolding $(\mathcal{N}^U, \lambda^U)$ of $\mathcal{G}$ to $\mathcal{N}^b$ such that $\lambda^U(p) = \lambda^b(\lambda(p))$ for all nodes p of $\mathcal{N}^U$, and, furthermore, each place p of $\mathcal{G}$ occurs at most $b(p)$ times in $\mathcal{N}^b$, i.e., $|(\lambda^b)^{-1}(p)| \leq b(n)$ for every $p \in \mathcal{P}$. Figure 3

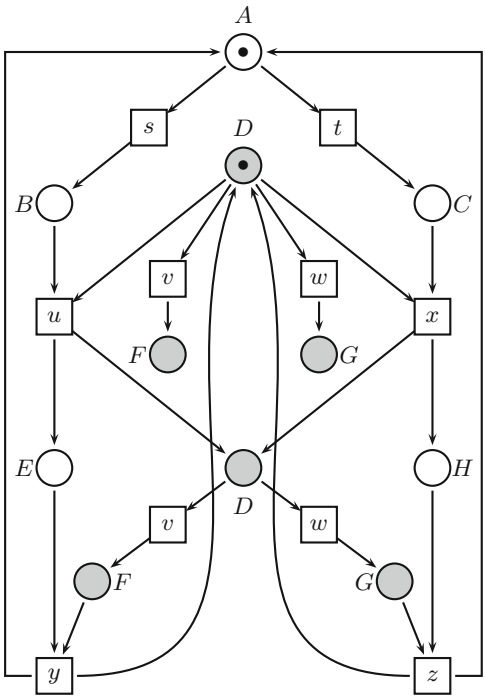

Fig. 4. Bounded unfolding of the Petri game from Fig. 1 for a bound b' that allows only two instances of place D: $b'(p) = 1$ for $p \in \{A, B, C, E, H\}$ and $b'(p) = 2$ for $p \in \{D, F, G\}$. The b'-bounded unfolding does not admit a winning strategy.

shows a b-bounded unfolding of the Petri game from Fig. 1 for the bound b with $b(p) = 1$ for $p \in \{A, B, C, E, H\}$ and $b(p) = 3$ for $p \in \{D, F, G\}$.

We find bounded strategies by restricting the flow of a bounded unfolding. A *b-bounded strategy* is a finite net $\mathcal{N}^f$ such that there is a b-bounded unfolding $\mathcal{N}^b$ of G with $\mathcal{P}^f = \mathcal{P}^b$, $\mathcal{T}^f = \mathcal{T}^b$, $In^f = In^b$ and $\mathcal{F}^f \subseteq \mathcal{F}^b$, and there is a strategy σ that is finitely generated by $\mathcal{N}^f$. We say that $\mathcal{N}^b$ *admits* the bounded strategy $\mathcal{N}^f$. The bounded strategy $\mathcal{N}^f$ is winning iff σ is winning.

In Fig. 3, a bounded strategy is depicted as part of the bounded unfolding. The thick lines indicate the flow that is preserved by the strategy. Note that the strategy from Fig. 2 is finitely generated by this bounded strategy. The bounded strategy is thus winning. Obviously, not every bounded unfolding admits a winning strategy. Consider, for example, the b'-bounded unfolding of our game in Fig. 4. The bound b' with $b'(p) = 1$ for $p \in \{A, B, C, E, H\}$ and $b'(p) = 2$ for $p \in \{D, F, G\}$ only allows two instances of D. The transitions u and x thus lead to the same instance of place D. As a result, the information whether the environment token is in place E or H is not available in the place reached by the two transitions. The strategy cannot forbid both outgoing transitions v and w, because this would result in a deadlock; however, no matter if the strategy

chooses to enable v or w, there is always a corresponding choice of s vs. t for the environment player that results in a bad marking.

4 Finding Bounded Strategies

We look for winnning strategies of a given game by considering bounded unfoldings for a sequence of increasing bounds. For each such bounded unfolding, we check whether it admits a winning strategy. In this section, we describe an efficient method that carries out this check.

Our method is based on an encoding into quantified boolean formula (QBF) satisfiability. Syntactically, the *quantified boolean formulas* over a set of boolean variables $\mathcal{V}$ are described by the following grammar:

$$\phi ::= x \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \phi \Rightarrow \phi \mid \phi \Leftrightarrow \phi \mid \mathit{true} \mid \mathit{false}$$
$$\Phi ::= \phi \mid \Phi \wedge \Phi \mid \Phi \vee \Phi \mid \exists x.\ \Phi \mid \forall x.\ \Phi$$

where x denotes boolean variables from $\mathcal{V}$ and $\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$ are the usual boolean connectives.

