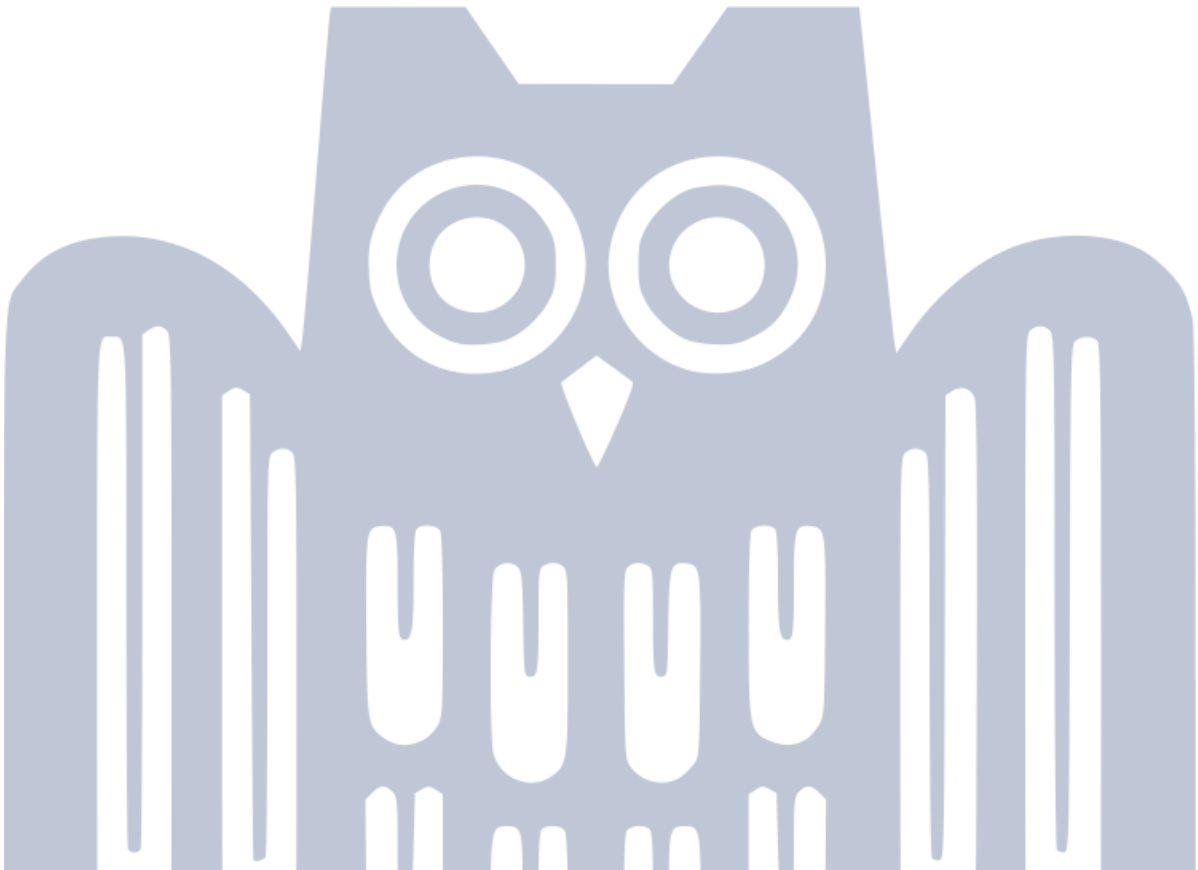
Synthesis of Asynchronous Distributed Systems from Global Specifications

A dissertation submitted towards the degree
Doctor of Natural Sciences (Dr. rer. nat.)
of the Faculty of Mathematics and Computer Science
of Saarland University

by
Jesko Hecking-Harbusch

Saarbrücken, 2021



Date of the colloquium:	December 10th, 2021
Dean of the faculty:	Prof. Dr. Thomas Schuster
Chair of the committee:	Prof. Dr. Jan Reineke
Examination board:	Prof. Bernd Finkbeiner, Ph.D. Prof. Dr. Ernst-Rüdiger Olderog Dr. Rayna Dimitrova
Academic assistant:	Dr. Dominic Steinhöfel

Abstract

The synthesis problem asks whether there exists an implementation for a given formal specification and derives such an implementation if it exists. This approach enables engineers to think on a more abstract level about what a system should achieve instead of how it should accomplish its goal. The synthesis problem is often represented by a game between system players and environment players. Petri games define the synthesis problem for asynchronous distributed systems with causal memory. So far, decidability results for Petri games are mainly obtained for local winning conditions, which is limiting as global properties like mutual exclusion cannot be expressed.

In this thesis, we make two contributions. First, we present decidability and undecidability results for Petri games with global winning conditions. The global safety winning condition of bad markings defines markings that the players have to avoid. We prove that the existence of a winning strategy for the system players in Petri games with a bounded number of system players, at most one environment player, and bad markings is decidable. The global liveness winning condition of good markings defines markings that the players have to reach. We prove that the existence of a winning strategy for the system players in Petri games with at least two system players, at least three environment players, and good markings is undecidable.

Second, we present semi-decision procedures to find winning strategies for the system players in Petri games with global winning conditions and without restrictions on the distribution of players. The distributed nature of Petri games is employed by proposing encodings with true concurrency. We implement the semi-decision procedures in a corresponding tool.

Zusammenfassung

Das Syntheseproblem stellt die Frage, ob eine Implementierung für eine Spezifikation existiert, und generiert eine solche Implementierung, falls sie existiert. Diese Vorgehensweise erlaubt es Programmierenden sich mehr darauf zu konzentrieren, was ein System erreichen soll, und weniger darauf, wie die Spezifikation erfüllt werden soll. Das Syntheseproblem wird oft als Spiel zwischen einem System- und einem Umgebungsspieler dargestellt. Petri-Spiele definieren das Syntheseproblem für asynchrone verteilte Systeme mit kausalem Speicher. Bisher wurden Resultate bezüglich der Entscheidbarkeit von Petri-Spiele meist für lokale Gewinnbedingungen gefunden.

In dieser Arbeit präsentieren wir zuerst Resultate bezüglich der Entscheidbarkeit und Unentscheidbarkeit von Petri-Spielen mit globalen Gewinnbedingungen. Wir beweisen, dass die Existenz einer gewinnenden Strategie für die Systemspieler in Petri-Spielen mit einer beschränkten Anzahl an Systemspielern, höchstens einem Umgebungsspieler und schlechten Markierungen entscheidbar ist. Wir beweisen ebenfalls, dass die Existenz einer gewinnenden Strategie für die Systemspieler in Petri-Spielen mit mindestens zwei Systemspielern, mindestens drei Umgebungsspielern und guten Markierungen unentscheidbar ist.

Danach präsentieren wir Semi-Entscheidungsprozeduren, um gewinnende Strategien für die Systemspieler in Petri-Spielen mit globalen Gewinnbedingungen und ohne Restriktionen für die Verteilung von Spielern zu finden. Wir benutzen die verteilte Natur von Petri-Spielen, indem wir Enkodierungen einführen, die Nebenläufigkeit ausnutzen. Die Semi-Entscheidungsprozeduren sind in einem entsprechenden Tool implementiert.

Acknowledgements

I am very grateful to Bernd Finkbeiner for introducing me to the beautiful world of Petri games and for allowing me to explore this topic. Notably, I am thankful for the aspiring guidance, invaluable constructive criticism, and friendly advice. I sincerely appreciate the many and illuminating views on several issues related to Petri games.

