Symbolic vs. Bounded Synthesis for Petri Games[*]

Bernd Finkbeiner
Universität des Saarlandes
Saarbrücken, Germany
finkbeiner@react.uni-saarland.de

Manuel Gieseking
Carl von Ossietzky Universität Oldenburg
Oldenburg, Germany
gieseking@informatik.uni-oldenburg.de

Jesko Hecking-Harbusch
Universität des Saarlandes
Saarbrücken, Germany
hecking-harbusch@react.uni-saarland.de

Ernst-Rüdiger Olderog
Carl von Ossietzky Universität Oldenburg
Oldenburg, Germany
olderog@informatik.uni-oldenburg.de

Petri games are a multiplayer game model for the automatic synthesis of distributed systems. We compare two fundamentally different approaches for solving Petri games. The symbolic approach decides the existence of a winning strategy via a reduction to a two-player game over a finite graph, which in turn is solved by a fixed point iteration based on binary decision diagrams (BDDs). The bounded synthesis approach encodes the existence of a winning strategy, up to a given bound on the size of the strategy, as a quantified Boolean formula (QBF). In this paper, we report on initial experience with a prototype implementation of the bounded synthesis approach. We compare bounded synthesis to the existing implementation of the symbolic approach in the synthesis tool ADAM. We present experimental results on a collection of benchmarks, including one new benchmark family, modeling manufacturing and workflow scenarios with multiple concurrent processes.

1 Introduction

The *synthesis problem* asks, given a formal specification, for the existence of an implementation and derives it automatically if existent [2]. This problem can be described as a game between the system and the environment. A strategy is winning for the system and therefore corresponds to an implementation if the strategy fulfills the winning condition of the game against all behaviors of the environment. The synthesized implementation for a given specification is correct by construction, which is beneficial for error-prone implementation tasks. Synthesis has been applied successfully to implement several practical applications like the protocol of the AMBA bus circuit [1].

Synthesis is especially interesting for *distributed systems* where several concurrent components can communicate with each other to fulfill the specification. Pnueli and Rosner defined the first setting for distributed synthesis [14]. *Information forks* have been identified as a necessary and sufficient criterion for the undecidability of the synthesis problem in this setting [9], which prevents most practical applications. Even for the decidable cases without information forks like pipelines and rings, the synthesis problem has non-elementary complexity [15, 11]. More recently, Zielonka's asynchronous automata were introduced as another framework for distributed synthesis. Whereas the synthesis problem for some cases has non-elementary complexity, the general complexity of synthesis for asynchronous automata remains open [18, 12].

We take Petri games as a starting point to synthesize distributed systems [8]. Petri games define a multiplayer game model where several distributed system players cooperatively play against several

[*]This work was partially supported by the European Research Council (ERC) Grant OSARES (No. 683300) and by the German Research Foundation through the Research Training Group (DFG GRK 1765) SCARE.

distributed environment players. Petri games are based on an underlying Petri net where each token represents a player. Each player in a Petri game has only local information about other players it synchronized with on joint transitions.

We compare two fundamentally different approaches to synthesize winning strategies of Petri games: The *symbolic* approach is based on the result that the synthesis problem for Petri games with a bounded number of system players, a single environment player, and the avoidance of bad places as winning condition can be solved in single exponential time [8]. This approach performs a reduction to a two-player game over a finite graph with full information. The implementation is realized in the tool ADAM using a BDD-based fixed point iteration [6]. The restriction to only a single environment player is an impediment for convenient modeling.

The second approach to find winning strategies in a Petri game is *bounded* synthesis [10]. Here, the size of possible strategies and of the proof that the strategy is winning is limited. It is increased incrementally when no strategy of the previous size can be found. This steers the search towards small winning strategies at the cost of being unable to prove the non-existence of a winning strategy. Bounded synthesis can find winning strategies for Petri games with more than one environment player [5]. The bounded approach for Petri games limits the number of fired transitions in the proof of the strategy being winning. It further takes a second bound on the size of the memory each player can utilize as the memory of players is infinite in general. For the pair of bounds, bounded synthesis searches for termination or loops in the strategy and can thereby prove the existence of an infinite winning strategy. This search is encoded in a quantified Boolean formula (QBF) and solved by a QBF solver. We report on our experience with a first prototype implementation of bounded synthesis generating the QBF and solving it with the QBF solver QUABS [17] in comparison to the symbolic approach implemented in ADAM.

ADAM has been used to synthesize several case studies from manufacturing and workflow scenarios. These case studies have been extended into scalable benchmark families to evaluate the behavior of the implementation of the symbolic approach and to show the applicability of Petri games to synthesize distributed systems [6].

The key contributions of this paper are the following:

- We add the new benchmark family of a distributed *alarm system* to the benchmark families of ADAM. The new benchmark family serves to secure several locations with an alarm system such that a burglary at any location is indicated at all locations including the information where exactly the burglary takes place.
- We state our experience with the prototype implementation for the generation of the QBFs representing the bounded synthesis problem for the given pair of bounds and with the solving of the generated QBFs. We found out that the bounded unfolding of Petri games benefits from pruning techniques and that solvers for non-CNF QBFs refining an abstraction based on counterexamples show the best performance for solving.
- We empirically compare the symbolic synthesis approach implemented in ADAM with the bounded synthesis approach realized by our prototype implementation. The symbolic approach solves more instances overall from the extended set of benchmark families whereas bounded synthesis derives strategies of smaller size. We present reasons for the observed behavior based on the number of variables in the two approaches.

The remainder of the paper is structured as follows. Section 2 recaps the theory of Petri games on an abstract level and introduces the distributed alarm system as a running example. In Sec. 3, the symbolic and the bounded approach for solving Petri games are presented, including their application to the distributed alarm system. The experimental results comparing the two approaches are given in Sec. 4.

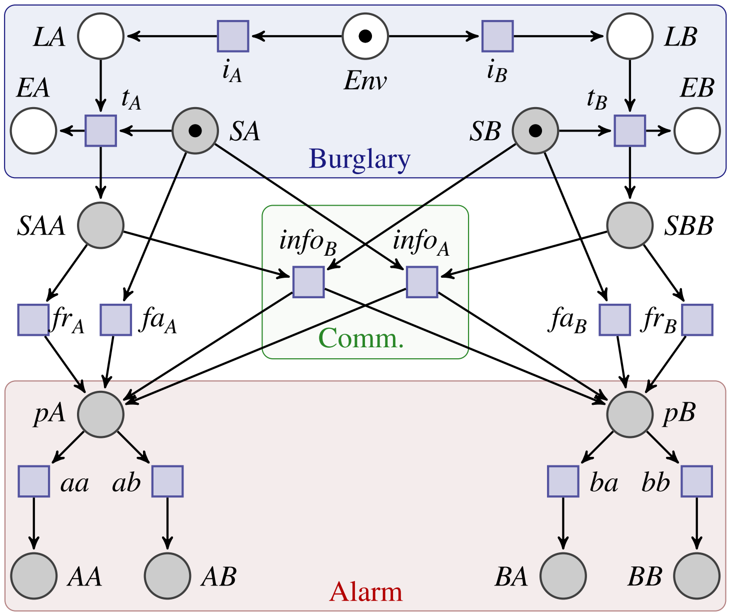


Figure 1: *The Petri game depicting the synthesis problem of an alarm system being distributed over two independent locations with the possibility to communicate. The alarm system has to detect a burglary and set off alarms at both locations.*

2 Petri Games

Petri games are a multiplayer game model for the synthesis of distributed systems [8]. They define a game on an underlying Petri net by characterizing certain places as bad such that the system has to cooperate to avoid reaching these places to win. The distinction between system and environment is achieved by distributing each place of the Petri net to either belong to the system or to the environment. System places are depicted gray whereas environment places remain white. The players in a Petri game are the tokens flowing through the underlying Petri net. If a token resides in a system place then it is controlled by the token's strategy whereas the behavior of tokens in environment places is uncontrollable. Each token in a Petri game has local memory, which is only exchanged with the other participating tokens of joint transitions. The causal history of tokens is utilized by their local strategy to make decisions on which transitions to fire. Therefore, the strategy of the system in a Petri game is a local controller for each process and not a global controller with information about the state of tokens in all system places.
Example. *We model a distributed alarm system. A burglar (modeled by the environment player residing in place Env) decides to intrude one of several secured locations. In Fig. 1, the situation is displayed for the distributed locations A (shown to the left) and B (shown to the right). This situation has been used as the introductory example in [8] and is extended to a benchmark family for an arbitrary number of secured locations later in this paper. The goal of the game is to find a strategy for each of the two alarm processes initially residing in places SA and SB. A token in place XY (for $X,Y \in \{A,B\}$) represents that in location X an alarm is set off indicating that the strategy of the alarm system presumes an intrusion at location Y. If the burglar intrudes location A by entering place LA, the strategies should steer the token in SA to place AA and the token in SB to place BA in order to correctly indicate the burglar's intrusion, and similarly for the burglar intruding location B. If a token resides in one of the system places SA, SB, SAA, SBB, pA, or pB then the strategy of the player has to resolve nondeterminism between the outgoing*

transitions. For this, the strategies for the players in SA and SB need to collect sufficient information about the moves of the other system player and the burglar. For example, the player in place SB does not know whether the alarm system in SA has fired transition t_A unless it gets informed by the communication transition $info_B$. We omit the bad places and transitions to them representing false alarms and false reports, which have to be avoided by a correct alarm system. A false alarm occurs when an intrusion is indicated before the burglar actually intruded the object whereas a false report occurs when the alarm system at a certain location indicates an intrusion at a location where no intrusion occurred.