QBF satisfiability can be reduced to boolean satisfiability (SAT) by replacing every existentially quantified formula $\exists x.\phi$ by a disjunction $\phi[x/\mathit{true}] \vee \phi[x/\mathit{false}]$ where the quantified variable is replaced by *true* and *false*, respectively, in the two disjuncts, and by replacing every universally quantified formula $\forall x.\phi$ by a conjunction $\phi[x/\mathit{true}] \wedge \phi[x/\mathit{false}]$. QBF solving is, however, more difficult than SAT, both in theory (PSPACE-complete vs. NP-complete), and in practice [17]. Nevertheless, QBF is increasingly being used for practical applications and several powerful QBF solvers are available (cf. [14]).

Let $\mathcal{N}^b$ be a bounded unfolding. We encode the existence of a winning strategy that is admitted by $\mathcal{N}^b$ as a formula

$$\Phi_n = \exists V_S.\ \forall V_{T,n}.\ \phi_n,$$

where V_S is a set of boolean variables that encode which transitions are chosen by the strategy, $V_{T,n}$ is a set of boolean variables that encode a sequence of transitions, and ϕ_n is a boolean formula that expresses that if the choices of V_S and $V_{T,n}$ result in a firing sequence in $\mathcal{N}^b$, then the sequence is won by the system players. The index $n \in \mathbb{N}$ is a natural number that indicates the length of the firing sequence to be considered. We consider firing sequences that end in a repeated marking. Since there are only $2^{|\mathcal{P}^b|}$ many different markings, such a repetition must occur after at most exponentially many steps, and it suffices to set $n = 2^{|\mathcal{P}^b|} + 1$. However, we leave n as a parameter, which allows us to restrict the encoding to shorter firing sequences.

Our encoding of the existence of a winning strategy as a quantified boolean formula resembles the reduction of the bounded *model checking* problem to SAT [11]. The main difference is that in model checking, one is interested in finding a *single* firing sequence that leads from the initial marking to a bad marking, while in synthesis one must ensure that *all* firing sequences are correct.

We accomplish this by quantifying universally over the variables in $V_{T,n}$, which select the sequence of transitions, and by requiring that every firing sequence has a loop (unless it ends in a deadlock).

Proposition 1. *Let $\mathcal{N}^b$ be a bounded unfolding. There is a family of quantified boolean formulas Φ_n such that Φ_n is satisfiable for some $n \leq 2^{|\mathcal{P}^b|} + 1$ iff N^b admits a deadlock-avoiding winning strategy for the system players.*

Proof. We define $\Phi_n = \exists V_S.\, \forall V_{T,n}.\, \phi_n$ as follows. The set $V_S = \{(p, \lambda^b(t)) \mid p \in \mathcal{P}^b_S, t \in \mathcal{T}^b, p \in {}^\bullet t\}$ consists of boolean variables encoding the system strategy. There is a variable for each pair of a system place and a transition where the preset of some instance of the transition contains the system place[2]. The set $V_{T,n} = \{(p,i) \mid p \in \mathcal{P}^b, i \in \{1, \dots n\}\}$ contains one boolean variable for each place and index position between 1 and n, representing a sequence of markings of length n. The formula ϕ_n expresses that every sequence up to length n is winning for the system players. If the sequence reaches the full length n (i.e., there is no previous deadlock) it must be a loop:

$$\phi_n = \left(\bigwedge_{i \in \{1,\dots,n\}} \mathit{sequence}_i \Rightarrow \mathit{winning}_i\right) \wedge (\mathit{sequence}_n \Rightarrow \mathit{loop}_n)$$