I want to thank Manuel Gieseking for the extraordinary collaboration and the many exciting and fruitful discussions. This collaboration not only helped tremendously with this thesis but allowed me to grow personally. I want to thank Ernst-Rüdiger Olderog for his helpful insights, advice, and views on Petri games, which were an immense help for me when writing this thesis. I am very grateful for the offer by Bernd Finkbeiner and Ernst-Rüdiger Olderog to review this thesis. A special thank you goes to Rayna Dimitrova for spontaneously agreeing to review this thesis as well. I also want to thank Jan Reineke and Dominic Steinhöfel for completing my examination board.

I am thankful to Raven Beutner, Bernd Finkbeiner, Manuel Gieseking, Niklas Metzger, Ernst-Rüdiger Olderog, and Ann Yanich for directly collaborating on topics related to Petri games. I want to thank my colleagues Jan Baumeister, Raven Beutner, Norine Coenen, Peter Faymonville, Hadar Frenkel, Michael Gerke, Christopher Hahn, Jana Hofmann, Swen Jacobs, Felix Klein, Florian Kohn, Niklas Metzger, Sabine Nermerich, Noemi Passing, Mouhammad Sakr, Christa Schäfer, Malte Schledjewski, Frederik Schmitt, Maximilian Schwenger, Julian Siber, Leander Tentrup, Hazem Torfah, Alexander Weinert, and Martin Zimmermann from the *Reactive Systems Group* for the productive discussions, collaborations, coffee breaks, lunch breaks, and cake breaks.

I want to thank David, Elena, Florian, Immanuel, Jannik, Jeldrik, Lara, Marcel, Marius, and Sebastian for making my studies and free time such a wonderful experience. I will never forget the joint lunches, bike rides, theatre visits, parties, and other good times. I also want to thank everybody I played basketball with and against at *Hochschulsport* at Saarland University for the friendly competition and joint physical activity.

I am thankful to my family for always having my back, supporting me in every way possible, and believing in me. Last but not least, I sincerely appreciate the love, support, and understanding of my girlfriend, Noemi. This thesis would not have been possible without you.

Contents

Chapter 1

Introduction

How to construct correctly functioning computer systems is a fundamental challenge in computer science. Every day, we interact with computer systems and rely on their correct behavior. We greatly benefit from these systems because they automate the solution of complex problems for us. Think of the enormous technical achievements accomplished by the internet browser on your mobile phone. It allows you to read the newspaper, watch movies, and buy tickets for public transport, to mention just three examples of its endless possibilities of use. Meanwhile, a malfunctioning internet browser can have all sorts of terrible effects for its users, including monetary harm and leaking personal as well as shared data. In the worst case, users can be surveilled continually due to an incorrectly functioning internet browser on their mobile phone.

Constructing correctly functioning computer systems is a challenging task. One reason for this is that computer systems can engage in complex interactions with humans and other systems. Often, a lot of work goes into developing and testing these systems, and we rely on their correct behavior. However, correctness is rarely *proven*. Because human errors do happen and are not consistently recognized in time, one should not be surprised by the unintended behavior of a deployed system. Users of such a system can only hope that the unintended behavior is recognized and disclosed by people with positive intentions and fixed by the engineers of the system. If all goes well, the engineers can deploy the fix before people with harmful intentions can exploit the unintended behavior and cause harm to the users.

Numerous domains of computer science help engineers to construct correct systems, e.g., formal methods, programming languages, security, software engineering, and testing, to name just a few. In the following, we dive into the subject of *formal methods*. Here, mathematically precise models of the system are investigated with the goal of verifying the correctness of a system. The advantage of a verified system consists in the fact that we cannot only *assume* (or hope) but, in fact, *know* that it behaves correctly, i.e., in an intended way. A verified system can be deployed without fear of unintended behavior in the system, which could harm its users.

1.1. The Synthesis Problem

In this thesis, we primarily focus on the *synthesis* approach as one of the major techniques from formal methods to construct correct systems. The *synthesis problem* probes whether there exists an implementation satisfying a given specification and derives such an implementation satisfying the given specification if it exists [Chu57; Chu64]. A synthesized system is correct-by-construction, i.e., it always satisfies the specification. The synthesis approach frees engineers from the burden of manually coding systems. While writing the specification, the engineers can focus on *what* the system should achieve instead of *how* the system should accomplish its goal.

In the main part of this thesis, we present new decidability and undecidability results for the synthesis problem of *asynchronous distributed systems* from *global specifications*. These results allow us to obtain asynchronous distributed systems, which are provably correct. Asynchronous distributed systems cover most realistic systems.

As a general intuition, a *distributed system* consists of several components which can work together to accomplish a goal. For example, several concurrent robots that work on the same product constitute a distributed system.

A distributed system is an *asynchronous system* if each component progresses according to its individual clock. This contrasts with synchronous distributed systems, where either all components follow a predefined rate to progress or global synchronization between all components occurs constantly. Again, the several concurrent robots constitute a suitable example for an asynchronous distributed system because they do not necessarily have to move at the same rate.

With the help of *global specifications* (in contrast to local specifications as the current state-of-the-art), we can express powerful requirements such as mutual exclusion for asynchronous distributed systems. The requirement of mutual exclusion enforces that, at most, one process has access to a specific resource. For example, this allows for the several concurrent robots to share some tool while ensuring that at most one of them is using it at every point in time.

We use the term *monolithic system* to differentiate a system consisting of only one component from a distributed system. This distinction is helpful because the early developments regarding the synthesis approach focus on monolithic systems, whereas our contributions focus on distributed systems.

In the remainder of this chapter, we will illustrate how to use formal logics to specify the correct behavior of a system by an example from *model checking*. For simplicity, we use model checking instead of synthesis. Model checking is another central technique in formal methods. Afterward, we will survey significant developments in the field of synthesis both for monolithic and distributed systems. This will lead to in-depth descriptions of what asynchronous distributed systems are and why they are crucial to represent realistic systems. In the process, we will dissect the underlying concepts of asynchrony and distribution in asynchronous distributed systems. Furthermore, we will focus on the memory model of the synthesized systems because the memory model is paramount to obtain realistic systems and to solve the synthesis problem. We

will also emphasize our main contribution, which consists of enabling the utilization of global specifications for the synthesis of asynchronous distributed systems. Our main contribution pushes the sate-of-the-art from local specifications to more expressive global specifications, including mutual exclusion.

1.2. Specifications

In the following, we illustrate how to use formal logics [HR04] to specify correct behavior. To focus on using specifications, we introduce model checking as a conceptually more straightforward approach than synthesis that also requires a specification as input. As mentioned before, *model checking* is another fundamental technique in formal methods to verify the correctness of a system. Same as the synthesis approach, model checking is based on formal logics to specify the correct behavior of the system.

The *model checking problem* probes whether a model of the system satisfies the specification given in a formal logic [CE81; QS82]. If the model does not satisfy the specification, then a counterexample is provided. A counterexample describes some possible behavior of the system that is not included in the specification. Techniques such as *bounded model checking* [Cop+01; Bie+03; HBS06; BELM12] as well as advances in *Boolean satisfiability* (SAT) solving [JS97; SS99; IP01; CIP09; HJS19; VW21] and *satisfiability modulo theories* (SMT) solving [BMS05; MB08; Bar+11; Dut14; Web+19] made model checking viable in practice.

Tool implementations of model checking problems for several representations of models against several formal logics exist (cf. [BMMR01; McM03; CKL04; Bra11; PR11; CKP15; FGHO20a]). These tool implementations are called *model checkers*. As part of the development process, model checkers are used in industry to provide engineers with counterexamples that point them to problematic aspects of the system. Thus, model checkers help engineers to verify the final system. Model checking and model checkers constitute one crucial technique to produce correct systems.

