Efficient Trace Encodings of Bounded Synthesis for Asynchronous Distributed Systems*

Jesko Hecking-Harbusch and Niklas O. Metzger

Saarland University, Saarbrücken, Germany

Abstract. The manual implementation of distributed systems is an error-prone task because of the asynchronous interplay of components and the environment. Bounded synthesis automatically generates an implementation for the specification of the distributed system if one exists. So far, bounded synthesis for distributed systems does not utilize their asynchronous nature. Instead, concurrent behavior of components is encoded by all interleavings and only then checked against the specification. We close this gap by identifying true concurrency in synthesis of asynchronous distributed systems represented as Petri games. This defines when several interleavings can be subsumed by one true concurrent trace. Thereby, fewer and shorter verification problems have to be solved in each iteration of the bounded synthesis algorithm. For Petri games, experimental results show that our implementation using true concurrency outperforms the implementation based on checking all interleavings.

1 Introduction

One ambitious goal in computer science is the automatic generation of programs. For a given specification, a *synthesis algorithm* either generates a program satisfying the specification or determines that no such program exists. Nowadays, most synthesis tools deploy a game-theoretic view on the problem [23,4,2,6]. The synthesis of *distributed systems* [30] can be represented by a team of system players and a team of environment players playing against each other. Each system player acts on individual information and requires a local strategy, which in combination with the strategies of the other system players satisfies an objective against the decisions of the team of environment players. The environment players can cooperate to prevent the satisfaction of the objective by the system players. In the *synchronous* setting where all players progress at the same rate, the synthesis problem for distributed systems is undecidable [31,12].

Petri games represent *asynchronous* behavior in the synthesis of distributed systems where processes can progress at individual rates between synchronizations. Furthermore, the players of the team of system players have *causal memory*, i.e., a system player can base decisions on its local past and the local past of

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

all other players up to their last synchronization. The synthesis problem for Petri games is decidable if for a maximum of one for the number of system players or the number of environment players [11,10]. If the restrictions on the team size cannot be met, *bounded synthesis* [13] is applied to incrementally increase the memory of possible system strategies until a winning one is found.

Each iteration of the bounded synthesis algorithm for Petri games [7] checks the existence of a winning system strategy with bounded memory by simulating the resulting Petri game. This simulation is represented as all *interleavings* of fired transitions allowed by possible system strategies. For two independent decisions, it makes no difference whether one decision or the other is scheduled first. It suffices to only check one scheduling where both decisions happen *true concurrently*. The true concurrent scheduling not only considers fewer schedulings but also shorter ones. Furthermore, the true concurrent scheduling enables us to refine the detection of loops in bounded synthesis. This results in a considerable speed-up of the verification part of bounded synthesis for Petri games.

To identify true concurrency, we introduce *environment strategies* for Petri games which explicitly represent the decisions of environment players. Environment strategies restrict a given system strategy and try to reach markings which prove the system strategy to *not* be winning. We present how the explicit environment decisions of environment strategies allow the firing of maximal sets of true concurrent transitions while preserving the applicability to bounded synthesis. This requires some *stalling* options for the environment. For bounded synthesis, we encode the assumptions on system and environment strategies as well as the winning objective of Petri games as *quantified Boolean formula* (QBF). We compare the implementations of the *sequential encoding* based on all interleavings and our new *true concurrent encoding* on an extended set of benchmarks[1]. Our experimental results show that the true concurrent encoding outperforms the sequential encoding by a considerable margin.

The key contributions of this paper are the following:

- We develop the theoretical foundation of *true concurrency* of components in synthesis for asynchronous distributed systems by representing environment decisions explicitly in *environment strategies* of Petri games.
- We prove that environment strategies *preserve existence of winning system strategies* and encode them as QBFs for bounded synthesis for Petri games.
- We *implement* the true concurrent encoding and show considerable improvements against the sequential encoding on an extended benchmark set.

The paper is structured as follows: In Section 2, we give an intuitive introduction to Petri games and the benefits of true concurrent scheduling for bounded synthesis for Petri games. Section 3 recalls the required background on Petri nets, Petri games, and bounded synthesis. In Section 4, we introduce environment strategies and prove that they preserve the existence of winning strategies. Section 5 gives the true concurrent encoding formally as QBF. Section 6 surveys experimental results for the implementation of the true concurrent encoding.

[1] The sequential and the concurrent encoding can be tested online as part of the ADAM toolkit [9]: https://www.react.uni-saarland.de/tools/online/ADAM/

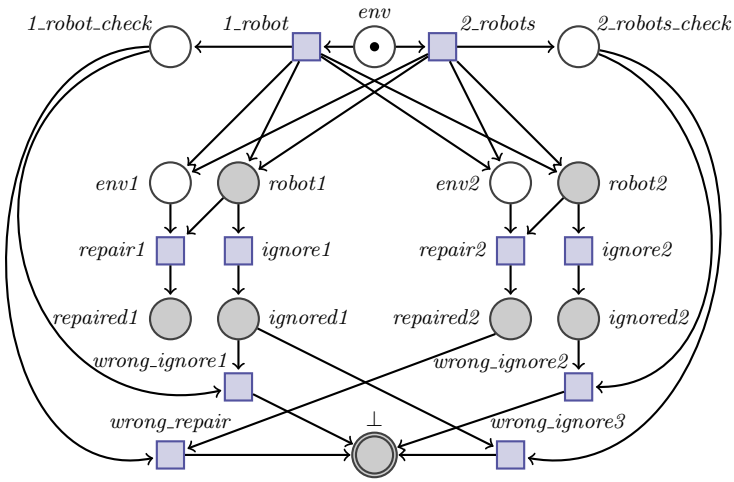

Fig. 1: This Petri game specifies a production line where two robots can repair a product. The product either requires repair by only one or by both robots.

2 Example of Bounded Synthesis for Petri Games

Figure 1 illustrates how Petri games represent the synthesis problem of asynchronous distributed systems and how true concurrency simplifies bounded synthesis for Petri games. This Petri game specifies a production line for repairing a product. The different possible requirements for repair are modeled as choices of the environment. The product can either require repair by a single robot or by both robots concurrently. These robots are represented by system players and have to collectively meet the requirement of the product.