Condition $\mathit{sequence}_n$ describes that the sequence of markings encoded by $V_{T,n}$ is indeed a firing sequence:

$$\mathit{sequence}_i = \mathit{initial} \wedge \neg \mathit{deadlock}_1 \wedge \mathit{flow}_1 \wedge \neg \mathit{deadlock}_2 \wedge \mathit{flow}_2 \wedge \dots \wedge \neg \mathit{deadlock}_{i-1} \wedge \mathit{flow}_{i-1}$$

where $\mathit{initial}$, $\mathit{deadlock}_i$, and flow_i encode that the first marking is the initial marking, the occurrence of deadlock in the ith marking, and the satisfaction of the flow relation from the ith to the $(i+1)$st marking, respectively:

$$\begin{aligned}
\mathit{initial} &= \left(\bigwedge_{p \in In^b} (p,1)\right) \wedge \left(\bigwedge_{p \in \mathcal{P}^b \setminus In^b} \neg(p,1)\right) \\
\mathit{deadlock}_i &= \bigwedge_{t \in \mathcal{T}^b} \left(\bigvee_{p \in {}^\bullet t} \neg(p,i)\right) \vee \left(\bigvee_{p \in {}^\bullet t \cap \mathcal{P}^b_S} \neg(p, \lambda^b(t))\right) \\
\mathit{flow}_i &= \bigvee_{t \in \mathcal{T}^b} \left(\bigwedge_{p \in {}^\bullet t}(p,i)\right) \wedge \left(\bigwedge_{p \in {}^\bullet t \cap \mathcal{P}^b_S}(p, \lambda^b(t))\right) \wedge \left(\bigwedge_{p \in t^\bullet}(p, i+1)\right) \\
&\quad \wedge \left(\bigwedge_{p \in \mathcal{P}^b \setminus ({}^\bullet t \cup t^\bullet)}(p,i) \Leftrightarrow (p, i+1)\right) \wedge \left(\bigwedge_{p \in {}^\bullet t \setminus t^\bullet} \neg(p, i+1)\right)
\end{aligned}$$

Condition $\mathit{winning}_i$ ensures that there are no bad markings, that all deadlocks are terminating markings (i.e., the deadlock was already present in the net), and that the strategy is deterministic in all markings (as required by condition (S1)):

[2] The variable refers to $\lambda^b(t) \in \mathcal{T}$ instead of $t \in \mathcal{T}^b$ because of condition (S3) on strategies, which requires that all instances of a transition must be uniformly forbidden.

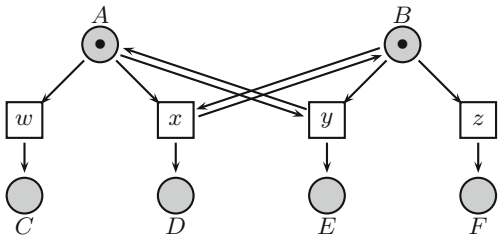

Fig. 5. Trade-off between the memory requirements of two players. The Petri game has a single bad marking $\{C, F\}$. Both the b-bounded unfolding with $b(A) = 1$ and $b(B) = 2$ and the b'-bounded unfolding with $b'(A) = 2$ and $b'(B) = 1$ admit a winning strategy, but the b''-bounded unfolding with $b''(A) = 1$ and $b''(B) = 1$ does not admit a winning strategy.

$$
\begin{aligned}
&\mathit{winning}_i = \mathit{nobadmarking}_i \wedge \mathit{deadlocksterm}_i \wedge \mathit{deterministic}_i\\
&\mathit{nobadmarking}_i = \textstyle\bigwedge_{M \in \mathcal{B}} \left(\bigvee_{p \in M} \neg(p,i) \right) \vee \left(\bigvee_{p \in \mathcal{P}^b \setminus M} (p,i) \right)\\
&\mathit{deadlocksterm}_i = \mathit{deadlock}_i \Rightarrow \mathit{terminating}_i\\
&\mathit{terminating}_i = \textstyle\bigwedge_{t \in \mathcal{T}^b} \bigvee_{p \in {}^\bullet t} \neg(p,i)\\
&\mathit{deterministic}_i = \textstyle\bigwedge_{t_1,t_2 \in \mathcal{T}^b, t_1 \neq t_2, {}^\bullet t_1 \cap {}^\bullet t_2 \cap \mathcal{P}^b_S \neq \emptyset}\\
&\qquad \left(\bigvee_{p \in {}^\bullet t_1 \cup {}^\bullet t_2} \neg(p,i) \right) \vee \left(\bigvee_{p \in ({}^\bullet t_1 \cup {}^\bullet t_2) \cap \mathcal{P}^b_S} \neg(p, \lambda^b(t)) \right)
\end{aligned}
$$

Finally, condition loop_n ensures that the nth marking is a repetition of an earlier marking:

$$\mathit{loop}_n = \textstyle\bigvee_{i \in \{1,\ldots n-1\}} \left(\bigwedge_{p \in \mathcal{P}^b} (p,i) \Leftrightarrow (p,n) \right)$$

□

For each choice of n, the formula Φ_n can be translated into conjunctive normal form by tools like the Boolean circuit tool package (BCpackage) [12] and then solved by a standard QBF solver like DepQBF [14].