To illustrate the use of specifications, we elaborate on the considered class of systems. This thesis assumes that a system always is a *reactive system* [HP84], i.e., it continually receives inputs from the environment and does not terminate. For distributed systems, this implies that some components can terminate, but not all of them. Considering reactive systems makes both the model checking and synthesis problem especially hard because infinite behavior needs to be handled. *Data transforming systems* [Gre69; MW80; Sol13] are different from reactive systems and inevitably terminate.

Specifying an Automated Teller Machine

An Automated Teller Machine (ATM) is a reactive system that may look simple from the outside although handling considerable sums of money. Large portions of our everyday life rely on the availability of banknotes, especially in countries like Germany where banknotes are still very important and some shops accept no other way of payment than banknotes. An incorrectly functioning ATM could disrupt our daily errands. We

use systems such as an ATM almost every day, relying on their correct behavior even though their correct design can be challenging [HP84; SN07; Kup12].

An informal specification of an ATM could look like this: The ATM should deliver only a certain amount of money to a customer after they authenticated themselves and requested to withdraw this amount of money. In all other cases, the ATM should not give any money to customers regardless of which buttons are pressed on the ATM. After serving a customer, the ATM waits for the next customer to arrive and to repeat this process. Meanwhile, an ATM should allow being refilled by authorized personnel of the operating bank, and this task should not be too difficult for the personnel.

Specifying a system by exhaustively analyzing its behavior is vital to obtain a specification for both model checking and synthesis. The example of an ATM illustrates that enumerating the intended behavior does not constitute an overly abstract view to the engineers of the system. Most design processes of a system include a step where the intended behavior of the system is discussed in detail. As observed in some theoretical undergraduate courses on computer science, the textual specification of an ATM from above can be expressed in a corresponding formal logic like *linear-time temporal logic* (LTL) [Pnu77]. Still, the task of specifying a system is nontrivial and should not be underestimated [Roz16].

Together with a model of an implementation of an ATM and by utilizing a corresponding model checker, it becomes possible to verify that the model satisfies the specification given in LTL or a counterexample representing behavior violating the specification is produced. The first case constitutes proof that the ATM behaves in an intended way, i.e., it is impossible to trick the ATM into paying out money without authentication while each righteous customer can withdraw an intended amount of money. In the second case, the counterexample can be used to find issues with the implementation that prevent the specification from being satisfied.

1.3. Synthesis of Monolithic Systems

A grand vision in formal methods is to relieve engineers from the task of manually coding systems (as is necessary when using model checking). This can be achieved by automatically deriving implementations from specifications. This so-called *synthesis* approach can automate the creation of correct systems and constitutes another crucial technique in the quest for correct systems. Instead of manually coding a system, engineers only write a specification outlining the intended behavior of the system. During this process, engineers can think on a more abstract level about *what* the system should achieve instead of *how* the system should accomplish its goal. We will focus on the synthesis approach for the remainder of this thesis.

For an example, think again about an automated teller machine (ATM). A specification of an ATM's desired behavior will always be necessary to define correctness both for model checking and for synthesis. Using the synthesis approach, the engineers of the ATM need to develop an all-encompassing specification and obtain an implementation of the ATM automatically instead of coding the implementation manually

and then verifying its correctness. A not all-encompassing specification would result in some unspecified behavior for the synthesis approach. In this case, the synthesized system would do as little as possible or show arbitrary behavior for the cases of unspecified behavior. Most likely, the engineers of the systems do not intend this behavior. It could be a fascinating research direction to study the automatic generalization of not all-encompassing specifications for synthesis. We want to stress that model checking and synthesis are orthogonal approaches to obtain correct systems. On the one hand, synthesis eliminates the manual coding step necessary for model checking. On the other hand, model checking allows verifying only the most critical parts of the system.

The first formulation of the vision of synthesis dates back to Alonzo Church more than 60 years ago [Chu57; Chu64]. Formally, the *synthesis problem* probes whether there exists an implementation satisfying a given specification and derives such an implementation if it exists [Chu57; Chu64] (for some excellent introductions, see [Tho09; Fin16; BCJ18]). The derived implementation satisfies the specification by construction and can be directly deployed. When no implementation exists, a corresponding counterexample for every possible implementation can be derived. This facilitates analyzing which part(s) of the specification prevented the derivation of an implementation. A milestone on the road towards the successful synthesis of monolithic systems was an automatically derived implementation of the AMBA AHB bus protocol, an open industrial standard for the on-chip communication and management of functional blocks in system-on-a-chip (SoC) designs [Blo+07; Job07; Blo+12; GCH13].

The following introduction to the history of synthesis follows [Fin16]. The first formulation of the synthesis problem more than 60 years ago [Chu57; Chu64] predates the invention of temporal logics like LTL [Pnu77] by about 20 years. At that time, the specification for the synthesis problem was a regular set given as a formula of the *monadic second-order logic of one successor* (S1S), and the goal was to synthesize a monolithic system consisting of one component. Remember that we use the term monolithic system to identify a system consisting of one component, which is in contrast to a distributed system consisting of more than one component. In *Büchi's Theorem* [Büc60], the connection between S1S and automata over infinite words is established. This constitutes the first step of the first solutions to the synthesis problem for S1S. There are two different solutions, which were developed in parallel. In their respective first step, they transform the given S1S formula into an equivalent automaton over infinite words.

One solution, called *automata-based synthesis*, is due to Rabin [Rab72] and based on *automata over infinite trees* [Rab69]. The synthesis problem is reduced to an emptiness check for automata over infinite trees. Here, the automaton over infinite words is extended with a distinction between *system* transitions, representing outputs of the system, and *environment* transitions, representing inputs to the system. This distinction between system and environment transitions results in branching, which leads to automata over infinite trees. The original nonelementary runtime [Rab69] of the emptiness check for automata over infinite trees has been optimized via exponential runtime [Rab72; HR72] to polynomial runtime [EJ88; PR89a]. Next, we consider an alternative solution to the synthesis problem from S1S.

1.3.1. Game-based Synthesis

We focus on the other solution, which is called *game-based synthesis*, because its underlying concepts carry over to our approach for the synthesis of asynchronous distributed systems. Remember that we synthesize monolithic systems consisting of one component in game-based synthesis described in this subsection, same as for automata-based synthesis. Game-based synthesis is due to Büchi and Landweber [BL69].

For game-based synthesis, the synthesis problem is encoded by a game between the *environment player* and the *system player*. The environment player represents the inputs to the system, whereas the system player represents the outputs of the system. The input of the environment for the system is not limited apart from the alphabet of inputs. It can also be restricted to follow a transition system [ALW89]. In each round of the two-player game, the environment player first chooses a valuation of the inputs. Afterward, the system player chooses a valuation of the outputs as the response. Both the system player and the environment player have complete observation of the valuations of previous rounds of the game and can make decisions based on this. Playing the two-player game round-by-round produces a sequence of valuations. The system player wins if this sequence satisfies the given S1S formula. Otherwise, the environment player wins. Whether the synthesis problem has a solution can thus be decided by checking the existence of a winning strategy for the system player against all possible behaviors of the environment player.

In the first step of both game-based synthesis and automata-based synthesis, Büchi's theorem gives automata over infinite words with *Büchi* acceptance condition. The Büchi acceptance condition requires that at least one state from a specific set of states is visited infinitely often. In the following, we abbreviate automaton over infinite words with Büchi acceptance condition by *Büchi automaton*. This also applies to other winning conditions that we will encounter. The automata obtained Büchi's theorem can be nondeterministic automata. Nondeterministic automata can have multiple initial states and edges with the same label that do not necessarily lead to the same successor state. Therefore, the nondeterministic Büchi automata obtained by Büchi's theorem may need to be determinized before creating two-player games from them.