Petri games are based on an underlying Petri net and distribute the places into two disjoint sets for the team of environment players and for the team of system players. White places belong to the environment and represent the product and its requirements for repair. Gray places belong to the system and represent the robots of the production line. The players are represented as tokens and their team is determined by the type of the place they are residing in. Initially, there is one token in the place *env* representing an environment player. Transitions define the flow of tokens through the Petri game as in Petri nets. When all places before a transition contain a token, then this transition is enabled. Firing an enabled transition consumes the tokens in all places before the transition and produces tokens in all places after it. The firing of enabled transition *1_robot* results in a consumption of the token in *env* and the production of tokens in places *1_robot_check*, *env1*, *robot1*, *env2*, and *robot2*. By this transition, both robots are started and it is required that only the first one repairs a part of the product.

The winning objective of the game is represented by the bad place $\bot$. The team of system players has to avoid reaching this place for all choices of the environment. Based on its causal past, a system player can decide which outgoing

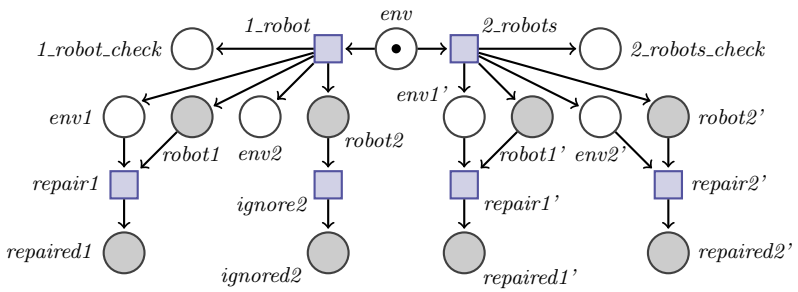

Fig. 2: A winning system strategy is presented for the Petri game from Fig. 1, which specifies a production line with two robots. The system players make different decisions depending on the choice of the environment. Transitions which cannot be enabled and unreachable places are removed.

transitions to fire. For example, the system place *robot2* can either be reached via transition *1_robot* or via *2_robots* and then the player can decide in both cases independently between transitions *repair2* and *ignore2*. Deciding independently is necessary because if the environment has chosen *1_robot*, no repair by the second robot is allowed whereas if the environment has chosen *2_robots*, repair by the second robot is required. The winning system strategy is presented in Fig. 2 where primed places and transitions result from different causal pasts. The outgoing transitions *ignore2* of place *robot2* and *repair2'* of *robot2'* represent the necessary different decisions of the system. Notice that the bad place is not reachable based on the decisions in the winning system strategy.

Bounded synthesis for Petri games uses quantified Boolean formulas (QBFs) to decide the existence of a winning system strategy for a given memory bound. The decisions at system places are represented explicitly as existentially quantified variables which are tested to be avoiding bad places for subsequent distributions of tokens until the game either terminates or reaches a loop. The memory bound implies the length of these sequences. The sequential encoding tests all possible interleavings of transitions, e.g., in our example, first the environment makes a decision between *1_robot* and *2_robots* and then two interleavings are tested depending on the ordering of the decisions of both system players. Our new concurrent flow semantics identifies such situations and replaces them with one true concurrent step for the decisions of both robots. Thereby, we reduce the number of considered traces from four interleavings of length three to two true concurrent traces of length two to verify the winning system strategy of Fig. 2.

3 Background

We introduce the necessary background on Petri nets [32], Petri games [11], and the sequential encoding of bounded synthesis for Petri games [7]. Notice that we limit ourselves to 1-*bounded (safe)* Petri nets for simpler notation.

3.1 Petri Nets

A (1-bounded) *Petri net* $\mathcal{N} = (\mathscr{P}, \mathscr{T}, \mathscr{F}, In)$ consists of a set of *places* $\mathscr{P}$, a set of *transitions* $\mathscr{T}$, a *flow relation* $\mathscr{F} \subseteq (\mathscr{P} \times \mathscr{T}) \cup (\mathscr{T} \times \mathscr{P})$, and an *initial marking* $In \subseteq \mathscr{P}$. The flow relation defines the *arcs* from places to transitions $(\mathscr{P} \times \mathscr{T})$ and from transitions to places $(\mathscr{T} \times \mathscr{P})$. The state of a Petri game is represented by a *marking* $M \subseteq \mathscr{P}$ which positions one *token* each in all places $p \in M$. The elements of $\mathscr{P} \cup \mathscr{T}$ are considered as *nodes*. We define the *preset* (and *postset*) of a node x from Petri net $\mathcal{N}$ as $pre^{\mathcal{N}}(x) = \{y \in \mathscr{P} \cup \mathscr{T} \mid (y, x) \in \mathscr{F}\}$ (and $post^{\mathcal{N}}(x) = \{y \in \mathscr{P} \cup \mathscr{T} \mid (x, y) \in \mathscr{F}\}$). The preset and postset of transitions are non-empty and finite. We use decorated names like $\mathcal{N}^b$ to also decorate the net's components. We abbreviate $pre^{\mathcal{N}^b}(x)$ and $post^{\mathcal{N}^b}(x)$ by $pre^b(x)$ and $post^b(x)$. A transition t is *enabled* at a marking M if $pre^{\mathcal{N}}(t) \subseteq M$ holds (denoted by $M[t\rangle$). An enabled transition t can be *fired* from a marking M resulting in the successor marking $M' = (M \setminus pre^{\mathcal{N}}(t)) \cup post^{\mathcal{N}}(t)$ (denoted by $M[t\rangle M'$). We define the set of *reachable markings* of a Petri net $\mathscr{R}(\mathcal{N}) = \{M \subseteq \mathscr{P} \mid \exists t_1, ..., t_n \in \mathscr{T} : \exists M_1, ..., M_n \subseteq \mathscr{P} : In[t_1\rangle M_1 ... [t_n\rangle M_n = M\}$. Two nodes x, y are *in conflict* (denoted by $x \sharp y$) if there exists a place $p \in \mathscr{P} \setminus \{x, y\}$ from which x and y can be reached, exiting p by different transitions.

3.2 (Bounded) Unfoldings and Subprocesses

The *unfolding* $\beta_U = (\mathcal{N}^U, \lambda^U)$ of a Petri net $\mathcal{N}$ explicitly represents the *causal pasts* of all places by eliminating all joins of places in the Petri net and separating these places into appropriate copies. Therefore, a loop in a Petri net results in an infinite unfolding. The homomorphism $\lambda^U : \mathscr{P}^U \cup \mathscr{T}^U \to \mathscr{P} \cup \mathscr{T}$ gives for nodes in the unfolding the corresponding original nodes. For bounded synthesis, we consider *bounded unfoldings* $\beta_U^b = (\mathcal{N}^b, \lambda^b)$, where the memory bound b defines how many causal pasts per place can be represented as separate copies. Thereby, loops are only finitely often unfolded. A net-theoretic *subprocess* of a Petri net or an unfolding (denoted by $\sqsubseteq$) is defined by removing a set of transitions and all following places and transitions that cannot be reached anymore.