Formally, a Petri game $\mathscr{P} = (\mathscr{P}_S, \mathscr{P}_E, \mathscr{T}, \mathscr{F}, In, \mathscr{B})$ is based on a Petri net $\mathscr{N} = (\mathscr{P}, \mathscr{T}, \mathscr{F}, In)$ with the set of places $\mathscr{P} = \mathscr{P}_S \cup \mathscr{P}_E$, the set of transitions $\mathscr{T}$, the flow relation $\mathscr{F} \subseteq (\mathscr{P} \times \mathscr{T}) \cup (\mathscr{T} \times \mathscr{P})$, and the initial marking In. A Petri game divides the places to either belong to the system ($\mathscr{P}_S$) or to the environment ($\mathscr{P}_E$) and it further defines some places as bad ($\mathscr{B} \subseteq \mathscr{P}$). We restrict ourselves to *safe* Petri games, i.e., at most one token can reside in each place. We identify a token residing in a place by the name of the place. A *marking* is a set of places where one token resides at each place. From a marking, a transition is *enabled* if all places in the transition's preset have one token residing in them. The *firing* of an enabled transition removes one token from each place in the transition's preset (${}^\bullet t$) and creates one token in each place in the transition's postset ($t^\bullet$). We also refer to the sets of transitions preceding and following a place as the place's preset (${}^\bullet p$) and postset ($p^\bullet$). The behavior of Petri games is defined by sequences of *reachable markings*. These sequences start from the initial marking and between each pair of subsequent markings one transition is fired. We say that a place p is *reachable* if there exists a sequence of reachable markings such that one of the markings contains p.

The notion of *unfoldings* from Petri nets [4] translates to Petri games. In an unfolding, all cycles, i.e., all reachable sequences of markings starting and ending with the same marking, are unrolled infinitely and all backward branches, i.e., places with at least two transitions merging into them, are unfolded. A place is unfolded by copying the place and all its outgoing transitions including the following subgames and changing one of the incoming flows of a transition from the place's preset to the copied place. Therefore, the unfolding explicitly represents the unique causal history of each process. As for Petri nets, the unfolding can be infinite for Petri games. In *bounded unfoldings* of Petri games [5], loops and backward branches are only unfolded up to a given bound b for the number of copies per place. Upon reaching the bound for a place, both the original place and its existing copies can be part of a loop or can have backward branch. Therefore, the bounded unfolding is finite even for Petri games with infinite unfoldings. One bounded unfolding for the Petri game in Fig. 1 is depicted in Fig. 5 in Sec. 3.2. The places *AA*, *AB*, *BA*, and *BB* are not unfolded despite them being reachable with different causal pasts, i.e., having backward branches.

A *strategy* of a Petri game is a restriction to the unfolding of the Petri game where certain branches of transitions and places are removed. We assume that this removal is based on the decisions of system players not to fire certain transitions. The behavior of a strategy is defined by the remaining sequences of reachable markings. The following four requirements have to hold for a strategy to be *winning*:

1. *Safety.* No marking containing a bad place is reachable in the strategy.
2. *Determinism.* For all system places in all reachable markings of the strategy, at most one outgoing transition is enabled.
3. *Deadlock avoidance.* For all reachable markings in the strategy, if a transition is enabled for that marking in the unfolding then one transition is enabled in the strategy as well. This requirement prevents trivial solutions where the system does not fire any transition to avoid reaching a marking containing a bad place.

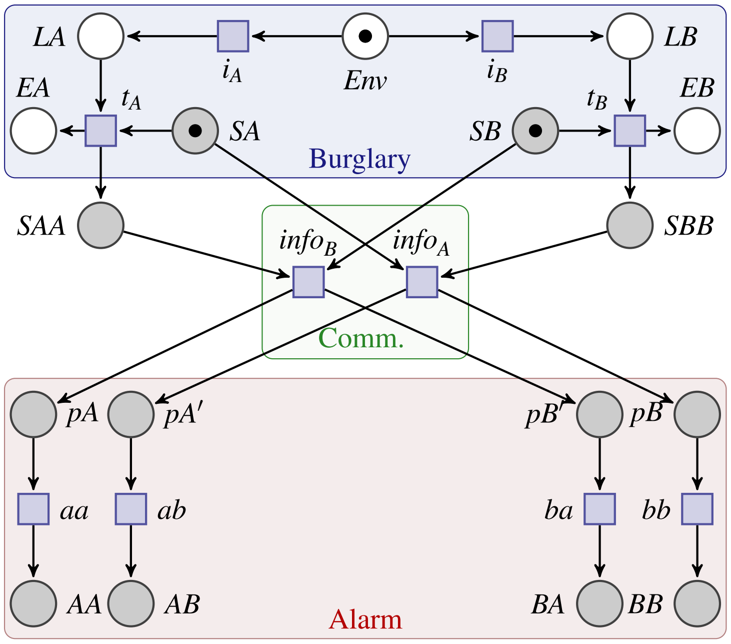


Figure 2: *The winning strategy for the system players for the Petri game depicted in Fig. 1 modeling an alarm system. To win, both system players have to communicate the detection of the burglary before setting off their alarms.*

4. *Justified refusal.* When a transition is removed from the unfolding to create the strategy then there is at least one system place in the removed transition's preset for which all copies of the removed transition are removed as well. This prevents the system from differentiating copies of transitions resulting from the unfolding.

Petri games with a bounded number of system tokens, one environment token, and bad places as winning condition can be solved in single exponential time [8].

Example. *The winning strategy for the alarm system from Fig. 1 is depicted in Fig. 2. The displayed strategy avoids bad places because no false alarm can occur, as the alarm is always triggered after the burglar intruded, and because no false report can occur, as both system tokens exchange information and utilize it to indicate the correct location of intrusion. The strategy is deterministic because only one of the two respective outgoing transitions of SA and SB is enabled, depending on the location of intrusion, and all other system places have only one outgoing transition. The strategy is deadlock-avoiding because after indicating the alarm, the system terminates as no transitions are enabled. The unfolding of the game only allows justified refusal by the system. Therefore, the displayed strategy is winning.*

The strategy contains the local controllers for the alarm systems at location A and at location B. The local controller for the alarm system at A behaves in the following way. The alarm system at SA waits until it either recognizes an intrusion at A via transition t_A or is informed about an intrusion at B via transition $info_A$. The place pA' is reached after getting informed about an intrusion at B from which the transition ab is fired setting off an alarm at A indicating an intrusion at B. The place SAA is reached after recognizing an intrusion at A. The local alarm system informs the local alarm system at B with the transition $info_B$ and afterwards fires the transition aa to reach the place AA. This place represents an alarm at location A that there was an intrusion at location A. The two local system controllers can be found in Fig. 3. Note that they can only behave correctly as they rely on the other local alarm system to

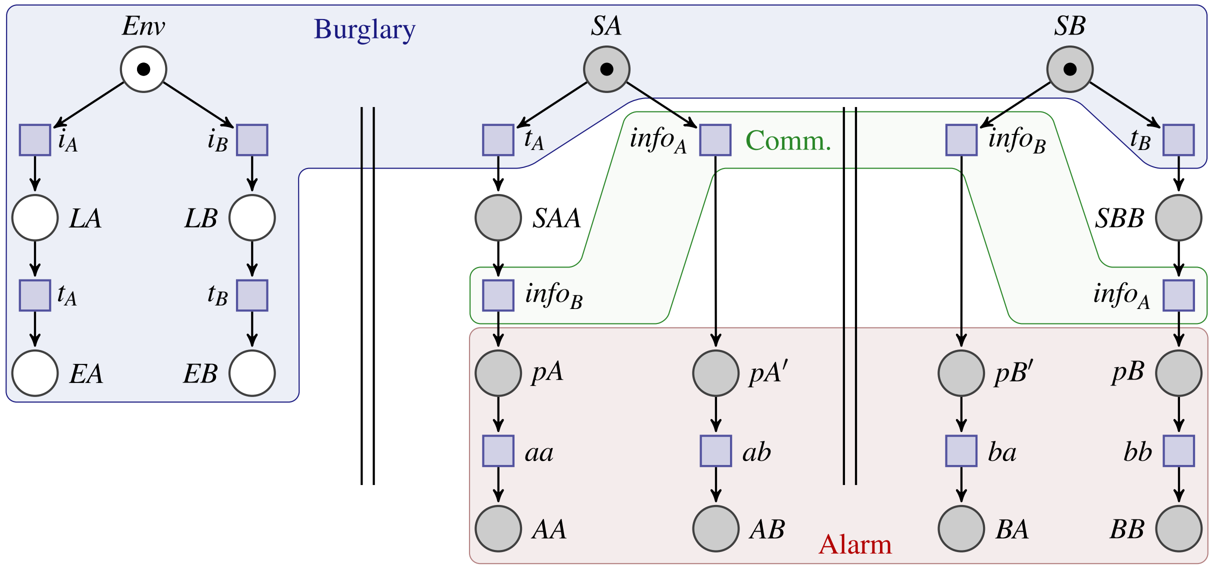


Figure 3: *The distributed local controllers for each player of the winning strategy depicted in Fig. 2. The parallel bars indicate the parallel composition of Petri nets [13] for the environment and the two system controllers, with synchronization on equally labeled transitions. First, the burglary is detected, then the detection is communicated, and afterwards the alarm is set off.*

faithfully inform them about an ongoing burglary, i.e., the system players cooperate. This distribution of a winning strategy into local controllers is possible for all winning strategies of safe, concurrency-preserving Petri games with only one environment player [8].