Unfortunately, deterministic Büchi automata are strictly less expressive than nondeterministic Büchi automata. This contrasts with automata over finite words. There, each nondeterministic automaton can be translated into an equivalent deterministic one. Thus, the determinization of nondeterministic Büchi automata results in deterministic automata with a more expressive acceptance condition. Possible winning conditions are *Muller* [McN66], *Rabin* [Saf88], and *parity* [Mos84].

The Muller acceptance condition requires that the set of infinitely often visited states is an element of a given set of sets of states. The Rabin acceptance condition requires that, for at least one pair from a given set of pairs of sets of states, states from the first element of the pair do not occur infinitely often while at least one state from the second element of the pair occurs infinitely often. For the parity acceptance condition, every state is labeled by a number, which is also called *color*, and it is required that the largest number occurring infinitely often is even. Notice that it leads to equivalent expressive-

ness when changing largest to smallest, even to odd, or both for the definition of the parity acceptance condition. The Muller, Rabin, and parity acceptance conditions carry over as winning condition to the resulting game to solve the posed synthesis problem in game-based synthesis [BL69]. Therefore, both the size of the required strategies and the complexity of determining the winning player are of great interest.

The following results have been obtained regarding the size of the required strategies either for the winning system player or for the winning environment player: In Muller games, the memory necessary and sufficient for the winning player is bounded by the factorial of the number of states in the game. This follows from using so-called *index appearance records* [GH82; Les95]. In Rabin games, a memoryless strategy suffices for the system player [Eme85], whereas the environment player may need exponential memory [Les95]. In parity games, the winning player has a memoryless strategy [EJ91].

The following results have been obtained regarding the complexity of determining the winning player: For Muller games, the complexity depends on the representation of the winning condition. For an explicit representation of the winning condition as a set of sets of states, the winner can be determined in polynomial time [Hor08]. The *Emerson-Lei* representation [EL87] is a succinct alternative representation of the set of sets of states by a Boolean formula with variables representing the states in the game. For the Emerson-Lei representation, the problem of determining the winning player is PSPACE-complete [HD05]. For Rabin games, determining the winning player is NP-complete [EJ88]. For parity games, determining the winning player is known to be in NP $\cap$ co-NP [EJS93] and, more precisely, in UP $\cap$ co-UP [Jur98]. Recently, a solving algorithm in quasipolynomial time was found [Cal+17].

1.3.2. Synthesis from Temporal Logics

The temporal logic *linear-time temporal logic* (LTL) [Pnu77] is often more intuitive than S1S and leads to lower complexity for synthesis. In LTL, there is no explicit mechanism for quantifications as in S1S. Instead, both references to points in time and quantification over time happen implicitly by modal operators. Automata over infinite words with Büchi acceptance condition can represent LTL formulas [VW94]. There are several optimizations to the original construction [GPVW95; SB00; GO01]. After this translation, the same techniques from game-based synthesis for S1S apply to synthesis from LTL specifications. The synthesis of finite-state machines from specifications in LTL is 2-EXPTIME-complete [PR89a], measured in the size of the LTL specification.

There is a history of synthesis tools for LTL [JGWB07; Ehl11; Boh+12], and state-of-the-art tools synthesize systems of considerable size [FFT17; MC18; LMS20]. A yearly competition fosters the development and comparison of synthesis tools for LTL [Jac+17b; Jac+15; Jac+16; JB16; Jac+17a; Jac+19]. *Quantitative synthesis* allows for optimizing the behavior of correct systems further by using quantitative objectives [BCHJ09]. For example, it becomes possible to specify that all requests are answered as quickly as possible by grants.

LTL assumes implicit universal quantification over all possible paths in the system. By contrast, *branching-time* temporal logics introduce path quantifiers leading to ex-

plicit existential and universal quantification over all paths that are possible in the system. This makes it possible to specify that certain futures are feasible or not. For example, this allows us to specify that a specific alternative is always possible, even for a path where it is never taken.

The temporal logic *computation tree logic* (CTL) [CE81] introduces existential and universal quantification over paths directly followed by a modal operator. The expressiveness of LTL and CTL is incomparable. CTL* is a generalization of CTL [EH85] and subsumes the expressiveness of both LTL and CTL. Synthesis from a specification given in CTL is EXPTIME-complete [VW86]. The lower bound follows from the satisfiability problem of CTL [FL79]. Synthesis from a specification given in CTL* is 2-EXPTIME-complete [Eme90].

1.3.3. Summary

We summarize the primary takeaway from the history of synthesis of monolithic systems for the remainder of this thesis. The synthesis problem is often represented as a two-player game with complete observation between the *system* player and the *environment* player. The game is played on an underlying game arena. We have seen that the game arena can directly originate from the specification, but we will see later that the game arena can also be part of the specification. In the game arena, each move by the system player corresponds to a part of the possible implementation of the system, and each move by the environment player corresponds to a possible environment input to the system. The system player and the environment player encode that the system continuously interacts with the environment as in a reactive system.

The game is turn-taking between the environment player and the system player. If we want to encode that one of the players can make consecutive moves, then they must be subsumed as one move by the respective player. We will see later that the assumption of players making moves in a turn-taking fashion must and can be relaxed. All moves represent the possible functions of the system in its environment. It might be the case that, in a particular state of the game, not all moves by the current player are possible due to the structure of the game arena. Both players can choose their next move depending on the previous moves of the system player and the environment player. Strategies for the players determine their next move, given the current position of the game and the history of previous moves.

The specification defines the winning condition of the game. The winning condition of the game is interpreted from the point of view of the system player. It can be represented directly as states in the game that should never be reached, that should be reached, or that should be reached repeatedly. The winning condition can also be represented by a finite automaton that is run in parallel to the two-player game to determine the winner of the game. The goal of the system player is to satisfy the winning condition of the game, whereas the goal of the environment player is to violate the winning condition. A strategy for the system player satisfying the winning condition is a correct-by-construction implementation because the strategy embodies the correct reaction of the system to all behaviors of the environment.

1.4. Synthesis of Distributed Systems

More complex interactions between the system and environment than are possible in monolithic systems can be realized by *distributed systems* [TS07]. In distributed systems, there are several concurrent components of a larger system. These components can repeatedly interact with other components and the environment. This contrasts with monolithic systems, where the system consists of one component. In this thesis, we consider reactive systems, i.e., not all components of the distributed system terminate and these non-terminating components interact continually with the environment.

Although we extended the model of the system from monolithic to distributed systems, the game-based approach for the synthesis of monolithic systems proves to be sustaining for the synthesis of distributed systems. The *synthesis of a distributed system* can be encoded by multiple system players playing against multiple environment players. Each system player is a separate component of a larger system, e.g., one of many autonomous transportation robots in a large factory. Each environment player is an independent source of uncontrollable behavior for the system consisting of multiple separate components. Each player has individual memory.

Each player can perform local actions as well as joint actions with some other players. Joint actions between players are called *synchronizations* and enable the exchange of information between these players. Each system player acts on its individual information and requires a local strategy which, in combination with the strategies of the other system players, satisfies the winning condition 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.

Multiple environment players are necessary to ensure the independence of different parts of the environment. This enables the environment to hide certain decisions from the system while sharing certain other decisions with some parts of the system. An environment player can, for example, be a worker in the factory requesting the delivery of material by a robot or noise in the sensor readings of a robot.

1.4.1. Synchronous vs. Asynchronous Systems

Early formulations of the synthesis problem for distributed systems focused on the *synchronous* setting [PR89a]. In the synchronous setting, all components progress at the same specific rate, i.e., all system players and all environment players make their next move simultaneously. This results in an undecidable problem [PR90; KV01; FS05] because the synthesis of distributed systems in the synchronous setting can encode the halting problem for Turing machines [Tur37].