3.3 Petri Games

A *Petri game* $\mathcal{G} = (\mathscr{P}_S, \mathscr{P}_E, \mathscr{T}, \mathscr{F}, In, \mathscr{B})$ [11] with $\mathscr{B} \subseteq \mathscr{P}_S \cup \mathscr{P}_E$ has an underlying Petri net $\mathcal{N} = (\mathscr{P}, \mathscr{T}, \mathscr{F}, In)$ with $\mathscr{P} = \mathscr{P}_S \uplus \mathscr{P}_E$. The sets $\mathscr{P}_S$, $\mathscr{P}_E$, and $\mathscr{B}$ define the *system places*, the *environment places*, and the *bad places*. Unfoldings translate from Petri nets to Petri games by keeping the classification of places as system, environment, and bad places. A *system strategy* for $\mathcal{G}$ is a subprocess $\sigma = (\mathcal{N}^\sigma, \lambda^\sigma)$ of the unfolding $\beta_U = (\mathcal{N}^U, \lambda^U)$ of $\mathcal{G}$ where system places can remove outgoing transitions such that the following requirements hold:

(S1) *Determinism*:
$\forall M \in \mathscr{R}(\mathcal{N}^\sigma) : \forall p \in M \cap \mathscr{P}_S^\sigma : \exists^{\leq 1} t \in \mathscr{T}^\sigma : p \in pre^\sigma(t) \wedge pre^\sigma(t) \subseteq M$

(S2) *System refusal*: $\forall t \in \mathscr{T}^U : t \notin \mathscr{T}^\sigma \wedge pre^\sigma(t) \subseteq \mathscr{P}^\sigma \implies (\exists p \in pre^\sigma(t) \cap \mathscr{P}_S^\sigma :$
$\forall t' \in post^U(p) : \lambda^U(t) = \lambda^U(t') \implies t' \notin \mathscr{T}^\sigma)$

(S3) *Deadlock-avoidance*:
$\forall M \in \mathscr{R}(\mathcal{N}^\sigma) : \exists t_U \in \mathscr{T}^U : pre^U(t_U) \subseteq M \implies \exists t_\sigma \in \mathscr{T}^\sigma : pre^\sigma(t_\sigma) \subseteq M$

Determinism requires each system player to have at most one transition enabled for all reachable markings. *System refusal* requires that the removal of a transition from the unfolding is based on a system place deleting all outgoing copies of that transition. This enforces that system players base their decisions only on their causal past. *Deadlock-avoidance* requires the system strategy to enable at least one transition for each reachable marking as long as one transition is enabled in the unfolding. A system strategy is *winning* for the winning condition *safety* if no bad place can be reached in the system strategy, i.e., $\forall M \in \mathscr{R}(\mathcal{N}^\sigma) : \lambda^\sigma[M] \cap \mathscr{B} = \emptyset$. The synthesis problem for Petri games with safety as winning objective is EXPTIME-complete if we limit the number of system players or the number of environment players to one [11,10].

3.4 Sequential Encoding of Bounded Synthesis for Petri Games

The bounded synthesis algorithm [7] takes a Petri game and increases the memory bound b until a winning system strategy is found (or runs forever). The finite bounded unfolding $\beta_U^b = (\mathcal{N}^b, \lambda^b)$ is used to encode the existence of a winning system strategy (as variables $\boldsymbol{\mathcal{S}}^b$) for all sequences of markings (as variables $\boldsymbol{\mathcal{M}}_n$) up to the maximal simulation length $n \leq 2^{|\mathscr{P}^b|}+1$ as QBF. In the encoding, concurrent transitions are represented by all possible interleavings as between two markings only a single transition is fired. For readability, we abbreviate $pre^{\mathcal{N}^b}(x)$ by ${}^\bullet x$ and $post^{\mathcal{N}^b}(x)$ by $x^\bullet$. The QBF has the form $\exists \boldsymbol{\mathcal{S}}^b : \forall \boldsymbol{\mathcal{M}}_n : \phi_n$ where $\boldsymbol{\mathcal{S}}^b \stackrel{def.}{=} \{(p, \lambda^b(t)) \mid p \in \mathscr{P}_S^b \wedge t \in p^\bullet\}$ and $\boldsymbol{\mathcal{M}}_n \stackrel{def.}{=} \{(p,i) \mid p \in \mathscr{P}^b \wedge 1 \leq i \leq n\}$.

The system strategy $\boldsymbol{\mathcal{S}}^b$ consists of Boolean variables representing the system's choice for each pair of system place in the bounded unfolding and outgoing transition of the corresponding system place in the original game. This encoding ensures that each system strategy satisfies *system refusal* (S2) because neither pure environment transitions can be disabled nor can transitions be differentiated due to the bounded unfolding. The marking sequence $\boldsymbol{\mathcal{M}}_n$ contains Boolean variables for each pair of place in the bounded unfolding and number $1 \leq i \leq n$ to encode in which of the n subsequent markings this place is contained.

The matrix ϕ_n of the QBF $\exists \boldsymbol{\mathcal{S}}^b : \forall \boldsymbol{\mathcal{M}}_n : \phi_n$ is defined as follows:

$$\phi_n \stackrel{def.}{=} \bigwedge_{1 \leq i < n} \Big(sequence_i \implies win_i \Big) \wedge \big(sequence_n \implies win_n \wedge loop \big)$$

$$sequence_i \stackrel{def.}{=} initial \wedge seqflow_1 \wedge seqflow_2 \wedge \cdots \wedge seqflow_{i-1}$$

$$initial \stackrel{def.}{=} \bigwedge_{p \in In^b} (p,1) \wedge \bigwedge_{p \in \mathscr{P}^b \setminus In^b} \neg(p,1)$$

$$seqflow_i \stackrel{def.}{=} \bigvee_{t \in \mathscr{T}^b} \Bigg(\bigwedge_{p \in {}^\bullet t} (p,i) \wedge \bigwedge_{p \in {}^\bullet t \cap \mathscr{P}_S^b} (p, \lambda^b(t)) \wedge \bigwedge_{p \in t^\bullet} (p, i+1) \wedge$$

$$\bigwedge_{p \in {}^\bullet t \setminus t^\bullet} \neg(p, i+1) \wedge \bigwedge_{p \in \mathscr{P}^b \setminus ({}^\bullet t \cup t^\bullet)} \big((p,i) \iff (p, i+1)\big)\Bigg)$$