5 Trade-Offs

An interesting type of analysis made possible by the bounded synthesis approach is to trade different bounds against each other. The Petri game in Fig. 5 illustrates a trade-off between the memory needed in different processes of a system. The bad marking of the game is the combination of the places C of the first player and F of the second player. The bad marking is reached if the two players choose their local transitions w and z. To avoid the bad marking, one of the players must *wait* for the other player before taking the local transition. For example, the token in A could initially enable transition y, and, once the token in B has moved via y to E, safely take w to C. This winning strategy requires two instances of A and only a single instance of B. Symmetrically, the token in B could first enable x and then z, provided that two instances of B are available.

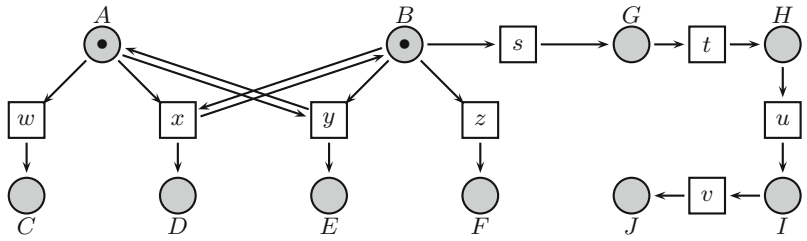

Fig. 6. Trade-off between memory and proof complexity. The Petri game has a single bad marking $\{C, F\}$. The b-bounded unfolding with $b(A) = 1$ and $b(B) = 2$, the b'-bounded unfolding with $b'(A) = 2$ and $b'(B) = 1$, and also the b''-bounded unfolding with $b''(A) = 1$ and $b''(B) = 1$ admit a winning strategy. However, the synthesis of the winning strategy in the b''-bounded unfolding requires a firing sequence of length six, while the firing sequences of the winning strategies in the b-bounded and b'-bounded unfoldings only have length three.

A winning strategy thus requires either two instances of A or two instances of B. Both the b-bounded unfolding with $b(A) = 1$ and $b(B) = 2$ and the b'-bounded unfolding with $b'(A) = 2$ and $b'(B) = 1$ admit a winning strategy, indicating that the memories allocated to the two players can be traded against each other, while the b''-bounded unfolding with $b''(A) = 1$ and $b''(B) = 1$ does not admit a winning strategy.

Since the encoding of the bounded synthesis problem in Section 4 is parametric in the length of the firing sequences, i.e., in the complexity of the *correctness proof*, another interesting trade-off to be analyzed is between memory and proof complexity. This is illustrated by the Petri game in Fig. 6, where, similarly to the previous example, the goal is to avoid the bad marking in places C and F. This time, however, the players can avoid the synchronization, if the token on place A takes the local transition to C and the token on B moves along the long chain towards the right. While this solution can be implemented within bound $b''(A) = b''(B) = 1$, the length of the firing sequence corresponds to the length of the chain plus the firing of w, resulting in length six, while the b-bounded unfolding with $b(A) = 1, b(B) = 2$ (and, analogously, the b'-bounded unfolding with $b'(A) = 2, b'(B) = 1$) admits a winning strategy where the firing sequence has only three markings.

6 Conclusions

We have presented a bounded synthesis method for Petri games. Similarly to the bounded synthesis approach for the synthesis of distributed systems from temporal logic [7,20], our approach limits the size of the solution and therefore finds small solutions fast. Petri games appear to be particularly well-suited for bounded synthesis because the net typically provides more structure than a logical specification. Because specific bounds can be set for individual places

(and, hence, for individual players), bounded synthesis can also be used to analyze trade-offs in the memory needed for different the players, and even trade-offs between memory and proof complexity.

The bounded synthesis approach complements the BDD-based symbolic decision procedure for Petri games implemented in the ADAM tool [4]. In model checking, SAT-based bounded methods often dramatically outperform BDD-based symbolic methods [3]. It will be interesting to see if the situation is similar for the synthesis problem.

It is important to note, however, that the two methods are not directly comparable. While the symbolic decision procedure is limited to games with a single environment token, the bounded approach is universally applicable. On the other hand, the symbolic decision procedure can prove the absence of a strategy (of arbitrary size), while the bounded approach is currently limited to proving the existence of a strategy. Combining the two approaches is an interesting topic for future work.