In the construction of the proof, two system players act on incomplete information about the other system player. The construction of the proof ensures that the only possibility for them to win the game is to simulate the Turing machine at each of the two system players. The incomplete information can be used to shift one system player one simulation step of the Turing machine ahead of the other system player. This possibility for a shift can be used to ensure that both system players simulate the Turing machine

correctly. Then, the synthesis of distributed systems in the synchronous setting can be used to decide the halting problem for Turing machines. This makes the synthesis of distributed systems in the synchronous setting undecidable. We will come back to this proof technique and use it for our undecidability results.

An *architecture* defines which components of a distributed system can communicate with each other. The synthesis of distributed systems in the synchronous setting remains decidable for some specific architectures. For these architectures, both an *automata-based* solution [KV01] and a *game-based* solution [MW03] are possible, as for the synthesis of monolithic systems. Architectures with a decidable synthesis problem are one-way ring architectures [KV01] specifically, and generally, all architectures where processes can be ordered according to their informedness, i.e., all architectures without so-called *information forks* [FS05]. An information fork occurs when two processes receive information from the environment according to the architecture such that they cannot completely deduce the information that the other process received. For architectures without information forks, the complexity of the synthesis of distributed systems is nonelementary in the number of processes.

To circumvent this high complexity, *compositional synthesis* methods have been proposed. In compositional synthesis, processes are synthesized individually, and their implementations are composed afterward. This approach often requires processes to make assumptions on the behavior of other processes. *Assume-guarantee synthesis* allows processes to make assumptions on the behavior of other processes while guaranteeing that all these assumptions are fulfilled [CH07]. There are multiple algorithms to find these assumptions [AMT15; MMSZ20; FP21]. Using *dominant strategies*, we do not explicitly need to search for assumptions between processes [DF11; DF14; BRS17; FP20]. Instead, we use a weaker winning condition, i.e., we search for dominant strategies instead of winning strategies. This technique leads to implicit assumptions between the processes. Because these assumptions are weaker than explicit ones, compositional synthesis with dominant strategies is not necessarily always possible.

For the synthesis of monolithic systems, the synchronous setting is well-suited to represent hardware circuits. Still, this setting reaches its limitations in the case of distributed systems, as observed by the undecidability result mentioned above and the high complexity for the decidable classes. Therefore, we consider the *asynchronous* setting for the synthesis of distributed systems [PR89b; SF06], which is even more desirable from the developer's point of view. Here, each component, i.e., each system player and each environment player, can proceed at its individual rate instead of at a fixed rate for all players. The asynchronous setting allows for maximal freedom in modeling distributed systems because components can proceed at their individual rates and wait for their synchronization partners when needed. For example, a worker in the factory can perform several steps of their task locally before communicating with other workers, while a robot moves through the factory to obtain some required material. The trajectory of the robot is controlled locally. Of course, relative levels of abstraction are essential in this example: One could also focus more sharply on how the robots navigate the factory while modeling the workers only rudimentarily.

1.4.2. Memory Models

The memory model presupposed in the synthesis of distributed systems is another crucial factor to be considered in order to obtain a realistic and decidable class of systems [KPS11]. In monolithic systems, we assume *complete information* between the system player and the environment player because this is a realistic assumption, and there are not many other options due to only having two players. Maintaining complete information for distributed systems is incompatible with the possibility of having multiple system players and multiple environment players. This is the case because each player would be informed about the behavior of all other players. Then, the parallel composition of all system players would give one system player, and the parallel composition of all environment players would give one environment player, which would bring us back to the two-player games as representation for the synthesis of monolithic systems. Therefore, we need an *incomplete information* model [KV00].

One possibility of an incomplete information model is *local information*, where players do not learn anything about the past, present, and future behavior of other players. Such a framework could still allow synchronization between the players but would forbid the strategies for the players to make different decisions based on their past and synchronizations with other players. Such a strategy for the system players, which bases decisions only on local information, is called a *positional strategy*. However, this framework is unrealistic, leading to cumbersome representations of real-world problems because players can never base their decisions on whether or not some behavior of other players has actually occurred, and was or was not observed by them.

Causal memory has evolved as the natural model for information flow in asynchronous distributed systems. Here, all participating components of a synchronization exchange their complete history [GLZ04a; GLZ04b; MTY05]. Components remain uninformed about the parallel progress of other components unless communication in the form of synchronization takes place. In case of a synchronization, the components learn the entire past of all participating components, including their previous synchronizations with other components. Notice that components remain uninformed about the future decisions of other components when communication occurs. Before deploying a synthesized system, the exchange of unused or unneeded information between players could be pruned to optimize the information exchange. As we wish to equip the players with maximal options, we opt for *causal memory* in the asynchronous distributed systems, which we consider in the remaining parts of this thesis.

1.5. Petri Games

Our work utilizes *Petri games* [FO17] for the synthesis of asynchronous distributed systems with causal memory. Petri games are based on *Petri nets* [NPW81; Rei85; Old91] and are a suitable technique for the synthesis of realistic distributed systems. Petri nets define the flow of tokens from and to places via transitions. In a Petri game, the set of places of an underlying Petri net is distributed into the set of system places and the

set of environment places. The underlying Petri nets of Petri games can have loops, i.e., a marking as the representation of the current places of all tokens can be reachable infinitely often. This is necessary to encode reactive systems and makes the synthesis problem for Petri games especially difficult.

Using Petri nets as the underlying game arena of Petri games naturally encodes asynchronous systems because the flow of tokens occurs asynchronously in Petri nets. In Petri games, system players are represented by tokens in system places, whereas environment players are represented by tokens in environment places. Having two or more tokens on system places naturally encodes distributed systems because every system player represents one component of the system that should be synthesized.

All participating players of a joint transition exchange their causal past, which is represented by the *unfolding* [Eng91; MMS96; EH08] of the underlying Petri net. A Petri game is played as follows: Based on the causal past, the local strategy of each system player can deactivate outgoing transitions of that system player. Next, each environment player chooses one of its remaining outgoing transitions. Then, one of the remaining (i.e., by all participating players allowed) transitions occurs and the participating players are moved. Afterward, the moved players make their decisions as before, and this process repeats itself. Notice that, due to concurrency between independent players, all orders of transitions are possible. Notice further that transitions do not either belong to the system or the environment. Instead, all participating system players and all participating environment players have to agree on a move for it to happen. This approach also replaces the assumption of two-player games that the system and the environment player make moves in a turn-taking manner.

For safety winning conditions, the strategies for the local system players have to avoid deadlocks to prevent them from satisfying the safety winning condition by only moving very little or not at all. The local safety winning condition of *bad places* defines places the players have to avoid from the point of view of the system players. Therefore, the system players win the Petri game when there exist local strategies for the system players such that the bad places are always avoided. Otherwise, the environment players win the Petri game. For bad places, the synthesis problem for Petri games with a bounded number of system players and one environment player is decidable in single exponential time [FO17].

The global safety winning condition of *bad markings* defines markings (i.e., simultaneous positions of all players) that the players have to avoid from the point of view of the system players. Here, the system players win the Petri game when there exist local strategies for the system players such that the bad markings are always avoided. Otherwise, the environment players win the Petri game. For bad markings, the synthesis problem for Petri games with one system player and a bounded number of environment players is decidable in single exponential time [FG17]. Local winning conditions cannot express global properties like mutual exclusion, where only one player at a time is allowed access to a resource. The natural question arises whether the synthesis problem for further classes of Petri games with *global* winning conditions is also decidable. We will answer this question in the remainder of this thesis.