For each simulation point $1 \leq i \leq n$, it is tested whether the variables in $\boldsymbol{\mathcal{M}}_n$ represent a correct $\mathit{sequence}_i$ of markings up to i corresponding to a play in the bounded unfolding. If this is the case then win_i tests whether the marking at i fulfills the requirements to be winning. If $i = n$, i.e., the limit on the simulation is reached, it is additionally tested that a *loop* occurred. A correct *sequence* of markings starts from the *initial* marking followed by the *sequential flow* of $i-1$ enabled and by the system strategy allowed transitions. The *sequential flow* of a transition from time point i requires all places in its preset to contain a token and the system strategy of system places in its preset to allow the transition. Then, at $i+1$, the places of its postset are set to true, places in its preset but not its postset are set to false, and all other places retain their truth value.

$$\mathit{win}_i \overset{def.}{=} \mathit{nobadplace}_i \wedge \mathit{deterministic}_i \wedge \big(\mathit{deadlock}_i \implies \mathit{terminating}_i\big)$$

$$\mathit{nobadplace}_i \overset{def.}{=} \bigwedge_{p \in \mathscr{B}^b} \neg(p,i)$$

$$\mathit{deterministic}_i \overset{def.}{=} \bigwedge_{\substack{t_1,t_2 \in \mathscr{T}, t_1 \neq t_2,\\ {}^\bullet t_1 \cap {}^\bullet t_2 \cap \mathscr{P}^b_S \neq \emptyset}} \Bigg(\bigvee_{p \in {}^\bullet t_1 \cup {}^\bullet t_2} \neg(p,i) \vee \bigvee_{\substack{p_1 \in {}^\bullet t_1 \cap \mathscr{P}^b_S,\\ p_2 \in {}^\bullet t_2 \cap \mathscr{P}^b_S}} \neg(p_1, \lambda^b(t_1)) \vee \neg(p_2, \lambda^b(t_2))\Bigg)$$

$$\mathit{deadlock}_i \overset{def.}{=} \bigwedge_{t \in \mathscr{T}^b} \Bigg(\bigvee_{p \in {}^\bullet t} \neg(p,i) \vee \bigvee_{p \in {}^\bullet t \cap \mathscr{P}^b_S} \neg(p, \lambda^b(t))\Bigg)$$

$$\mathit{terminating}_i \overset{def.}{=} \bigwedge_{t \in \mathscr{T}^b} \Bigg(\bigvee_{p \in {}^\bullet t} \neg(p,i)\Bigg)$$

$$\mathit{loop} \overset{def.}{=} \bigvee_{1 \leq i_1 < i_2 \leq n} \Bigg(\bigwedge_{p \in \mathscr{P}^b} \big((p,i_1) \iff (p,i_2)\big)\Bigg)$$

If $\mathit{sequence}_i$ is fulfilled then win_i tests whether the last marking fulfills the requirements to be winning at i. If $i = n$, i.e., the limit on the simulation is reached, it is additionally tested that a *loop* occurred. The play is winning if *no bad place* is reached, the system makes only *deterministic* decisions (S1), and each *deadlock* is caused by *termination* (S3). A deadlock occurs when no transition is enabled including the choices of the system strategy $\boldsymbol{\mathcal{S}}^b$. Meanwhile, termination occurs when no transition is enabled independently of the system strategy. Therefore, $\mathit{deadlock}_i \implies \mathit{terminating}_i$ ensures that the system does not prevent the reaching of bad places by stopping to fire transitions, but deadlocks are only allowed when the entire game terminates. A *loop* in a Petri game occurs when the same marking is repeated at two different simulation points. As the system strategy has to be deterministic, its behavior repeats infinitely often in the loop such that the system strategy is also winning in an infinite play.

4 True Concurrency in Petri Games

In this section, we define true concurrency in Petri games. Therefore, we first formalize *environment strategies* to explicitly represent environment decisions in response to a given system strategy. This enables us to define the *true concurrent flow semantics* for Petri games, which enforces that transitions are fired as early and as parallel as possible. We prove that this semantics agrees with the interleaving semantics on the existence of a winning strategy for the system.

4.1 Environment Strategy

System strategies represent the system's restrictions of enabled transitions but purely environmental transitions remain uncontrollable. Therefore, a system strategy can result in different fired transitions. We introduce *environment strategies* to explicitly represent decisions of environment players and to obtain a unique sequence of fired transitions up to reordering of independent transitions.

An *environment strategy* $\gamma = (\mathcal{N}^\gamma, \lambda^\gamma)$ is a subprocess of a system strategy $\sigma = (\mathcal{N}^\sigma, \lambda^\sigma)$ (which, in turn, is a subprocess of the unfolding $\beta_U = (\mathcal{N}^U, \lambda^U)$ of the given Petri game $\mathcal{G}$) where environment places can remove outgoing transitions such that the following three requirements hold:

(E1) *Explicit choice*: $\forall p \in \mathcal{P}_E^\gamma : \exists^{\leq 1} t \in \mathcal{T}^\gamma : p \in pre^\gamma(t)$
(E2) *Environment refusal*: $\forall t \in \mathcal{T}^\sigma : t \notin \mathcal{T}^\gamma \wedge pre^\sigma(t) \subseteq \mathcal{P}^\gamma \Rightarrow pre^\sigma(t) \cap \mathcal{P}_E^\gamma \neq \emptyset$
(E3) *Progress*:
$\forall M \in \mathcal{R}(\mathcal{N}^\gamma) : \exists t_\sigma \in \mathcal{T}^\sigma : pre^\sigma(t_\sigma) \subseteq M \Rightarrow \exists t_\gamma \in \mathcal{T}^\gamma : pre^\gamma(t_\gamma) \subseteq M$

Explicit choice requires each environment player to choose at most one of its outgoing transitions. *Environment refusal* enforces environment strategies to only remove transitions with at least one environment place in their preset. *Progress* requires the environment strategy to enable at least one transition for each reachable marking as long as a transition is enabled in the system strategy. Environment strategies resolve the remaining conflicts of a Petri game:

Theorem 1. *An environment strategy γ leads to a unique sequence of fired transitions up to reordering of independent transitions* ($\forall p \in \mathcal{P}^\gamma : |post^\gamma(p)| \leq 1$).