A *bounded strategy* for a bounded unfolding is generated in the same way as a strategy is generated for the unfolding. Based on the decisions of system players, transitions and the following sub-games are removed. The bounded strategy has to fulfill the same requirements as a strategy for an unfolding but may have fewer system places where transitions can be removed. Places are not unfolded infinitely often in the bounded unfolding, which implies that certain histories are aggregated in one place for which the same decision is repeated. A bounded strategy for a bounded unfolding can easily be extended into a strategy for the general unfolding by repeating the same decisions at places, which were not unfolded in the bounded unfolding. The converse direction is not true, i.e., a strategy for the unfolding cannot in general be translated into a strategy for a bounded unfolding [5].

3 Symbolic and bounded solving

We recall the symbolic solving approach implemented in ADAM and the bounded solving approach implemented as a prototype. Both approaches are compared in Sec. 4. Symbolic solving is based on the reduction of Petri games to a two-player game on finite graphs. This is solved using BDDs in ADAM. The bounded solving approach utilizes two bounds n on the size of the proof and b on the available memory. The question of the existence of a strategy within these bounds is encoded in a 2-QBF. Our prototype automates the generation of the 2-QBF, invokes the QBF solver QUABS to solve it, and generates a winning strategy if the QBF is satisfiable.

3.1 Symbolic Solving

In the symbolic synthesis approach for Petri games introduced in [7], the existence of a winning strategy for the system players is decided via a reduction to a two-player game over a finite graph with complete information. We recap the ideas of the reduction in this section for the comparison to the bounded approach presented in Sec. 3.2. In this paper, we only consider the case of one environment and a bounded number of system players for the symbolic synthesis approach. This case can be solved with a safety objective in single-exponential time [8]. Furthermore, we stick to safe Petri nets, since the implementation of ADAM and the bounded synthesis approach are limited to safe nets.

The general approach for the symbolic solving of Petri games is done in three steps: Firstly, from a given safe Petri game with one environment player, a bounded number of system players, and a safety objective, a two-player game over a finite graph is constructed. The environment player is represented by Player 1 (depicted as white rectangles with sharp corners) and all system players together are represented by Player 0 (depicted as gray rectangles with rounded corners). Secondly, a winning strategy of the two-player game is constructed such that the system players can cooperatively play against all behaviors of the hostile environment without encountering any bad situations. Thirdly, the winning strategy for the system players (Player 0) of the two-player game over a finite graph is transformed into a winning strategy of the system players in the Petri game and distributed into local controllers for each system player.

The two-player game over a finite graph simulates a subset of the behavior of the Petri game in such a way that the game over the graph can be considered as *completely informed*, i.e., both players have full information about their opponent at all times. Even though only a subset of the behavior is considered, [8] shows the existence of a strategy of the Petri game if and only if a strategy for the two-player game exists. Intuitively, the omitted behavior corresponds to situations where the system players exploit knowledge about the environment player's behavior of which they had not been informed. The key idea to achieve complete information is to delay every *environment transition*, i.e., transitions $t \in \mathscr{T}$ with ${}^{\bullet}t \cap \mathscr{P}_E \neq \emptyset$, until every next possible action of the system has to be done directly or indirectly in interaction with the environment (or there is no future interaction with the environment needed at all). Those states of the two-player game where the system players have progressed maximally are called *mcut*s. In an mcut, all system players will be informed of the environment's decision when executing their next step (or they will never be informed of the environment's decision). This idea restricts the proposed solving technique to only one environment player. The states corresponding to an mcut are assigned to the environment (Player 1) and all other states to the system (Player 0).

To simulate the Petri game, the states of the two-player game correspond to *cuts*, i.e., maximal sets of concurrent places. For the sophisticated handling of the causal memory model of Petri games, each place of a cut is enriched by a *commitment set*, i.e., a set of transitions currently selected by the corresponding system player to be allowed to fire. The transition relation of the two-player game mimics the firing of *chosen* (and enabled) transitions of the Petri net between the corresponding cuts.

Additionally, there is one extra kind of transitions in the two-player game allowing the system players to chose new commitment sets. Therefore, each place in a state is equipped with a Boolean flag $\top$. It is set to true for a place p in a state s if and only if s is a successor of an mcut reached by firing transition t and $p \in t^{\bullet}$ holds. Note that it is only harmful for successors of mcuts to directly choose their commitment sets without such an intermediate state with a true $\top$-flag, since for a winning strategy of the system players, *all* successors of the environment states have to avoid bad situations. Thus, the commitment sets of system state successors are directly chosen. The resolving of the $\top$, i.e., choosing new commitment sets, has to be made before any transition is allowed to fire to ensure the correct modeling of the players' informedness. It is therefore guaranteed that every decision of the system, which should be independent of the environment's decision, is actually taken independently.

Since environment decisions are delayed until the system players have maximally progressed, possibly infinite calculations can be encountered when the system players can infinitely proceed without any interaction with the environment. To prevent these infinite behaviors, a further Boolean flag $\texttt{type}_2$ for each place in a state is introduced. This flag set to true prohibits the corresponding player to maximally progress. Thus, in an mcut all non-$\texttt{type}_2$-typed places are blocked until the environment makes its next move due to the non-existence of an enabled and chosen transition and the $\texttt{type}_2$-typed places are blocked by definition. This ensures that the system players cannot pass over the environment's decision by just playing infinitely on their own.

There are three different types of bad situations in the two-player game. Firstly, a state s is bad if two different transitions t_1 and t_2 with ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$ are enabled and chosen in s. Those situations are called *nondeterministic*. Secondly, a state represents a bad situation if it contains bad places. Thirdly, *deadlock*s are bad situations. Deadlocks are states s where a transition exists which is enabled in the corresponding cut of the underlying Petri net, but there is no enabled and chosen transition in s. For more details, we refer the reader to [7, 6, 8].

Example. *We describe the two-player game over a finite graph obtained by the reduction of the Petri game of Fig. 1. In Fig. 4, a part of this game is visualized. Each state s is depicted as tuple* $s \in \mathscr{P}_E \times \mathbb{P}(\mathscr{P}_S \times \{0,1\} \times \{\top, !\top\} \times \mathbb{P}(\mathscr{T}))$, *where* 0 *indicates that the corresponding place is* $\texttt{type}_2$ *typed. In this example, there is no possibility for the system players to play infinitely long without any further interaction with the environment. Thus, all places are typed as not* $\texttt{type}_2$ *(indicated by* 1*).*

The initial state (labeled with 1) corresponds to the cut of the initial marking of the Petri net, where all system tokens initially have to decide on their commitment sets. A commitment set for a place p has to be chosen from the powerset of $p^\bullet$. *Hence, for the initial state the player on SA chooses from* $\mathbb{P}(\{t_A, fa_A, info_A\})$ *and SB from* $\mathbb{P}(\{t_B, fa_B, info_B\})$. *All of these possible combinations yield a successor of the initial state. Here, only four successors are displayed. In general, dashed borders designate that not all successors of a state are depicted in this figure.*

The checkerboard patterned states designate the bad situations of the game. Consider for example the upper left state. There, the token in pA has two possible chosen and enabled transitions (aa and ab). This situation corresponds to a nondeterministic choice in the Petri game. The other two depicted bad states correspond to deadlock situations, since the players in SA and SB decided not to allow to fire any transition, but the underlying Petri net can still fire in the corresponding cut $\{LA, SA, SB\}$ *(e.g., transition* t_A *is enabled). The situations where the game enters a state containing a bad place are not directly visualized, but reaching a bad place cannot be prevented in both situations representing a false alarm (depicted as the left branch in Fig. 4) and a false report (depicted as the right branch). In the depicted case of a false alarm, the alarm system of location A decides to use transition* fa_A *and since the environment is delayed until all system behavior is maximally processed, the alarm system will show a burglary before it has happened (cf. state 2 with a transition* $t_\perp$ *leading to a bad place and* ${}^\bullet t_\perp = \{Env, AA\}$*). Even if the system decides to not use any of the bad transitions, it cannot avoid a bad situation because it will end up in a deadlock (cf. state 3, where reaching a deadlock is mandatory). This is similarly in the depicted case of a false report. Since alarm system B decided to use transition* fr_B *(cf. state 4) and thus does not report the burglary at its location to the alarm system in A, the token in SA triggers a deadlock. If SA would have chosen some of the other possible transitions (*t_A *or* fa_A*), it would still have been a deadlock or a false alarm. The only possible solution for the alarm systems is to wait for the burglary and then use their information channel (*$info_A$ *and* $info_B$*) to inform the other player of the burglary. This situation corresponds to the orange underlaid states resulting in a winning strategy.*