1.6. Contributions

In this thesis, we make two contributions to the synthesis of asynchronous distributed systems with causal memory represented by Petri games. In the first part of this thesis, we broaden the landscape of Petri games with a decidable or an undecidable synthesis problem by providing one decidability result and two undecidability results for Petri games. In the second part of this thesis, we present bounded synthesis for Petri games to find winning strategies for the system players in all Petri games with a bounded number of players.

1.6.1. Part I: Decidability and Undecidability

In the first part of this thesis, we prove that it is decidable whether a winning strategy exists in Petri games with a bounded number of system players, at most one environment player, and bad markings as global winning condition. This result is obtained by a reduction to two-player games with complete observation. The two-player game has a Büchi winning condition, i.e., at least one state from a specific set of states has to be reached infinitely often for the system player to win. The general idea of the proof is inspired by the decidability result for bad places as local winning conditions [FO17]. We extend the reduction from the *local* winning condition of bad places to the *global* winning condition of bad markings, introducing some significant extensions.

In the beginning, we explain in detail what the reduction for bad places and the reduction for bad markings have in common: First, we outline how so-called *decision tuples* represent the multiple system players and the one environment player in the Petri game to obtain states in the two-player game. Second, we show how transitions with an environment place in their precondition can be fired as late as possible and how system players fix their next decision explicitly as soon as they participate in a transition. This proves that the infinite causal memory of the players in the Petri game can be represented by the finite number of states in the two-player game.

Afterward, we present two significant additions to extend the reduction from bad places as local winning condition to bad markings as global winning condition. First, we outline how the case where system players can play infinitely without firing a transition with an environment place in its precondition can be handled directly in the two-player game. Second, we present how to finitely store the history of each system player until its last synchronization with the environment player. We show that a sequential run in the two-player game suffices to check if a bad marking is reached for all possible interleavings between the concurrent moves of all players in the Petri game.

In the following chapter, we prove that it is undecidable whether a winning strategy exists in most Petri games with one of the following two winning conditions: The global winning condition of *good markings* defines markings that the players have to reach during every run allowed by the strategy for the system players to win. The global winning condition of *good and bad markings* defines good markings that the players have to reach during every run allowed by the strategy and bad markings that must be avoided until a good marking is reached for the system players to win.

First, we prove that it is undecidable whether a winning strategy exists in Petri games with at least two system players, one environment player, and good and bad markings as global winning condition. In the beginning, we outline how the independence of the three players and good markings can be used to simulate an undecidable synchronous setting in the asynchronous setting of Petri games. Therefore, we show how good markings can be used to disregard interleavings between the two independent system players that deviate too much from the synchronous setting. The main idea is that a good marking is reached as soon as players deviate from the synchronous setting. Afterward, the incomplete information of the two system players (due to their independence) can be used to force each system player simulate a Turing machine. Minor deviations from the synchronous setting can be used to ensure that both system players correctly simulate the Turing machine. Notice that, in the proof, we encode the undecidable Post correspondence problem (PCP) [Pos46] instead of the halting problem for a Turing machine for a more straightforward construction.

Second, we prove that it is undecidable whether a winning strategy exists in Petri games with at least one environment player, two players where each of the two players changes between being a system player and an environment player, and good markings as global winning condition. We show that the bad markings to detect wrong solutions to the PCP from the previous undecidability proof can be encoded by each system player repeatedly becoming an environment player. For each bad marking, there exists a transition only from the corresponding environment players that leads to a place where no good marking can be reached anymore. When no such bad marking is reachable, then the environment players can decide to become system players again for the next decision on how to simulate the Turing machine.

1.6.2. Part II: Bounded Synthesis

Bounded synthesis finds winning strategies in Petri games with a bounded number of players and the global winning condition of bad markings [Fin15]. A restriction to one environment player or one system player as for the decidability results is not necessary. Therefore, bounded synthesis is applicable to Petri games with multiple system players *and* multiple environment players. Bounded synthesis is based on bounding the size of the possibly infinite unfolding and can thereby also find winning strategies of minimal size. For a bounded unfolding, the existence of a winning strategy is encoded by a quantified Boolean formula (QBF). For QBFs, propositional formulas are extended with existential and universal quantification over Boolean variables. The QBFs for bounded synthesis start with an existential quantifier over corresponding variables to fix a strategy for the system players. Afterward, a universal quantifier over corresponding variables and the main part of the QBF follow to check whether all admitted runs of the strategy are winning.

In the second part of this thesis, we present two algorithms for finding bounded unfoldings depending on whether the graph of reachable markings of the Petri game has loops or not. For the case of loops in the graph of reachable markings, the algorithm is based on finite prefixes of the infinite unfolding [Esp94; KKV03; EH08; Bon+14] and

a bound on the number of copies per place. For the case of no loops in the graph of reachable markings, the algorithm efficiently calculates the finite unfolding. WE improve the previously existing encoding for the global safety winning condition of bad markings and introduce a similar encoding for the global liveness winning condition of good markings.

In both of these sequential encodings, all interleavings between moves of the players are enumerated and tested. This implies that, for two independent moves t_1 and t_2, both sequences t_1t_2 and t_2t_1 are checked. Enumerating and testing all interleavings between moves of the players is necessary when using global safety winning conditions like bad markings or global liveness winning conditions like good markings because they can identify specific sequences of moves by the players as bad or good. We present a second new encoding: In the so-called true concurrent encoding, independent moves of the players occur at the same time if possible. This has the drawback that only local safety winning conditions like bad places can be decided, but the performance of the true concurrent encoding is better than the performance of the sequential encoding. Both encodings for the local winning condition of bad places are implemented and evaluated in our tool AdamSYNT [Ada20].

1.7. Related Work

In this section, we first outline other flavors of synthesis to obtain a complete picture of the synthesis approach. Second, we highlight the subtle differences between Petri games and control games played on asynchronous automata. There is a close relationship between both game models. We will see that the difference is not only due to the fact that Petri games are played on Petri nets, whereas control games are played on asynchronous automata, but in the precise way strategies can restrict the possibilities of the system players.

1.7.1. Synthesis

First, we introduce different approaches for the synthesis of distributed systems. These approaches increase the applicability and extend the expressive power of the synthesis of distributed systems. Then, we outline the extension of synthesis to real-time systems and reactive programs (instead of models of such programs). These approaches represent different possibilities to extend the expressiveness of synthesis by including new and unique systems. Furthermore, we survey template-based synthesis and give more details on the general applicability of bounded synthesis. These approaches tackle the scalability and applicability of synthesis to make it more usable in practice.

Synthesis of Distributed Systems

There are multiple ways to extend the synthesis of distributed systems to solve problems of interest. These approaches deal with scalability and applicability and extend the expressive power of the considered systems.

In many cases, the synthesis of distributed systems does not scale to real-world systems. This problem can be tackled by restricting the systems which should be synthesized. During the design of distributed protocols, synthesis can be used to automate the creation of parts of the implementation for the engineers. Here, synthesis completes an incomplete implementation of the protocol in such a way that the specification is fulfilled [AT17]. Alternatively, one can enforce and utilize symmetry between components of the distributed system: Often, the components of a distributed system are not each of unique form but instead follow the same blueprint because this makes them easier to build and maintain. Such distributed systems are called *symmetric systems*. In this case, i.e., when all components of a distributed system are symmetric, the synthesis problem can be solved more efficiently [EF17]. One can also synthesize the components of the distributed systems from local specifications such that their fair composition satisfies the specification [GS13].

To extend the expressive power, features of the considered distributed systems are added. In large-scale distributed systems, hardware failures of individual components cannot be avoided. To cope with this fact, the synthesis of distributed systems can be extended to include *failure resilience* [BBJ16]. Here, systems are generated that can ensure their correctness despite some components failing. In the asynchronous setting, the synthesis problem for certain specifications can be reduced to the synthesis problem in the synchronous setting [KPP12; BNS18]. This allows the synthesis of more systems by increased expressivity and scalability. Furthermore, a parametrized number of processes can be considered in the asynchronous setting [BBLS20]. Here, cutoffs can be identified such that it is enough to examine a bounded number of process architectures to solve the synthesis problem.