Proof. A system strategy σ satisfies for all system places $p \in \mathcal{P}_S^\sigma$ either the condition $|post^\sigma(p)| \leq 1$ or the non-determinism in the choice of the successor transition is resolved by the environment strategy γ. Since the environment strategy explicitly chooses at most one outgoing transition in each environment place, $\forall p \in \mathcal{P}_S^\gamma : |post^\gamma(p)| \leq 1$ is satisfied. For all environment places $p \in \mathcal{P}_E^\gamma$, the condition $|post^\gamma(p)| \leq 1$ is satisfied by the definition of environment strategies. Since $\mathcal{P}_S^\gamma \cup \mathcal{P}_E^\gamma = \mathcal{P}^\gamma$ holds, $\mathcal{N}^\gamma$ has a unique sequence of fired transitions up to reordering of independent transitions. □

The requirements for environment strategies are similar to the ones for system strategies: (E1) does not iterate over reachable markings in comparison to (S1) to require unique decisions by environment players, (E2) allows differentiation of

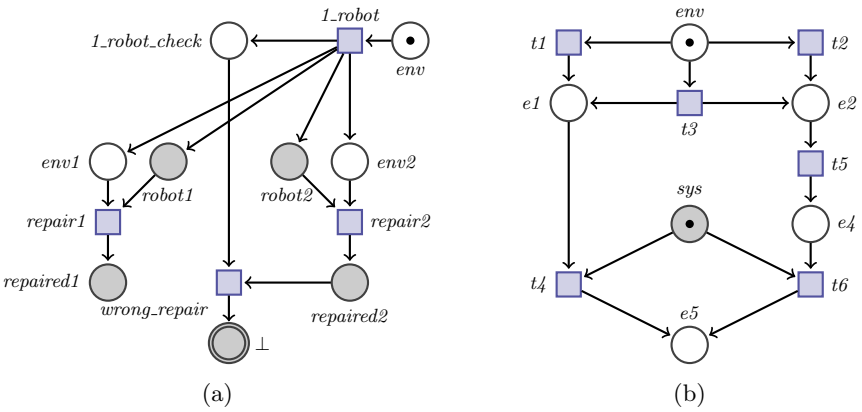

Fig. 3: Two strategies are depicted for the Petri game specifying a production line from Fig. 1: a winning environment strategy for a system strategy (a) and a winning system strategy with more than one outgoing transition at place *sys* (b).

transitions due to the unfolding in comparison to (S2), again, to enable unique decision, and (E3) is (S3) lifted directly to environment strategies.

$\gamma \sqsubseteq_E \sigma$ denotes an environment strategy γ as subprocess of a system strategy σ subject to (E1) to (E3). $\sigma \sqsubseteq_S \beta_U$ denotes a system strategy σ as a subprocess of the unfolding β_U subject to (S1) to (S3). An environment strategy γ is *winning* (and a *counterexample* to the system strategy σ being winning) if it reaches a bad place. We define a system strategy to be *winning* against all its environment strategies: a system strategy σ is winning if no bad places are reached for all environment strategies, i.e., $\forall \gamma \sqsubseteq_E \sigma : \forall M \in \mathscr{R}(\mathcal{N}^\gamma) : \lambda^\gamma[M] \cap \mathscr{B} = \emptyset$.

Figure 3a shows a winning environment strategy for a system strategy of our running example with the bad place $\bot$. By the initial decision for *1_robot* by the environment strategy, the right side of the system strategy becomes unreachable. The system chooses the transitions *repair1* and *repair2* in response to *1_robot* by the environment strategy. By choosing *1_robot*, the second robot should have ignored the product. The system strategy has to enable *wrong_repair* to avoid a deadlock and the environment strategy agress on firing it to reach the bad place.

4.2 True Concurrent Flow Semantics

We define the *true concurrent flow semantics* for Petri games by firing a maximal set of enabled, conflict-free transitions in every step. For the marking M and the set of enabled, conflict-free transitions $T = \{t_1, \ldots, t_n\}$, the successor marking M' is defined by $M[T\rangle M'$, where $pre^{\mathcal{N}}(t_1) \uplus \ldots \uplus pre^{\mathcal{N}}(t_n) \subseteq M$ and $M' = (M \backslash (pre^{\mathcal{N}}(t_1) \uplus \ldots \uplus pre^{\mathcal{N}}(t_n))) \uplus post^{\mathcal{N}}(t_1) \uplus \ldots \uplus post^{\mathcal{N}}(t_n)$. The set of reachable markings according to the true concurrent flow semantics is defined by

$$\mathscr{R}^{tc}(\mathcal{N}) = \{M \subseteq \mathscr{P} \mid \exists \text{ maximal } T_1, \ldots, T_n \subseteq \mathscr{T} : \exists M_1, \ldots, M_n \subseteq \mathscr{P} : In[T_1\rangle M_1[T_2\rangle \ldots [T_n\rangle M_n = M\} \quad \text{where } |T_1|, \ldots, |T_n| > 0$$

We denote the set of reachable markings in the sequential flow semantics by $\mathscr{R}^{seq}(\mathcal{N}) = \mathscr{R}(\mathcal{N})$. Firing all enabled transitions in the true concurrent flow semantics at once yields a unique sequence of markings and therefore a unique sequence of sets of fired transitions. This brings us to the following theorem:

Theorem 2. *There exists a winning system strategy of a Petri game under the sequential flow semantics iff there exists a winning system strategy of a Petri game under the true concurrent flow semantics.*

Proof. We show that (1) $\exists \sigma \sqsubseteq_S \beta_U : \forall M \in \mathscr{R}^{seq}(\mathcal{N}^\sigma) : \lambda^\sigma[M] \cap \mathscr{B} = \emptyset \iff \exists \sigma \sqsubseteq_S \beta_U : \forall \gamma \sqsubseteq_E \sigma : \forall M \in \mathscr{R}^{seq}(\mathcal{N}^\gamma) : \lambda^\gamma[M] \cap \mathscr{B} = \emptyset$ and that (2) $\exists \sigma \sqsubseteq_S \beta_U : \forall \gamma \sqsubseteq_E \sigma : \forall M \in \mathscr{R}^{seq}(\mathcal{N}^\gamma) : \lambda^\gamma[M] \cap \mathscr{B} = \emptyset \iff \exists \sigma \sqsubseteq_S \beta_U : \forall \gamma \sqsubseteq_E \sigma : \forall M \in \mathscr{R}^{tc}(\mathcal{N}^\gamma) : \lambda^\gamma[M] \cap \mathscr{B} = \emptyset$. Since (1) is based on the sequential flow, every sequence of markings in $\mathscr{R}^{seq}(\mathcal{N}^\sigma)$ can be produced with an environment strategy choosing exactly the transitions of the sequence and vice versa. For (2), we show that the environment wins on the same nets by reaching a bad place: either $\exists \gamma \sqsubseteq_E \sigma : \exists M \in \mathscr{R}^{seq}(\mathcal{N}^\gamma) : \lambda^\gamma[M] \cap \mathscr{B} \neq \emptyset$ holds or not. As each environment strategy results in a unique sequence of fired transitions (up to reordering of independent transitions), the sets of reachable places in the reachable markings $\mathscr{R}^{seq}(\mathcal{N}^\gamma)$ and $\mathscr{R}^{tc}(\mathcal{N}^\gamma)$ are the same. □

5 True Concurrent Encoding of Bounded Synthesis

We show how the requirements (E1) to (E3) on environment strategies and the true concurrent flow semantics can be encoded as QBF. We introduce *stalling* of transitions to let environment players find non-determinism in a system strategy. Furthermore, we present how the true concurrent flow semantics can be used to detect loops earlier in the encoding of bounded synthesis for Petri games.