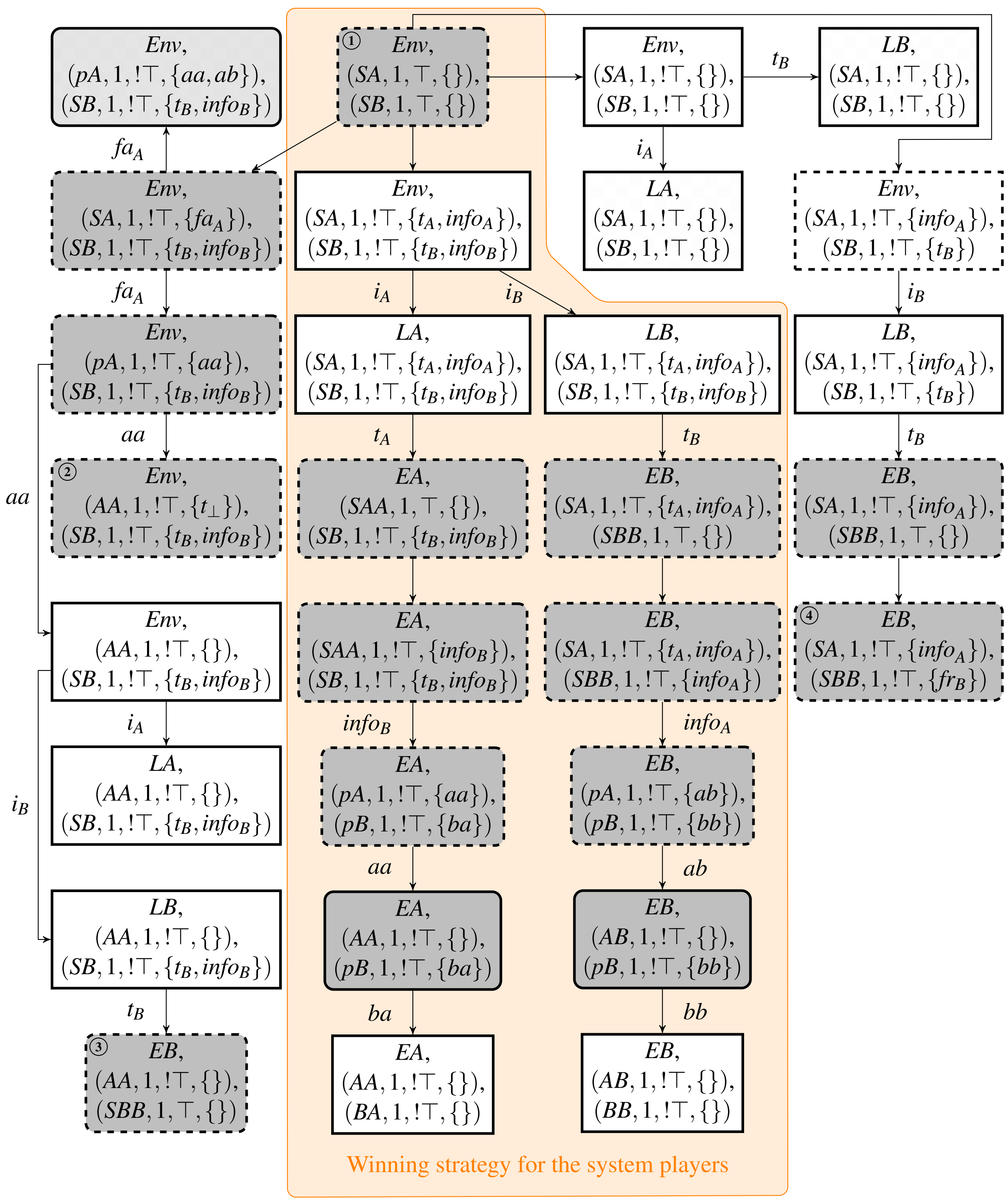

$Env, (pA,1,!\top,\{aa,ab\}), (SB,1,!\top,\{t_B,info_B\})$
fa_A
$Env, (SA,1,!\top,\{fa_A\}), (SB,1,!\top,\{t_B,info_B\})$
fa_A
$Env, (pA,1,!\top,\{aa\}), (SB,1,!\top,\{t_B,info_B\})$
aa
② $Env, (AA,1,!\top,\{t_\perp\}), (SB,1,!\top,\{t_B,info_B\})$
aa
$Env, (AA,1,!\top,\{\}), (SB,1,!\top,\{t_B,info_B\})$
i_A
$LA, (AA,1,!\top,\{\}), (SB,1,!\top,\{t_B,info_B\})$
i_B
$LB, (AA,1,!\top,\{\}), (SB,1,!\top,\{t_B,info_B\})$
t_B
③ $EB, (AA,1,!\top,\{\}), (SBB,1,\top,\{\})$
① $Env, (SA,1,\top,\{\}), (SB,1,\top,\{\})$
$Env, (SA,1,!\top,\{t_A,info_A\}), (SB,1,!\top,\{t_B,info_B\})$
i_A
i_B
$LA, (SA,1,!\top,\{t_A,info_A\}), (SB,1,!\top,\{t_B,info_B\})$
t_A
$EA, (SAA,1,\top,\{\}), (SB,1,!\top,\{t_B,info_B\})$
$EA, (SAA,1,!\top,\{info_B\}), (SB,1,!\top,\{t_B,info_B\})$
$info_B$
$EA, (pA,1,!\top,\{aa\}), (pB,1,!\top,\{ba\})$
aa
$EA, (AA,1,!\top,\{\}), (pB,1,!\top,\{ba\})$
ba
$EA, (AA,1,!\top,\{\}), (BA,1,!\top,\{\})$
$Env, (SA,1,!\top,\{\}), (SB,1,!\top,\{\})$
t_B
i_A
$LA, (SA,1,!\top,\{\}), (SB,1,!\top,\{\})$
$LB, (SA,1,!\top,\{t_A,info_A\}), (SB,1,!\top,\{t_B,info_B\})$
t_B
$EB, (SA,1,!\top,\{t_A,info_A\}), (SBB,1,\top,\{\})$
$EB, (SA,1,!\top,\{t_A,info_A\}), (SBB,1,!\top,\{info_A\})$
$info_A$
$EB, (pA,1,!\top,\{ab\}), (pB,1,!\top,\{bb\})$
ab
$EB, (AB,1,!\top,\{\}), (pB,1,!\top,\{bb\})$
bb
$EB, (AB,1,!\top,\{\}), (BB,1,!\top,\{\})$
$LB, (SA,1,!\top,\{\}), (SB,1,!\top,\{\})$
$Env, (SA,1,!\top,\{info_A\}), (SB,1,!\top,\{t_B\})$
i_B
$LB, (SA,1,!\top,\{info_A\}), (SB,1,!\top,\{t_B\})$
t_B
$EB, (SA,1,!\top,\{info_A\}), (SBB,1,\top,\{\})$
④ $EB, (SA,1,!\top,\{info_A\}), (SBB,1,!\top,\{fr_B\})$
Winning strategy for the system players

Figure 4: *The part of the two-player game over a finite graph constructed from the Petri game depicted in Fig. 1. The states belonging to the environment player are white with sharp corners whereas the states belonging to all system players together are gray with rounded corners. The winning strategy for the system players is underlaid in orange. Dashed states indicate that not all possible successors are depicted. Checkerboard patterned states designate bad states of the two-player game.*

3.2 Bounded Solving

We recall the bounded synthesis approach for Petri games [5]. In bounded synthesis, a bound is introduced to limit the search space of possible winning strategies to small strategies. Therefore, bounded synthesis can find small implementations fast. The bound is increased incrementally if no winning strategy can be found. If a winning strategy for a certain bound is found, bounded synthesis ensures that this solution is winning in general. We denote this bound by n. Bounded synthesis constitutes a semi-decision procedure, i.e., it can prove the existence of a winning strategy but not the non-existence of winning strategies in general. Bounded synthesis finds strategies, which are, because of their small size, interesting for practical applications as they avoid unnecessary (and possibly expensive) computation steps.

In Petri games, each local player can have different information about the other players. Recall that only participating players of a fired transition exchange their complete causal history. A place can have infinitely many different histories, which are represented explicitly in the possibly infinite unfolding. For bounded synthesis of Petri games, we have to introduce a second bound b on the size of the memory for each place in order to retain a finite representation of the bounded synthesis problem. A player residing in the place can only differentiate causal history up to this bound and further history is treated on par with some previous history. The original bound of bounded synthesis n limits the size of the proof of correctness for the strategy. It defines how many transitions are fired until the game has to terminate or has to repeat its behavior in a loop while fulfilling the requirements for a winning strategy. The conditions for a winning strategy are safety, determinism, deadlock-avoidance, and justified refusal as discussed in Sec. 2.

We utilize 2-QBFs to realize the bounded synthesis approach for Petri games. 2-QBFs restrict QBFs to only have one quantifier alternation. A QBF starts with an alternation of either existentially ($\exists$) or universally ($\forall$) quantified sets of Boolean variables. This prefix is followed by the matrix which is a Boolean formula using the standard operators ($\wedge$, $\vee$, $\neg$) and abbreviated operators ($\implies$, $\iff$) on Boolean variables. We focus on 2-QBFs of the form $\exists V_1.\forall V_2.\,\phi$ where $V_1 \cup V_2$ are all Boolean variables in ϕ. The meaning of a 2-QBF is that there exists an assignment for the Boolean variables in V_1 such that for all assignments to the Boolean variables in V_2 the formula ϕ over the assigned Boolean variables is satisfied.

For bounded synthesis of Petri games, the bound b is used to build a bounded unfolding $\mathscr{P}^b$ of the Petri game $\mathscr{P}$. $\mathscr{P}^b$ is again a Petri game explicitly modeling all available decision points for the bounded strategy. For a Petri game, the existence of a winning strategy for a play of length n can be encoded as a 2-QBF $\exists S.\forall M.\,\phi_n$. The set S describes the strategy and contains pairs (p,t) to indicate whether the system place $p \in \mathscr{P}^b_S$ decides to fire the transition $t \in p^\bullet$ or not. We further ensure that a decision for or against t does not violate justified refusal such that all bounded strategies fulfill this condition. The set M describing the sequence of markings contains pairs (p,i) to indicate that on place p resides a token at time point $1 \leq i \leq n$. The formula ϕ_n ensures that if M represents a play following the decisions by the strategy S and the rules of a Petri game for n steps, then the play is winning. This approach can handle finite and infinite plays by accepting termination before the last simulation step is reached and checking for loops if the last simulation step is reached.