Acknowledgments. I am deeply grateful to Ernst-Rüdiger Olderog for our productive and most enjoyable collaboration on Petri games. Thanks are also due to the anonymous referees for their helpful comments on this paper.

References

1. Büchi, J.R., Landweber, L.H.: Solving sequential conditions by finite-state strategies. Transactions of the American Mathematical Society **138** (1969)
2. Church, A.: Applications of recursive arithmetic to the problem of circuit synthesis. In: Summaries of the Summer Institute of Symbolic Logic, vol. 1, pp. 3–50. Cornell Univ., Ithaca (1957)
3. Copty, F., Fix, L., Fraer, R., Giunchiglia, E., Kamhi, G., Tacchella, A., Vardi, M.Y.: Benefits of bounded model checking at an industrial setting. In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, p. 436. Springer, Heidelberg (2001)
4. 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, Heidelberg (2015)
5. Finkbeiner, B., Olderog, E.: Petri games: synthesis of distributed systems with causal memory. In: Peron, A., Piazza, C. (eds.) Proc. Fifth Intern. Symp. on Games, Automata, Logics and Formal Verification (GandALF). EPTCS, vol. 161, pp. 217–230 (2014). http://dx.doi.org/10.4204/EPTCS.161.19
6. Finkbeiner, B., Schewe, S.: Uniform distributed synthesis. In: Proc. LICS, pp. 321–330. IEEE Computer Society Press (2005)
7. Finkbeiner, B., Schewe, S.: Bounded synthesis. International Journal on Software Tools for Technology Transfer **15**(5–6), 519–539 (2013). http://dx.doi.org/10.1007/s10009-012-0228-z
8. Gastin, P., Lerman, B., Zeitoun, M.: Distributed games with causal memory are decidable for series-parallel systems. In: Lodaya, K., Mahajan, M. (eds.) FSTTCS 2004. LNCS, vol. 3328, pp. 275–286. Springer, Heidelberg (2004)

9. 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, Part II. LNCS, vol. 7966, pp. 275–286. Springer, Heidelberg (2013)
10. Green, C.: Application of theorem proving to problem solving. In: Proceedings of the 1st International Joint Conference on Artificial Intelligence. IJCAI 1969, pp. 219–239. Morgan Kaufmann Publishers Inc., San Francisco (1969). http://dl.acm.org/citation.cfm?id=1624562.1624585
11. Heljanko, K.: Bounded reachability checking with process semantics. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR 2001. LNCS, vol. 2154, pp. 218–232. Springer, Heidelberg (2001)
12. Junttila, T.A., Niemelä, I.: Towards an efficient tableau method for boolean circuit satisfiability checking. In: Palamidessi, C., Moniz Pereira, L., Lloyd, J.W., Dahl, V., Furbach, U., Kerber, M., Lau, K.-K., Sagiv, Y., Stuckey, P.J. (eds.) CL 2000. LNCS (LNAI), vol. 1861, pp. 553–567. Springer, Heidelberg (2000)
13. Kupferman, O., Vardi, M.Y.: Synthesizing distributed systems. In: Proc. LICS, pp. 389–398. IEEE Computer Society Press (2001)
14. Lonsing, F., Biere, A.: DepQBF: A dependency-aware QBF solver. JSAT **7**(2–3), 71–76 (2010)
15. 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)
16. Madhusudan, P., Thiagarajan, P.S.: Distributed controller synthesis for local specifications. In: Orejas, F., Spirakis, P.G., van Leeuwen, J. (eds.) ICALP 2001. LNCS, vol. 2076, p. 396. Springer, Heidelberg (2001)
17. Mangassarian, H.: QBF-based formal verification: Experience and perspectives. JSAT 133–191
18. Pnueli, A., Rosner, R.: Distributed reactive systems are hard to synthesize. In: Proc. FOCS 1990, pp. 746–757 (1990)
19. Rabin, M.O.: Automata on Infinite Objects and Church's Problem, Regional Conference Series in Mathematics, vol. 13. Amer. Math. Soc. (1972)
20. Schewe, S., Finkbeiner, B.: Bounded synthesis. In: Namjoshi, K.S., Yoneda, T., Higashino, T., Okamura, Y. (eds.) ATVA 2007. LNCS, vol. 4762, pp. 474–488. Springer, Heidelberg (2007)
21. Zielonka, W.: Asynchronous automata. In: Rozenberg, G., Diekert, V. (eds.) Book of Traces, pp. 205–248. World Scientific (1995)