5.1 Stalling of Transitions to Find Non-Determinism

To use the true concurrent flow semantics in bounded synthesis for Petri games, we ensure that all possible system strategies fulfill the assumptions (S1) to (S3) and do not reach any bad place. The determinism requirement can be violated when the sequential flow encoding is simply replaced by the true concurrent flow encoding as markings may be skipped by firing transitions as early as possible.

Figure 3b shows a Petri game without bad places. It is not winning for the system, as *t4* and *t6* have to be enabled (deadlock-avoidance) and there is a marking where both transitions are enabled (non-determinism). This contradicts *determinism* (S1) but in the true concurrent flow semantics, *t4* will always be fired before *t6* such that the marking with non-determinism of the system is never reached. To check the requirements for system strategies in the true concurrent encoding, environment players can *stall* transitions with at least one system place in their preset globally to catch up with the system. The requirement *determinism* (S1) can only be violated at system places. In Fig. 3b, the environment strategy needs to stall the firing of $t4$ until $t5$ is fired to prove that a potential system strategy enabling both transitions is non-deterministic.

5.2 Encoding True Concurrency as QBF

We extend the sequential encoding of bounded synthesis for Petri games [7,8] to environment strategies with stalling and the true concurrent flow semantics. The strategy of the environment is translated into additional universally quantified variables. The QBF-formula is $\exists \mathcal{S}^b : \forall \mathcal{M}_n : \forall \mathcal{E}^b : \phi_n$ with $\mathcal{E}^b$ as the union of variables for each environment choice in the firing of transitions and variables for transitions with at least one system place in their preset to stall their progress. This encoding preserves the requirement of *environment refusal* (E2):

$$\mathcal{E}^b \overset{def.}{=} \{(p,t,i) \mid p \in \mathscr{P}_E^b \wedge t \in p^\bullet \wedge 1 \leq i < n\} \cup \{(t) \mid t \in \mathscr{T}^b \wedge {}^\bullet t \cap \mathscr{P}_S^b \neq \emptyset\}$$

Bounded unfoldings may contain loops. The variables for the environment are different for every simulation point, such that decisions of revisited environment places do not depend on previous visits. By contrast, a global decision independent of the simulation points suffices for stalling. The case when variable (t) is set to false results in the stalling of transition t. In the following, not mentioned formulas are as they are in the sequential encoding. We apply the requirement *explicit choice* (E1) of the environment strategy to ϕ_n and encode it in *choice*:

$$\phi_n \overset{def.}{=} \mathit{choice} \implies \bigwedge_{1 \leq i < n} \left(\mathit{seq}_i \implies \mathit{win}_i \right) \wedge \left(\mathit{seq}_n \implies \mathit{win}_n \wedge \mathit{loop} \right)$$

$$\mathit{choice} \overset{def.}{=} \bigwedge_{p \in \mathscr{P}_E^b, 1 \leq i < n} \left(\bigvee_{t \in p^\bullet} \left((p,t,i) \wedge \bigwedge_{t' \in p^\bullet \setminus \{t\}} \neg (p,t',i) \right) \right)$$

$$\mathit{seq}_i \overset{def.}{=} \mathit{initial} \wedge \mathit{tcflow}_1 \wedge \mathit{tcflow}_2 \wedge \cdots \wedge \mathit{tcflow}_{i-1}$$

Each environment place has to choose exactly one outgoing transition which results in the firing of at most one outgoing transitions per environment place, because the other places in the preset of the transition also have to decide for the transition. This encoding furthermore ensures *progress* (E3). We substitute the sequential flow $\mathit{seqflow}_i$ by the true concurrent flow tcflow_i, which enforces the firing of all enabled and not stalled transitions and maintains all other tokens.

$$\mathit{tcflow}_i \overset{def.}{=} \mathit{fireenabled}_i \wedge \mathit{updateplaces}_i$$

$$\mathit{fireenabled}_i \overset{def.}{=} \bigwedge_{t \in \mathscr{T}^b} \left(\mathit{enabled}_{i,t} \implies \bigwedge_{p \in {}^\bullet t \setminus t^\bullet} \neg (p, i+1) \wedge \bigwedge_{p \in t^\bullet} (p, i+1) \right)$$

$$\mathit{updateplaces}_i \overset{def.}{=} \bigwedge_{p \in \mathscr{P}^b} \left(\bigwedge_{t \in {}^\bullet p \cup p^\bullet} \neg \mathit{enabled}_{i,t} \implies \left((p,i) \iff (p, i+1) \right) \right)$$

$$\mathit{enabled}_{i,t} \overset{def.}{=} \bigwedge_{p \in {}^\bullet t} (p,i) \wedge \bigwedge_{p \in \mathscr{P}_S^b \cap {}^\bullet t} (p, \lambda^b(t)) \wedge \bigwedge_{p \in \mathscr{P}_E^b \cap {}^\bullet t} (p,t,i) \wedge (t)$$

$\mathit{enabled}_{i,t}$ requires tokens in all places in the preset of the transition, both the system and the environment strategy to allow the transition for corresponding places in the preset of the transition, and that stalling allows the transition.

win_i remains unchanged. Therefore, environment strategies and stalling only affect the flow of tokens but not the check that reached markings are winning.