The encoding for bounded synthesis has the following form:

$$
\begin{aligned}
\phi_n &\overset{Def.}{=} \left(\bigwedge_{i\in\{1,\dots,n-1\}} sequence_i \implies win_i \right) \wedge (sequence_n \implies win_n \wedge loop)\\
sequence_i &\overset{Def.}{=} initial \wedge \bigwedge_{j\in\{1,\dots,i-1\}} flow_j\\
win_i &\overset{Def.}{=} nobadplaces_i \wedge deterministic_i \wedge deadlocksterm_i\\
deadlocksterm_i &\overset{Def.}{=} deadlock_i \implies terminating_i\\
loop &\overset{Def.}{=} \bigvee_{j,k\in\{1,\dots,n\},j<k} \left(\bigwedge_{p\in\mathscr{P}} (p,j) \iff (p,k) \right)
\end{aligned}
$$

For each time point $1 \leq i \leq n$, it is tested whether the variables in M represent a correct *sequence* of markings corresponding to a play in the Petri game. If this is the case then win_i ensures that the strategy fulfills the requirements at i to be winning. If $i = n$, i.e., the limit on the simulation length is reached, then it is additionally tested that a *loop* occurred. A correct sequence is defined by the play starting from the *initial* marking and firing one enabled and (by the strategy) chosen transition at each time step ($flow_j$). The play is winning if *no bad places* are reached, the system makes only *deterministic* decisions, and each *deadlock* is caused by *termination*. A deadlock occurs when all transitions are either not enabled or not chosen by the strategy. Meanwhile, termination occurs when no transition is enabled, i.e., only the lack of tokens in the preset of transitions is responsible for this and not the decisions of the system. Therefore, $deadlocksterm_i$ ensures that the system does not prevent the reaching of bad places by just stopping to fire transitions but deadlocks are only allowed when the whole game terminated. A loop in a Petri game occurs when the exact marking is repeated at two different time points j and k. Since it is tested that between j and k the strategy is deterministic, this behavior is repeated infinitely often such that the strategy is also winning in an infinite play. For further details on the definition of *initial*, $flow_j$, $deterministic_i$ etc., we refer the interested reader to [5].

Example. *In Fig. 5, a bounded unfolding of the distributed alarm system is given. Only the places pA and pB are unfolded three times resulting in the four respective places pA', pA_A, pA_B and pB', pB_B, pB_A. The unfolded places of pA are on the lefthand side whereas the unfolded places of pB are on the righthand side. For the four places of pA from left to right, we thereby can differentiate the situation that the alarm system in the corresponding wing successfully tested for the environment and then decided not to inform the other system player (place pA), did not test for the environment at all (place pA'), successfully tested for the environment and then informed the other system player (place pA_A), or was informed by the other system player about an intrusion at the other location (place pA_B). The same holds in converse direction for pB. We did not unfold the places AA, AB, BA, and BB because they are used only to determine bad behavior. This is only possible in the* bounded *unfolding.*

We argue why bounded synthesis rejects and accepts certain strategies for the bounded unfolding from Fig. 5. The Petri game is the running example of an alarm system from Fig. 1 in Sec. 2. For example, the strategy (SA,t_A), (SA,fa_A), $(SA,info_A)$ activated and all other transitions deactivated is not winning because for the allowed sequence of markings $(Env,1)$, $(SA,1)$, $(SB,1)$, $(LA,2)$, $(SA,2)$, $(SB,2)$ in M there is nondeterminism between the enabled transitions t_A and fa_A. Meanwhile, the strategy allowing (SA,t_A), $(SA, info_A)$, (SB,t_B), $(SB,info_B)$, $(SAA,info_B)$, $(SBB,info_A)$, (pA_A,a), (pA_B,b), (pB_A,a), (pB_B,b) is winning because for all markings that represent a valid play of the game no bad place is reached, all decisions are deterministic, and all deadlocks are caused by termination. The places pA_A,

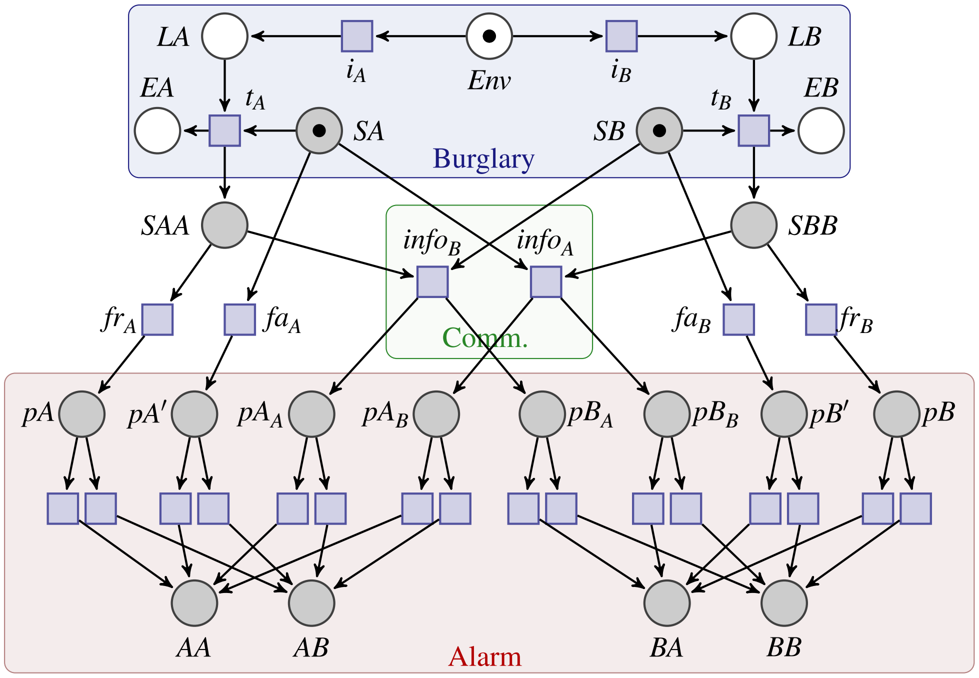


Figure 5: *A bounded unfolding for the Petri game depicted in Fig. 1 modeling an alarm system. The places pA and pB are unfolded three times, respectively, into the places pA', pA_A, pA_B and pB', pB_B, pB_A whereas the places AA, AB, BA, and BB are not unfolded.*

pA_B and pB_A, pB_B describe the unfolded places of pA and pB reached after firing $info_B$ and $info_A$, respectively. For the pairs of unlabeled transitions, the left transitions are based on the original transition a and the right ones on b. The places pA, pA', pB and pB' are not reached by the strategy and arbitrary decisions can be made there. We choose not to fire any transitions in this case. For example, the sequence of markings $(Env,1),(SA,1),(SB,1),(LA,2),(LB,2),\ldots$ in M does not represent a valid play because both outgoing transitions of Env have been fired, which is illegal in a Petri game ($flow_1$ is violated).

4 Experimental Results

We compare the implementation of the symbolic approach in the tool ADAM against our prototype implementation of the bounded synthesis approach on an extended set of benchmarks. We take the benchmark set of ADAM and add the benchmark family of an alarm system. At first, we describe all benchmark families. Then, we outline the technical details of our comparison framework and implementation specific particularities of the two approaches. Afterwards, we state our observations and explanations concerning the times for finding winning strategies and the sizes of these strategies.

4.1 Benchmark families

The results in the table from Fig. 6 refer to the following scalable benchmark families where the alarm system is the new benchmark family:

- AS: *Alarm System*. There are *m* secured locations belonging to one person. A burglar can intrude one of the locations. Each location has a local alarm system, which can communicate with all

other local alarm systems. The goal is that the alarm system in each location has to indicate the position where the burglar intruded. Furthermore, the alarm system should not issue unsubstantiated warnings of an intrusion at any location.
Parameters: m locations

- CM: *Concurrent Machines*. There are m machines which should process k orders. Each machine is allowed to process at most one order. The hostile environment disables one arbitrary machine such that it cannot process any order. The system's goal is to still process all k orders.
  Parameters: m machines and k orders.
- SR: *Self-reconfiguring Robots*. There are m robots having m different tools at their disposal each. The robots can only equip one tool at a time. From a global perspective, all robots together have to maintain a functioning state such that material can be processed by each of the m different tools. The environment can destroy k tools in total. This can occur at the same robot or on different robots. The robots have to reconfigure theirselves to maintain a functioning global state for the processing of material.
  Parameters: m robots with m tools each and k destroyed tools in total.
- JP: *Job Processing*. A job requires processing by a, from the environment chosen, subset of m processors in ascending order.
  Parameter: m processors.
- DW: *Document Workflow*. There are m clerks having to unanimously endorse or reject a document. The environment decides which clerk gets the document first. In DWs, it is required that all clerks endorse the document.
  Parameter: m clerks.

4.2 Comparison framework

We compare the symbolic synthesis approach implemented in ADAM with our prototype implementation of the bounded synthesis approach. ADAM and the bounded synthesis approach are the only tools existing to find winning strategies of Petri games but they are inherently different. On the one hand, the symbolic approach models, in theory, infinite memory and unbounded firing sequences of transitions. On the other hand, bounded synthesis has two parameters n and b, which can be increased to find a winning strategy. Therefore, the bounded synthesis approach can be parallelized easily because the QBF-solver can be called twice for two different pairs (n,b) and the resulting encodings. We report in the following on the runtime results for the minimal b and the corresponding n such that a winning strategy exists because b turned out to be more expensive than n in terms of runtime. Notice that $b = 1$ enforces that the bounded unfolding is the original game, i.e., no bounded unfolding is utilized when searching for a winning bounded strategy of the corresponding Petri game.