Synthesis of Real-time Systems

A possible extension of the model of the system is to allow interactions between the system and the environment in *real-time*. Real-time systems are modeled primarily by *timed automata* [AD94; Alu99]. In timed automata, clocks are used to represent the passage of time. The generalization of timed automata for the synthesis of real-time systems are *timed games* [MPS95; AMPS98; Cas+05]. In timed games, actions of timed automata are distinguished as either *controllable* or *uncontrollable*. The strategy for a timed game does not only define which controllable actions are allowed but also at exactly which points in time. The goal of the strategy remains to satisfy the winning condition of the timed game. Timed games can be solved by the tool Uppaal Tiga [Beh+07], which interleaves forward state space exploration with backward propagation of winning sets of states. The tool Uppaal Tiga is used to synthesize real-time systems in practice [JRLD07; LCMT18].

The synthesis problem for real-time systems is undecidable when the state space is only partially observable [BDMP03]. When the number of clocks and the precision of the guards is limited in advance, i.e., when the so-called assumption of *fixed granularity* holds, then the synthesis problem for real-time systems is 2-EXPTIME-complete [DM02; BDMP03]. As enumerating granularities does not yield an effective

synthesis algorithm, a *counter-example guided abstraction refinement* (CEGAR) algorithm for timed games can be used in order to obtain an effective synthesis algorithm [DF12]. The CEGAR algorithm successively refines a set of observation predicates until a sufficiently precise abstraction is obtained. Alternatively, when the assumption of fixed granularity does not hold, the synthesis of real-time systems from templates has been proposed [FP12]. Here, the possible solutions are restricted to instances of a given timed automaton with a parametric control structure.

Synthesis of Programs

The synthesis algorithms considered so far produce finite-state automata, Petri nets, or other unstructured representations as output. Meanwhile, manual coding is always performed in some programming language. Synthesis of *reactive programs* overcomes the mismatch between programming languages and the output of the synthesis algorithms considered so far [Mad11; BCJK15; GKF18]. The synthesis problem for reactive programs can be solved by automata-theoretic transformations without requiring optimizations of the structural quality of the implementation. Instead, the output implementations can be directly deployed.

Syntax-guided synthesis (SyGuS) generalizes the problem of synthesizing a program for a specific programming language to the problem of synthesizing a program under some syntactic guidance captured by logics and grammars [Alu+15]. This is achieved by having the legal syntax of a solution as an additional parameter for the synthesis problem. Solutions to the syntax-guided synthesis problem are mainly based on *counter-example guided inductive synthesis* (CEGIS) [Sol+06], which can be implemented by enumerative learning [Udu+13; FB18], encoded to SAT and SMT [SRBE05; Sol+06; JGST10; GJTV11; KMPS12], and realized by stochastic search [SSA13].

There is a broad range of tools for SyGuS [RT17; PSM17; Rey+19] and a yearly competition to foster the development and comparison of such tools [AFSS17; Alu+19; PMNS19]. Syntax-guided synthesis has been successfully applied in practice [PSM16; Car+17; Si+18; FPMG19; RBLT20]. Furthermore, multiple extensions to the underlying formalism of SyGuS have been proposed to increase its expressivity [HD18; Hu+19; HCDR20; Mor+20].

The temporal logic *Temporal Stream Logic* (TSL) presents another possibility to synthesize realistic programs by separating data and control [FKPS19b]. TSL has been used successfully to synthesize a music player app for the mobile operating system *Android*, a controller for an autonomous vehicle in *The Open Race Car Simulator* (TORCS), a real-world kitchen timer application, and a space shooter arcade game realized on a *field-programmable gate array* (FPGA) [FKPS19b; FKPS19a; GHKF19]. The separation of data and control trades theoretical decidability for practical scalability. Data is represented by predicates and functions, which are not part of the synthesis problem. Instead, a system has to satisfy the TSL specification for all possible interpretations of the functions and predicates in order to be a solution to the synthesis problem. This implicit quantification makes it possible to independently implement the data processing part, e.g., by deductive synthesis [MW80] or manual coding.

Bounded Synthesis

Bounded synthesis was not explicitly developed for Petri games [Fin15] but is a general framework that was applied to Petri games. Bounded synthesis probes the existence of an implementation represented by a finite-state machine having at most as many states as a given number [SF07; FS13]. This results in a finite state space, making the bounded synthesis problem decidable irrespective of the system model. Furthermore, bounded synthesis can be used to find small implementations by incrementally increasing the bound starting from a small number.

The algorithm for bounded synthesis can be implemented by an automata-theoretic construction [SF07; FS13] or via a reduction to a decidable SMT problem [SF07; FS13; KB17], a decidable SAT problem [FFRT17], a decidable QBF problem [FFRT17], or a decidable DQBF problem [FFRT17]. *Dependency quantified Boolean formulas* (DQBF) are an extension of QBFs, which allow for non-linear dependencies between quantified variables. For each existentially or universally quantified variable in a DQBF, we can specify precisely on which variables of previous quantifiers it depends. This contrasts with QBFs, where an existentially or universally quantified variable depends on *all* variables of previous quantifiers.

It was shown that the algorithm for bounded synthesis is NP-hard [PF12]. There are multiple extensions for bounded synthesis, both for the model of the system and the used algorithm. The model of the system can be extended to real-time systems [PF12], asynchronous distributed systems with causal memory [Fin15], reactive programs [GKF18], and systems with a bounded number of registers [KMB18; KK19]. The algorithmic extensions include an additional search for bounded unrealizable specifications [FT14a], bounding the cycles in the implementation [FK16; FK17], and a more efficient search for an implementation via refinement [FJ12; JS20].

Bounded synthesis can be extended to either bound both the behavior of the system and the environment or to bound only the behavior of the environment. This can be achieved by introducing a bound on the size of the state space of a transition system that describes the possible behaviors of the environment [KLVY11]. Alternatively, a bound can be associated with the sequences of values of input signals produced by the environment [DFT19]. These sequences must be ultimately periodic and need to be representable by a *lasso* of bounded size. A lasso consists of a sequence at the beginning followed by another sequence that is repeated infinitely often. Both sequences together have to be smaller than the bound on the size. The two mentioned approaches allow for a more precise analysis of the synthesis problem from LTL specifications.

Specification Logics

There are more logics than S1S, LTL, CTL, and CTL* to specify reactive systems. Here, we mention some notable extensions. An LTL formula that requires that every request is answered by a grant is satisfied by a system where the distance between every request and the following grant grows with the number of pairs of request and grant in the past. To overcome this unbounded distance between requests and grants, the

temporal logic *parametric temporal logic* (PLTL) has been introduced [AETP01]. It allows for explicit bounds for eventually operators. As an alternative to PLTL, the logic PROMPT-LTL extends LTL with the *prompt-eventually operator* [KPV09]. The satisfiability of a PROMPT-LTL formula requires that there is a bound on the waiting time of all prompt-eventually operators. Both the synthesis of distributed systems from PROMPT-LTL [JTZ18] and assume-guarantee synthesis for PROMPT-LTL [FMMV20] have been considered in the literature.