5.3 Shorter Loops via Strongly Connected Components

Environment strategies allow us to define the true concurrent flow semantics which allows us to detect loops earlier by searching for them in *strongly connected components* (SCCs) [22]. The definition of SCCs can be directly lifted to Petri games by including an additional set with all places that are not in any other SCC. With SCCs, we find loops in independent parts of the Petri game as early as possible. We encode that a loop no longer only occurs at the repetition of a global marking but also when all $SCCs \subseteq 2^{\mathcal{P}^b}$ repeat their marking, respectively:

$$loop \stackrel{def.}{=} \bigwedge_{scc \in SCCs} \left(\bigvee_{1 \leq i_1 < i_2 \leq n} \left(\bigwedge_{p \in scc} \left((p, i_1) \iff (p, i_2) \right) \right) \right)$$

6 Experimental Results

We compare the sequential encoding [7] with our new true concurrent encoding from Section 5 on five benchmark families. At first, we describe the asynchronous and distributed nature of these benchmark families stemming from alarm systems, routing, robotics, and communication protocols. Afterwards, we outline the technical details of our comparison framework and state our observations and explanations concerning the observed times for finding winning strategies.

6.1 Benchmark Families

Table 1 refers to the following scalable benchmark families where Collision Avoidance, Disjoint Routing, and Production Line are new benchmark families:

- **AS**: *Alarm System* [8]. *Parameters:* m locations. There are m secured locations and a burglar can intrude one of them. The local alarm system of each location can communicate with all other local alarm systems. The local alarm systems should indicate the position of an intrusion and should not issue unsubstantiated warnings of an intrusion.
- **CA:** *Collision Avoidance*. *Parameters:* m robots. A subset of m robots is initialized to drive on individual paths of increasing length with several goal states. They should avoid collisions and drive forever on the chosen route.
- **DR:** *Disjoint Routing*. *Parameters:* m packets. In a software-defined network, m packets should be routed disjointly between an ingress and an egress switch where the network allows m disjoint paths between the two switches.
- **PL:** *Production Line*. *Parameters:* m robots. The m independent robots are able to repair or ignore m features of a product. Depending on the product, some features need to be repaired while others must not be repaired.
- **DW:** *Document Workflow* [9]. *Parameters:* m workers. A document circulates between m workers with the environment choosing the first worker. It is required that all workers unanimously endorse or reject the document.

Table 1: Benchmarking results on our Petri game *benchmark families* for increasing *parameters*. For the *sequential* and the *true concurrent* encoding, the needed model checking *iterations* with accumulated *runtime in seconds* are reported.

		Sequential		*True Concurrent*	
Ben.	*Par.*	*Iter.*	*Runtime in sec.*	*Iter.*	*Runtime in sec.*
AS	2	7	13.26	**6**	**11.15**
	3	-	timeout	-	timeout
CA	2	8	7.27	**5**	**6.25**
	3	-	timeout	**6**	**14.21**
	4	-	timeout	**7**	**346.23**
	5	-	timeout	-	timeout
DR	2	8	6.16	**7**	**6.05**
	3	11	11.03	**9**	**10.07**
	4	14	69.50	**11**	**65.31**
	5	-	timeout	-	timeout
PL	1	**4**	**5.59**	**4**	**5.59**
	2	5	6.08	**4**	**5.85**
	3	6	8.51	**4**	**6.95**
	4	7	20.99	**4**	**12.54**
	5	8	87.33	**4**	**41.95**
	6	-	timeout	**4**	**742.36**
	7	-	timeout	-	timeout
DW	1	8	5.90	**7**	**5.79**
	2	10	6.58	**9**	**6.44**
	3	12	7.90	**11**	**7.80**
	4	14	11.45	**13**	**11.22**
	5	16	**16.59**	**15**	19.82
	...	...	...	...	...
	10	26	**716.61**	**25**	823.94
	11	**28**	**1304.14**	-	timeout
	12	-	timeout	-	timeout

6.2 Comparison Framework

As both the sequential and the true concurrent encoding result in a 2-QBF not in conjunctive normal form, we use the QBF solver QuAbS [33,19]. The results from Table 1 were obtained on an Intel i7-2700K CPU with 3.50 GHz and 32 GB RAM and are the average over five runs. For each benchmark family (column *Ben.*), we report on the attempted parameters of the benchmark (*Par.*), the necessary model checking iterations (*Iter.*) of bounded synthesis, and on the runtime for finding a winning system strategy. A timeout of 30 minutes is used. We prepared an artifact to replicate our experimental results [18].

6.3 Observation

The true concurrent encoding shows considerable improvements over the sequential encoding on the presented benchmark set: It solves more instances and has

mostly faster solving times as shown in Table 1. The improvements are based on fewer model checking iterations of the bounded synthesis algorithm witnessed by the *Iter.* column. The lower iteration count and runtime are indicated in bold.

We can make the following observations concerning the specific benchmark families: The complex communication structure of Alarm System prevents larger examples to be synthesized because the alarm system observing the intrusion has to broadcast the information to all other alarm systems. Similarly, Collision Avoidance has a complex pairwise communication structure which can be better synthesized by the true concurrent encoding. The simpler communication structure of Production Line allows constant bounds for the true concurrent encoding compared to linearly increasing bounds for the sequential encoding. The communication structure of Disjoint Routing lays between complex and simple such that the true concurrent encoding enables a smaller linear increase in the bound. The true concurrent encoding therefore can solve larger examples even though the bounded unfolding grows with the number of considered players for both encodings. The possibilities for communication of information are less open in the benchmark families DR and PL whereas they are completely open in the benchmark family AS and CA. In Document Workflow, the communication structure is fixed to a specific pairwise ring between neighboring clerks. However, this prevents almost all true concurrency between them. The difference in bound of one is caused by the concurrent test that all workers have seen the document and that the decisions of workers have been unanimously.

7 Related Work

The *control problem of asynchronous automata* is an alternative approach to the synthesis of distributed asynchronous systems with causal memory. The modeling with asynchronous automata does not allow the spawning and termination of players. Also, it does not explicitly represent environment processes. Instead, every process can have uncontrollable behavior. The decidability of the control problem of asynchronous automata is open in general [28]. There are some decidability results for the control problem of asynchronous automata for restrictions on the dependencies of actions [15] or on the synchronization behavior [25,26]. Decidability has also been obtained for acyclic communication structures [16,29]. The class of *Decomposable games* [17] proposes a new proof technique to unify and extend these results. Recently, an exponential gap between the control problem of asynchronous automata and Petri games has been identified [1].

There is a broad theory and several implementations for model checking of distributed systems: For Petri nets as representation of distributed systems, it often suffices to only consider finite prefixes of the unfolding [24,5,3]. It is most interesting whether these results can be lifted to Petri games and causal past. Partial order reduction and true concurrency have been studied thoroughly to speed up the model checking of finite distributed systems [20,21,14,27]. The systems we synthesize are especially powerful as both the system and the environment can run infinitely and non-determinism of the environment is represented.