The table in Fig. 6 shows the results of ADAM and our prototype implementation of bounded synthesis for the previously described benchmark families. The results were obtained on an Intel i7-2700K CPU with 3.50 GHz, 32 GB RAM, and a timeout of 1800 seconds. For each benchmark (column *Ben.*), we report on the attempted parameters of the benchmark (*Par.*), on the size of the Petri game (number of tokens (#*Tok*), places (#$\mathscr{P}$), and transitions (#$\mathscr{T}$)), and on the respective *time* and *memory* for solving and on the respective number of places (#$\mathscr{P}_{str}$) and transitions (#$\mathscr{T}_{str}$) of the winning strategies synthesized by ADAM and by our prototype implementation of bounded synthesis. The elapsed CPU time is measured in seconds and the used memory in gigabyte. For bounded synthesis, we additionally report the smallest b and corresponding n to find winning bounded strategies with the least memory requirement.

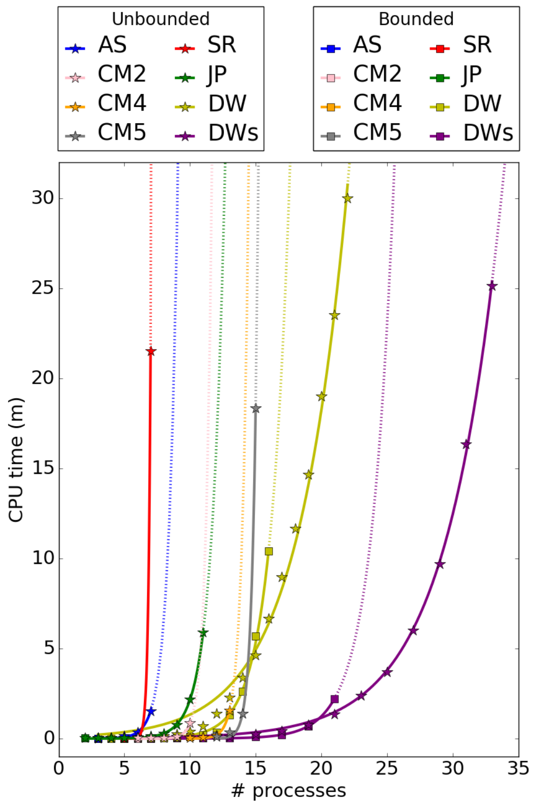
					Symbolic Synthesis						*Bounded Synthesis*			
Ben.	*Par.*	*#Tok*	$\#\mathscr{P}$	$\#\mathscr{T}$	*time*	*memory*	$\#\mathscr{P}_{str}$	$\#\mathscr{T}_{str}$	*n*	*b*	*time*	*memory*	$\#\mathscr{P}_{str}$	$\#\mathscr{T}_{str}$
AS	2	3	17	26	**1.9**	.30	**17**	**10**	7	2	18.0	**.18**	**17**	**10**
	3	4	28	69	2.5	.41	31	18	7	3	timeout			
			...			...			...					
	6	7	73	462	91.0	4.65	97	54	7	6	timeout			
	7				timeout				7	7	timeout			
CM	2/1	6	13	10	1.4	.30	14	**8**	6	3	**.6**	**.06**	**13**	**8**
	2/2	7	18	16	2.0	.30	-	-	-	-				
			...			...			...					
	2/5	10	33	34	50.9	2.74	-	-	-	-				
	2/6				timeout				-	-				
	3/1	8	18	15	2.0	.30	26	12	6	3	**1.7**	**.12**	**18**	**9**
	3/2	9	25	24	2.4	.30	36	18	6	4	timeout			
	3/3	10	32	33	3.8	.39	-	-	-	-				
	3/4	11	39	42	17.2	1.28	-	-	-	-				
	3/5				timeout									
	4/1	10	23	20	**2.3**	.30	42	16	6	3	6.0	**.19**	**21**	**12**
	4/2	11	32	32	5.0	.40	55	24	6	4	timeout			
	4/3	12	41	44	10.9	.84	68	32	6	5	timeout			
	4/4	13	50	56	92.2	4.17	-	-	-	-				
	4/5				out of memory				-	-				
	5/1	12	28	25	**7.2**	.39	62	20	6	3	11.1	**.19**	**22**	**11**
	5/2	13	39	40	20.8	.79	78	30	6	4	timeout			
	5/3	14	50	55	82.1	2.67	94	40	6	5	timeout			
	5/4	15	61	70	1101.3	16.70	110	50	6	6	timeout			
	5/5				out of memory				-	-				
	6/1	14	33	30	41.5	.80	86	24	6	3	**23.6**	**.31**	**25**	**12**
	6/2	15	46	48	183.4	2.67	105	36	6	4	timeout			
	7/1	16	38	35	289.5	5.35	114	28	6	3	**26.0**	**.36**	**27**	**13**
	8/1	18	43	40	1657.4	15.73	146	32	6	3	**94.7**	**.65**	**27**	**14**
	9/1	20	48	45	out of memory				6	3	152.4	1.22	36	25
			...			...			...					
	15/1	32	78	75	out of memory				6	3	1259.5	23.24	66	49
SR	2/1	5	18	17	**1.9**	.30	32	16	6	2	2.7	**.16**	**18**	**10**
	2/2	6	24	26	4.3	.39	-	-	-	-				
	2/3	7	30	35	1290.3	5.36	-	-	-	-				
	2/4				out of memory				-	-				
	3/1				out of memory				7	2	219.4	.61	27	19
JP	2	3	12	13	**1.3**	.30	**16**	**13**	7	3	1.5	**.08**	**16**	**13**
	3	4	18	23	1.8	.30	34	28	8	4	timeout			
			...			...			...					
	11	12	102	175	353.3	16.78	706	484	16	12	timeout			
	12				out of memory				17	13	timeout			
DW	1	3	12	10	1.1	.30	**10**	**6**	8	1	**.3**	**.04**	**10**	**6**
	2	4	19	16	2.0	.30	24	16	10	1	**.4**	**.05**	**16**	**12**
			...			...			...					
	11	13	82	70	137.6	2.82	420	286	28	1	**78.3**	**2.26**	**70**	**57**
	12	14	89	76	201.8	**2.82**	494	336	30	1	**157.3**	3.38	**76**	**62**
	13	15	96	82	**277.9**	**4.24**	574	390	32	1	341.1	8.18	**82**	**67**
	14	16	103	88	**400.1**	**4.22**	660	448	34	1	624.5	11.80	**88**	**72**
	15	17	110	94	537.2	4.87	752	510	36	1	timeout			
			...			...			...					
	20	22	145	124	1799.8	11.69	1302	880	46	1	timeout			
	21				timeout				48	1	timeout			
DWs	1	3	11	6	.7	.29	**8**	**3**	5	1	**.2**	**.04**	**8**	**3**
	2	5	21	12	1.6	.30	23	10	7	1	**.3**	**.05**	**17**	**8**
			...			...			...					
	7	15	71	42	14.4	.91	218	105	17	1	**5.3**	**.87**	**57**	**28**
	8	17	81	48	24.9	**1.52**	281	136	19	1	**11.7**	3.18	**65**	**32**
	9	19	91	54	45.4	**2.83**	352	171	21	1	**41.3**	8.84	**73**	**36**
	10	21	101	60	**80.0**	**2.85**	431	210	23	1	132.8	12.96	**81**	**40**
	11	23	111	66	142.6	4.42	518	253	25	1	timeout			
			...			...			...					
	16	33	161	96	1508.3	16.30	1073	528	35	1	timeout			
	17				timeout				37	1	timeout			

'-' means no winning strategy exists.

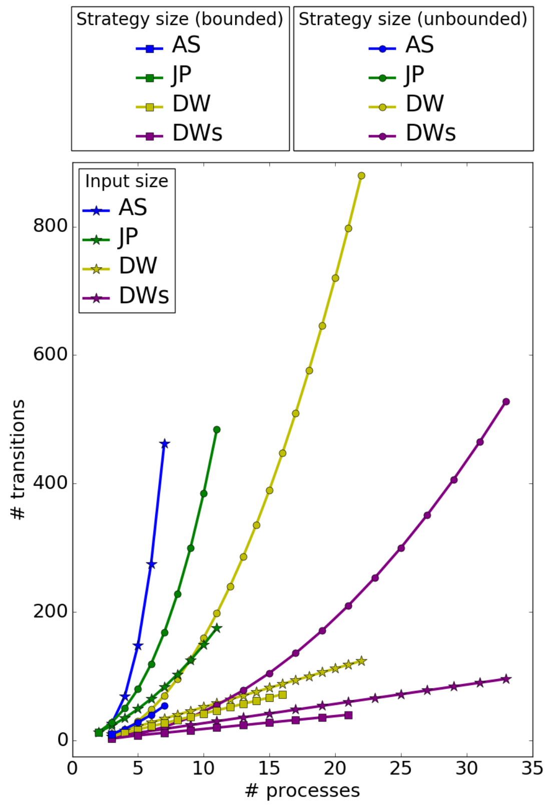
Figure 6: Comparison between the symbolic synthesis approach and the bounded synthesis approach.

When ADAM and our prototype implementation both synthesized a winning strategy, then we mark the minimal running time, the minimal memory usage, and the minimal number of places and of transitions in the winning strategy in bold, respectively.

A selection of these values and benchmark families are plotted in Fig. 7. In Fig. 7(a), the CPU times in minutes for the symbolic and the bounded approach on selected benchmark families are plotted for an increasing number of processes, i.e., the number of tokens of the underlying net. *CMi*, for $i \in \{2,4,5\}$, represents the subset of the concurrent machines benchmark, where the first parameter m for the number of machines is fixed to i. Therefore, the number of orders increases the number of processes in *CMi*. The dotted lines designate the expected running time obtained by fitting exponential curves through the actual values. In Fig. 7(b), we plotted the number of transitions of the input Petri games and of the corresponding winning strategies of the two approaches for an increasing number of processes from selected benchmark families.



(a) The CPU running time (in minutes) for a selection of benchmarks and the respective number of processes (tokens). The running time for the bounded approach is given as squares and the running time for the symbolic case is given by stars. The dotted lines designate the expected running time after the timeout of 30 minutes.



(b) The sizes (in the number of transitions) for a selection of benchmarks and the respective number of processes (tokens). The number of transitions of the Petri game is designated by stars, the number of transitions for the bounded strategy by filled circles, and the number of transitions of the strategy in the bounded case by squares.

Figure 7: Comparison of the symbolic synthesis approach and the bounded synthesis approach by the running times and sizes of the strategies.

4.3 Implementation details

During the implementation of our prototype for bounded synthesis of Petri games, we observed that a translation of the matrix ϕ_n into conjunctive normalform (CNF) and the usage of a QBF solver requiring input in CNF has poor performance. We found out that the `QCIR` file format for QBFs [16] allows competitive performance as ϕ_n does not need to be translated into CNF. As we have a 2-QBF not in CNF, solvers using counterexamples to refine an abstraction (CEGAR-based) [3] showed the best performance. We therefore decided for the solver QuAbS [17] as it combines fast parsing with fast solving. Given a bound on the length of the proof of correctness for a strategy, we further pruned the bounded unfolding of unreachable places and unreachable transitions to remove unnecessary variables from the 2-QBF, which increased the overall performance.

The running times of both approaches are highly dependent on the number of variables which are respectively used in the BBD and in the QBF. For the symbolic approach, the number of variables in the BDD grows significantly with the number of players in the Petri game (which are represented by tokens in the underlying net). The two-player game over a finite graph with complete information is represented by a BDD for each flow encoding the flow's source and target state. For optimizing the size of the BDD, we partition $\mathscr{P}_S$ into k disjunct sets $\mathscr{P}_{S_i} \subseteq \mathscr{P}_S$, for $i \in \{1,\dots,k\}$, such that for every reachable marking each place of the marking belongs uniquely to exactly one of the sets $\mathscr{P}_{S_i}$. In general, the number k corresponds to the number of system processes in the game. A state of the finite graph is encoded by a binary encoding of the ID of the environment place and the maximally k IDs of the system places of the marking, which results in encodings of logarithmic size. Whether a process's commitment set contains a transition is encoded explicitly by a Boolean flag. An explicit encoding is used because the size of the commitment set varies for each process depending on the place it resides in. In general, this approach yields smaller BDDs. The two additional variables per system process encode whether the process's commitment set has to be renewed, i.e., the $\top$-flag is set or not, and whether the place is a $\texttt{type}_2$-typed place. Thus, the number of variables for a BDD can be calculated by

$$2 \cdot \left(\texttt{log}_2(|\mathscr{P}_E|) + \sum_{i=1}^{k} (\texttt{log}_2(|\mathscr{P}_{S_i}|) + |\mathscr{T}_i| + 2) \right)$$

where $\mathscr{T}_i = \bigcup_{p \in \mathscr{P}_{S_i}} p^{\bullet}$.

For the bounded approach, we distinguish existentially and universally quantified variables and gate variables. The existentially quantified variables describe the system's strategy and the universally quantified variables encode all sequences of markings. The gate variables are used to describe the bounded synthesis problem in the matrix ϕ_n given the existentially and universally quantified variables. Thus, the size of the QBF is growing with the structure of the underlying net of the Petri game, i.e., with the number of places and transitions.

In Fig. 8, we compare the number of variables in the two approaches for benchmark families where several instances are solved by both approaches. DW, DWs, and CM with parameter $k = 1$ qualify for this comparison. For the symbolic approach, the number of variables in the BDD is given ($\#Var_{symbolic}$). For the bounded approach, the total number of variables in the QBF ($\#Var_{bounded}$) and the number of existentially ($\#Var_{\exists}$) and universally ($\#Var_{\forall}$) quantified variables thereof are given. The number of gate variables ($\#Var_{\phi_n}$) used to build the matrix ϕ_n is stated as the remaining variables of $\#Var_{bounded}$, i.e., $\#Var_{\exists} + \#Var_{\forall} + \#Var_{\phi_n} = \#Var_{bounded}$. From the size difference of the respective numbers of variables in the two approaches, one can derive that they are used for a different purpose in the respective approach. For the symbolic approach, the number of variables grows linearly for all benchmarks. For the

Ben.	*Par.*	$\#Var_{symbolic}$	n	b	$\#Var_{bounded}$	$\#Var_{\exists}$	$\#Var_{\forall}$	$\#Var_{\phi_n}$
CM	2/1	66	6	3	2743	53	162	2528
	3/1	92	6	3	7678	109	300	7269
	4/1	120	6	3	17849	197	462	17190
	5/1	146	6	3	25848	266	570	25012
	6/1	172	6	3	35287	343	678	34266
	7/1	198	6	3	46166	428	786	44952
	8/1	226	6	3	58485	521	894	57070
DW	1	46	8	1	1144	12	96	1036
	2	72	10	1	2591	20	190	2381
	3	98	12	1	4838	28	312	4498
	4	124	14	1	8052	36	462	7554
	5	148	16	1	12404	44	640	11720
	6	172	18	1	18059	52	846	17161
	7	198	20	1	25186	60	1080	24046
	8	224	22	1	33953	68	1342	32543
	9	248	24	1	44528	76	1632	42820
	10	272	26	1	57079	84	1950	55045
	11	296	28	1	71744	92	2296	69356
	12	320	30	1	88781	100	2670	86011
	13	344	32	1	108268	108	3072	105088
	14	368	34	1	130403	116	3502	126785
DWs	1	36	5	1	440	9	55	376
	2	70	7	1	1414	17	147	1250
	3	102	9	1	3204	25	279	2900
	4	136	11	1	6050	33	451	5566
	5	168	13	1	10192	41	663	9488
	6	200	15	1	15870	49	915	14906
	7	232	17	1	23324	57	1207	22060
	8	266	19	1	32794	65	1539	31190
	9	298	21	1	44520	73	1911	42536
	10	330	23	1	58742	81	2323	56338

Figure 8: Comparison between the numbers of different variables in the two approaches.

bounded approach, the number of existentially quantified variables grows linearly, the number of universally quantified variables grows quadratically, and the total number of variables grows cubicly in DW and DWs. For CM, all variables in the bounded encoding grow exponentially which is surprising as n and b remain constant. We suppose that this increase is caused by the construction of the bounded unfolding which produces an exponentially growing number of transitions for this benchmark family despite $b = 3$ remaining constant.

We further detected that the QBF problem files can become large, which requires a QBF solver with fast parsing of the problem file. The largest solved `QCIR` file is of size 40 MB and contains 208.877 variables (benchmark *concurrent machines* with parameters $m = 15$ and $k = 1$ and bounds $n = 6$ and $b = 3$). A very large `QCIR` file for a benchmark which ADAM solved but for which the QBF solver timed out (benchmark *job processing* with parameter $m = 4$ and bounds $n = 9$ and $b = 5$) has a size of 275 MB and contains 2.722.512 variables.

In summary, the size of the BDD for the symbolic approach is dominated by the number of tokens whereas the size of the 2-QBF for the bounded approach is dominated by the number of places and transitions of the underlying net of the Petri game.

4.4 Comparison

Both approaches work especially well on certain aspects of distributed synthesis. The symbolic approach implemented in ADAM solves more instances than the bounded synthesis approach for all benchmark families but for *concurrent machines* (CM) with one defective machine ($k = 1$). ADAM can further show the non-existence of a winning strategy for instantiations of benchmark families for which bounded synthesis is not applicable. For example, ADAM shows that for CM no strategy exists when equally many or more orders as machines are placed because one machine can process at most one order and the environment marks one machine as unable to process an order.

The bounded approach is well suited for finding small winning strategies. It holds for all winning strategies produced by the two approaches that the respective winning strategies from bounded synthesis have equally many or fewer places and transitions. For small instances, these differences are negligible as for the first instances of *alarm system* (AS) and both versions of *document workflow* (DW and DWs) the respective strategies are of equal size. The larger the benchmark instances become, the larger the differences in strategy size get. For DW with parameter $m = 14$, the strategy from ADAM has 660 places and 448 transitions whereas the strategy from the prototype implementation of bounded synthesis has only 88 places and 72 transition. This comes at a higher solving time of 625 seconds in contrast to 400 seconds and at using 12 GB of memory in comparison to 4 GB.

The difference in size of the strategies can also be observed from Fig. 7(b) where the size of the input and of the produced strategies by the two approaches are compared for a given number of processes. This difference in size is based on the different structure of the strategies in the two approaches. In bounded unfoldings and bounded strategies, more than one transition can merge into one place, if the different history of the token is not needed. In contrast, the symbolic approach has to unfold a place in every case notwithstanding the need for differentiation of its causal past. This becomes apparent in the benchmarks DW and DWs. In the symbolic case, for every choice of the environment which clerk has to decide on endorsing the document first, the places and transitions of each clerk are copied and put into the right order. The bounded algorithm detects that it is not necessary to unfold all these places, since the memory is not needed for finding a winning strategy and thus yields a much smaller strategy.