8 Conclusion

We presented how to utilize concurrency in bounded synthesis for asynchronous distributed systems by firing as many true concurrent transitions as possible in our new true concurrent encoding. The previous sequential encoding enumerated all interleavings. For the true concurrent encoding, we represent the decisions of the environment players explicitly as environment strategies for Petri games and showed that this enables us to fire all enabled transitions as early as possible while maintaining the existence of winning system strategies. The experimental results show that our tool implementation of the true concurrent encoding outperforms the sequential encoding on all benchmark families by a considerable margin. Even in the rare case of benchmark families without true concurrent transitions, the true concurrent encoding slightly outperforms the sequential encoding despite resulting in larger QBFs.

For future work, we want to apply environment strategies and true concurrency in Adam to improve synthesis for a bounded number of system players and one environment players. Furthermore, we plan to extend the bounded synthesis encoding further. On the one hand, we want to identify disconnected parts of the Petri game, solve them in isolation, and compose them back together. On the other hand, we plan to extend the expressivity of considered winning conditions. Local liveness conditions of places to reach should be straightforward whereas global winning conditions in the form of markings to reach or avoid could prove difficult for the true concurrent encoding as certain interleavings may be skipped. Therefore, we believe that local winning conditions on the progress of individual tokens could be a good middle ground between the current local winning conditions of bad places and global winning conditions.

References

1. Beutner, R., Finkbeiner, B., Hecking-Harbusch, J.: Translating asynchronous games for distributed synthesis. In: Proceedings of CONCUR. pp. 22:1–22:16 (2019)
2. Bohy, A., Bruyère, V., Filiot, E., Jin, N., Raskin, J.: Acacia+, a tool for LTL synthesis. In: Proceedings of CAV. pp. 652–657 (2012)
3. Bonet, B., Haslum, P., Khomenko, V., Thiébaux, S., Vogler, W.: Recent advances in unfolding technique. Theor. Comput. Sci. **551**, 84–101 (2014)
4. Ehlers, R.: Unbeast: Symbolic bounded synthesis. In: Proceedings of TACAS. pp. 272–275 (2011)
5. Esparza, J., Heljanko, K.: Unfoldings – A Partial-Order Approach to Model Checking. Springer (2008)
6. Faymonville, P., Finkbeiner, B., Rabe, M.N., Tentrup, L.: Encodings of bounded synthesis. In: Proceedings of TACAS. pp. 354–370 (2017)
7. Finkbeiner, B.: Bounded synthesis for petri games. In: Correct System Design. pp. 223–237 (2015)
8. Finkbeiner, B., Gieseking, M., Hecking-Harbusch, J., Olderog, E.: Symbolic vs. bounded synthesis for petri games. In: Proceedings of SYNT. pp. 23–43 (2017)

9. Finkbeiner, B., Gieseking, M., Olderog, E.: Adam: Causality-based synthesis of distributed systems. In: Proceedings of CAV. pp. 433–439 (2015)
10. Finkbeiner, B., Gölz, P.: Synthesis in distributed environments. In: Proceedings of FSTTCS. pp. 28:1–28:14 (2017)
11. Finkbeiner, B., Olderog, E.: Petri games: Synthesis of distributed systems with causal memory. Inf. Comput. **253**, 181–203 (2017)
12. Finkbeiner, B., Schewe, S.: Uniform distributed synthesis. In: Proceedings of LICS. pp. 321–330 (2005)
13. Finkbeiner, B., Schewe, S.: Bounded synthesis. STTT **15**(5-6), 519–539 (2013)
14. Flanagan, C., Godefroid, P.: Dynamic partial-order reduction for model checking software. In: Proceedings of POPL. pp. 110–121 (2005)
15. Gastin, P., Lerman, B., Zeitoun, M.: Distributed games with causal memory are decidable for series-parallel systems. In: Proceedings of FSTTCS. pp. 275–286 (2004)
16. Genest, B., Gimbert, H., Muscholl, A., Walukiewicz, I.: Asynchronous games over tree architectures. In: Proceedings of ICALP. pp. 275–286 (2013)
17. Gimbert, H.: On the control of asynchronous automata. In: Proceedings of FSTTCS. pp. 30:1–30:15 (2017)
18. Hecking-Harbusch, J., Metzger, N.O.: BoundedAdam – Efficient Trace Encodings for Bounded Synthesis of Petri Games (2019). https://doi.org/10.6084/m9.figshare.8313215
19. Hecking-Harbusch, J., Tentrup, L.: Solving QBF by abstraction. In: Proceedings of GandALF. pp. 88–102 (2018)
20. Heljanko, K.: Using logic programs with stable model semantics to solve deadlock and reachability problems for 1-safe petri nets. Fundam. Inform. **37**(3), 247–268 (1999)
21. Heljanko, K.: Combining symbolic and partial order methods for model checking 1-safe Petri nets. Ph.D. thesis, Aalto University, Helsinki, Finland (2002)
22. Jensen, K.: Coloured Petri nets: basic concepts, analysis methods and practical use, vol. 2. Springer Science & Business Media (2013)
23. Jobstmann, B., Galler, S.J., Weiglhofer, M., Bloem, R.: Anzu: A tool for property synthesis. In: Proceedings of CAV. pp. 258–262 (2007)
24. Khomenko, V., Koutny, M., Vogler, W.: Canonical prefixes of petri net unfoldings. Acta Inf. **40**(2), 95–118 (2003)
25. Madhusudan, P., Thiagarajan, P.S.: A decidable class of asynchronous distributed controllers. In: Proceedings of CONCUR. pp. 145–160 (2002)
26. Madhusudan, P., Thiagarajan, P.S., Yang, S.: The MSO theory of connectedly communicating processes. In: Proceedings of FSTTCS. pp. 201–212 (2005)
27. Meulen, J.V., Pecheur, C.: Combining partial order reduction with bounded model checking. In: Proceedings of CPA. pp. 29–48 (2009)
28. Muscholl, A.: Automated synthesis of distributed controllers. In: Proceedings of ICALP. pp. 11–27 (2015)
29. Muscholl, A., Walukiewicz, I.: Distributed synthesis for acyclic architectures. In: Proceedings of FSTTCS. pp. 639–651 (2014)
30. Pnueli, A., Rosner, R.: On the synthesis of an asynchronous reactive module. In: Proceedings of ICALP. pp. 652–671 (1989)
31. Pnueli, A., Rosner, R.: Distributed reactive systems are hard to synthesize. In: Proceedings of FOCS. pp. 746–757 (1990)
32. Reisig, W.: Petri Nets: An Introduction. Springer (1985)
33. Tentrup, L.: Non-prenex QBF solving using abstraction. In: Proceedings of SAT. pp. 393–401 (2016)