The bounded approach can be more subtle in choosing when to unfold places and therewith generally finds smaller strategies than the symbolic approach. This illustrates the difference between a bounded strategy (produced by the bounded synthesis approach) and a strategy (produced by the symbolic synthesis approach). Bounded strategies are based on the bounded unfolding which can consolidate different causal pasts into one system place for which the corresponding strategy has to make the same decision. In contrast, a strategy is based on the unfolding, which explicitly represents every causal past of a system place. At each such system place, an individual decision can be made. Therefore, when the same decision suffices for each causal past, these decisions are represented explicitly with each unfolded place. From the visualization of Fig. 7(b), we can conjecture for the displayed subset of benchmark families that the symbolic approach can only find strategies which grow in size exponentially because the unfolding is exponential in the number of places and transitions. In contrast, the bounded approach can find strategies which grow in size linearly when a linearly growing bounded unfolding suffices to represent the necessary causal history.

For the sizes of the solution in the symbolic case, we can see that in general the strategy sizes increase faster and also are larger than the sizes of the input in the benchmark families from Fig. 7(b). This is caused by our benchmarks mostly increasing linearly in the input sizes (e.g., by adding a new robot or a new machine). Meanwhile, the solution is getting more difficult due to the additional abilities and behaviors of the whole system (e.g., the factory) and the symbolic approach has to consider more different flows of tokens with different causal histories. In general, the unfolding and the strategy increase in size stronger than the input. One exception to the linear increase of the input is the new *alarm system* benchmark. There we also add only linearly bounded many places and transitions for every new alarm system, but we have to add an additional alarm signal at each already existing alarm system and, furthermore, transitions leading to bad places for all additional combinations of bad situations. Since those transitions are not present in the strategy, the size of the solution is smaller than the size of the input for the alarm system benchmark.

For the running time, we can see in Fig. 6 and in Fig. 7(a) that the bounded approach outperforms the symbolic one for smaller instances but increases more sharply. This stems from the different parameters in both approaches discussed in Sec. 4.3, which are responsible for the solving complexity. For the symbolic case, the number of processes are principally responsible for the increasing number of variables of the BDD and thus for the complexity. For the bounded approach, this is different because the number of QBF variables is more dependent on the structure of the net than on the number of tokens.

For the benchmarks of DW and DWs, the bounded synthesis approach outperforms ADAM for the first eleven respectively nine parameters in terms of runtime and memory usage. On the next three parameters (DW) respectively on the next parameter (DW), ADAM outperforms bounded synthesis. After that, bounded synthesis already reaches the time limit while ADAM can solve further five parameters for DW and DWs each.

The bounded synthesis approach further showed that no unfolding is necessary to solve instances of DW and DWs. It also revealed that for CM it is possible to find winning strategies for benchmark instances of growing size while maintaining the same values for n and b.

5 Conclusion

We added the new benchmark family of a distributed alarm system to the set of benchmark families for distributed synthesis with Petri games collected during the implementation of ADAM. The new benchmark family models an alarm system for a person with a scalable number of independent locations she needs to secure. Each of these locations has a local alarm system, which can detect the intrusion by a burglar. Furthermore, all alarm systems can communicate with each other and each alarm system can indicate the position of a detected burglary. We synthesized strategies to detect the position of a burglary and indicate it at the alarm systems of *all* independent locations.

We found out that the translation of bounded synthesis into 2-QBF resulted in large non-CNF formulas, which were solved best by a CEGAR-based QBF solver like QUABS. The automatic construction of the bounded unfolding benefits from a removal of unreachable places and transitions, which implies that there is still room for improvement when constructing the bounded unfolding.

We compared the symbolic synthesis approach implemented in the tool ADAM with the bounded synthesis approach on the extended set of benchmarks. We found out that symbolic synthesis can overall synthesize strategies for larger problems for all benchmark families except the benchmark family of concurrent machines (with parameter $k = 1$). For the smaller instances, bounded synthesis is faster but the running time grows at a higher rate such that ADAM can solve more instances overall. At the same time, bounded synthesis finds smaller strategies in the number of places and transitions. This difference is negligible for small instances but grows for larger instances. The difference in size of the strategies is caused by the distinction between the bounded unfolding and the unfolding. In the bounded unfolding, different causal pasts can be consolidated into one place whereas in the unfolding, unique causal pasts have to be differentiated. Therefore, bounded strategies can profit in terms of size from situations where only some causal past is needed. This adds to the general benefit of bounded synthesis in comparison with symbolic synthesis to steer the search to small strategies.

We identified that the number of variables in the BDD of the symbolic approach grows in the number of tokens in the underlying net of the Petri game whereas the number of variables in the QBF of the bounded approach grows in the number of places and transitions of the underlying net of the Petri game. We further showed that for the benchmark families DW and DWs local strategies for each system player suffice as the bounded unfolding is the same as the original game. This proved that both symbolic and bounded synthesis are well-suited for certain aspects of distributed synthesis with Petri games.

References

[1] Roderick Bloem, Stefan J. Galler, Barbara Jobstmann, Nir Piterman, Amir Pnueli & Martin Weiglhofer (2007): *Interactive presentation: Automatic hardware synthesis from specifications: a case study*. In: *2007 Design, Automation and Test in Europe Conference and Exposition, DATE 2007, Nice, France, April 16-20, 2007*, pp. 1188–1193, doi:10.1145/1266366.1266622.

[2] Alonzo Church (1963): *Application of recursive arithmetic to the problem of circuit synthesis*.

[3] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu & Helmut Veith (2000): *Counterexample-Guided Abstraction Refinement*. In: *Computer Aided Verification, 12th International Conference, CAV 2000, Chicago, IL, USA, July 15-19, 2000, Proceedings*, pp. 154–169, doi:10.1007/10722167_15.

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

[5] Bernd Finkbeiner (2015): *Bounded Synthesis for Petri Games*. In: *Correct System Design - Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday, Oldenburg, Germany, September 8-9, 2015. Proceedings*, pp. 223–237, doi:10.1007/978-3-319-23506-6_15.

[6] Bernd Finkbeiner, Manuel Gieseking & Ernst-Rüdiger Olderog (2015): *Adam: Causality-Based Synthesis of Distributed Systems*. In: *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, pp. 433–439, doi:10.1007/978-3-319-21690-4_-25.

[7] Bernd Finkbeiner & Ernst-Rüdiger Olderog (2014): *Petri Games: Synthesis of Distributed Systems with Causal Memory*. In: *Proceedings Fifth International Symposium on Games, Automata, Log-*

ics and Formal Verification, GandALF 2014, Verona, Italy, September 10-12, 2014., pp. 217–230, doi:10.4204/EPTCS.161.19.

[8] Bernd Finkbeiner & Ernst-Rüdiger Olderog (2017): *Petri games: Synthesis of distributed systems with causal memory*. *Inf. Comput.* 253, pp. 181–203, doi:10.1016/j.ic.2016.07.006.

[9] Bernd Finkbeiner & Sven Schewe (2005): *Uniform Distributed Synthesis*. In: *20th IEEE Symposium on Logic in Computer Science (LICS 2005), 26-29 June 2005, Chicago, IL, USA, Proceedings*, pp. 321–330, doi:10.1109/LICS.2005.53.

[10] Bernd Finkbeiner & Sven Schewe (2013): *Bounded synthesis*. *STTT* 15(5-6), pp. 519–539, doi:10.1007/s10009-012-0228-z.

[11] Orna Kupferman & Moshe Y. Vardi (2001): *Synthesizing Distributed Systems*. In: *16th Annual IEEE Symposium on Logic in Computer Science, Boston, Massachusetts, USA, June 16-19, 2001, Proceedings*, pp. 389–398, doi:10.1109/LICS.2001.932514.

[12] P. Madhusudan, P. S. Thiagarajan & Shaofa Yang (2005): *The MSO Theory of Connectedly Communicating Processes*. In: *FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science, 25th International Conference, Hyderabad, India, December 15-18, 2005, Proceedings*, pp. 201–212, doi:10.1007/11590156_16.

[13] Ernst-Rüdiger Olderog (1991): *Nets, Terms and Formulas: Three Views of Concurrent Processes and Their Relationship*. Cambridge University Press, doi:10.1017/CBO9780511526589.

[14] Amir Pnueli & Roni Rosner (1989): *On the Synthesis of a Reactive Module*. In: *Conference Record of the Sixteenth Annual ACM Symposium on Principles of Programming Languages, Austin, Texas, USA, January 11-13, 1989*, pp. 179–190, doi:10.1145/75277.75293.

[15] Amir Pnueli & Roni Rosner (1990): *Distributed Reactive Systems Are Hard to Synthesize*. In: *31st Annual Symposium on Foundations of Computer Science, St. Louis, Missouri, USA, October 22-24, 1990, Volume II*, pp. 746–757, doi:10.1109/FSCS.1990.89597.

[16] QBF Gallery 2014: *QCIR-G14: A Non-Prenex Non-CNF Format for Quantified Boolean Formulas*. Available at `http://qbf.satisfiability.org/gallery/qcir-gallery14.pdf`.

[17] Leander Tentrup (2016): *Non-prenex QBF Solving Using Abstraction*. In: *Proceedings of SAT*, *LNCS* 9710, Springer, pp. 393–401, doi:10.1007/978-3-319-40970-2_24.

[18] Wieslaw Zielonka (1987): *Notes on Finite Asynchronous Automata*. *ITA* 21(2), pp. 99–135, doi:10.1051/ita/1987210200991.