There are some temporal logics that extend the quantification of CTL and CTL* further: *Alternating-time temporal logics* like ATL, ATL*, and *game logic* (GL) offer selective quantification over paths that are possible outcomes of games [AHK02]. This allows encoding the synthesis problem for LTL, among others, as a model checking problem for ATL* [AHK02; Pin07]. *Strategy Logic* (SL) subsumes ATL, ATL*, and GL and allows expressing the existence of Nash equilibria and secure equilibria [CHP10]. *Coordination logic* again subsumes ATL, ATL*, GL, and SL and allows quantification over strategies under incomplete information [FS10].

1.7.2. Control Games

A formal connection [BFH19a; BFH19b; Beu19] exists between Petri games and *control games* [GGMW13] based on *asynchronous automata* [Zie87]: Petri games can be translated into control games, and vice versa, at the expense of an exponential blow-up in each direction. Both game types admit strategies based on causal information, but the formalisms for the possibilities of system players and environment players differ.

In control games, an action is either controllable or uncontrollable. Hence, it can be restricted by either all or by none of the involved players. From a given state of a process, controllable as well as uncontrollable behavior is possible. By contrast, Petri games utilize a partitioning into system and environment places. While this offers more precise information about which player can control a shared transition, a given place can no longer give rise to both system and environment behavior.

It is challenging to encode the controllability of transitions in Petri games into control games while preserving the causal information of players, and vice versa. For the translations from Petri games to control games and from control games to Petri games, the concept of *commitment sets* has been introduced: The local players do not allow behavior directly but move to a state or a place that explicitly encodes their decision of what to allow. Using this explicit representation, the controllability aspects of one game model can be expressed in the respective other game model: For the translation from Petri games to control games, commitment sets make it possible to have actions in control games, that can be controlled by only a subset of local players, in order to encode transitions from Petri games. For the translation from control games to Petri games, commitment sets make it possible to have places in Petri games, that comprise both environment and system behavior, in order to encode controllable and uncontrollable actions from control games.

The translations between Petri games and control games preserve the structure of winning strategies in a weak bisimilar way. In addition to the upper bounds established

by the exponential translations, matching lower bounds are provided. The translations show that contrasting formalisms can be overcome, whereas the lower bounds highlight an intrinsic difficultly to achieve this. The translations between Petri games and control games enable the transfer of decidability results in acyclic communication architectures [GGMW13], which were obtained initially for control games, to Petri games. Vice versa, the translations between Petri games and control games also enable the transfer of decidability results in single-process systems [FG17], which were obtained initially for Petri games, to control games.

Decidability for control games has also been obtained for restrictions on the dependencies of actions in series-parallel systems [GLZ04b]. Furthermore, decidability has been obtained for restrictions on the synchronization behavior. These restrictions enforce that players either only learn the length of the history of other players [MT02] or players can never (directly or indirectly) communicate after they have not done so for a certain amount of steps [MT02]. Additional decidability results exist for control games with acyclic communication architectures [MW14] and for decomposable games, where it only needs to be checked whether strategies without useless repetitions are winning [Gim17].

Control games can be divided into *action-based* and *process-based* [MWZ09]. In action-based control games (cf. [GLZ04b; MW14]), each process declares which actions it allows to execute, and an action can be executed when all involved processes allow it. In process-based control games (cf. [MT02; MTY05; GGMW13; Gim17]), whether an action is allowed or not depends on the inspection of all involved processes. This gives more power to the strategy for the processes because it can look at all involved processes when deciding whether to allow or to disallow an action. Process-based control games can be converted into action-based control games, but not the other way around [MWZ09].

1.8. Publications

This thesis is based on the following peer-reviewed publications:

[FGHO17] Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. “Symbolic vs. bounded synthesis for Petri games”. In: *Proceedings of SYNT@CAV*. EPTCS. Vol. 260. 2017, pp. 23–43. DOI: 10.4204/EPTCS.260.5

[HT18] Jesko Hecking-Harbusch and Leander Tentrup. “Solving QBF by abstraction”. In: *Proceedings of GandALF*. EPTCS. Vol. 277. 2018, pp. 88–102. DOI: 10.4204/EPTCS.277.7

[BFH19a] Raven Beutner, Bernd Finkbeiner, and Jesko Hecking-Harbusch. “Translating asynchronous games for distributed synthesis”. In: *Proceedings of CONCUR*. LIPIcs. Vol. 140. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 26:1–26:16. DOI: 10.4230/LIPIcs.CONCUR.2019.26

[HM19b] Jesko Hecking-Harbusch and Niklas Metzger. "Efficient trace encodings of bounded synthesis for asynchronous distributed systems". In: *Proceedings of ATVA*. Lecture Notes in Computer Science. Vol. 11781. Springer, 2019, pp. 369–386. DOI: 10.1007/978-3-030-31784-3_22

[FGHO19] Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. "Model checking data flows in concurrent network updates". In: *Proceedings of ATVA*. Lecture Notes in Computer Science. Vol. 11781. Springer, 2019, pp. 515–533. DOI: 10.1007/978-3-030-31784-3_30

[FGHO20a] Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. "AdamMC: a model checker for Petri nets with transits against Flow-LTL". in: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 12225. Springer, 2020, pp. 64–76. DOI: 10.1007/978-3-030-53291-8_5

[FGHO20b] Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. "Model checking branching properties on Petri nets with transits". In: *Proceedings of ATVA*. Lecture Notes in Computer Science. Vol. 12302. Springer, 2020, pp. 394–410. DOI: 10.1007/978-3-030-59152-6_22

[GHY21] Manuel Gieseking, Jesko Hecking-Harbusch, and Ann Yanich. "A web interface for Petri nets with transits and Petri games". In: *Proceedings of TACAS*. Lecture Notes in Computer Science. Vol. 12652. Springer, 2021, pp. 381–388. DOI: 10.1007/978-3-030-72013-1_22

[FGHO22] Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. "Global winning conditions in synthesis of distributed systems with causal memory". In: *Proceedings of CSL*. LIPIcs. Vol. 216. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 36:1–36:19. DOI: 10.4230/LIPIcs.CSL.2022.36

Furthermore, this thesis contains material published in the following reports:

[BFH19b] Raven Beutner, Bernd Finkbeiner, and Jesko Hecking-Harbusch. "Translating asynchronous games for distributed synthesis (Full Version)". In: *CoRR* abs/1907.00829 (2019). arXiv: 1907.00829

[FGHO21] Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. "Global winning conditions in synthesis of distributed systems with causal memory (Full Version)". In: *CoRR* abs/2107.09280 (2021). arXiv: 2107.09280

In this thesis, we present work in collaboration with many people. In the remainder of this section, we list details in authorship for the papers in which the results of this

thesis are also published. Minor differences in text between the published papers and this thesis are supposed to achieve better typesetting for this thesis.

Chapter 2 presents the background of this thesis and introduces Petri nets and Petri games. The invention of Petri nets and Petri games is not the work of the author of this thesis. The presentation of the definitions of Petri nets and Petri games has been used in the papers connecting Petri games and control games [BFH19a; BFH19b], which were co-authored by the author of this thesis.

The decidability and undecidability proofs from Chapter 3 and Chapter 4 have been published in [FGHO22]. Formal details of the proofs are from the corresponding full version [FGHO21]. Both chapters constitute the work of the author of this thesis, who is the lead author of both papers. The decidability and undecidability proofs have not and will not appear in other theses by the authors of these papers.

Chapter 5 presents an introduction to bounded synthesis for Petri games with bad markings as global winning condition, out of which parts have been published in [FGHO17]. The author of this thesis is a co-author of said paper. The invention of bounded synthesis for Petri games is not the work of the author of this thesis. For this chapter, the work of the author of this thesis includes two algorithms to obtain bounded unfoldings, corrections and optimizations of bounded synthesis for Petri games with bad markings, the implementation thereof as part of the tool AdamSYNT [Ada20], and the definition as well as the implementation of bounded synthesis for Petri games with good markings as global winning condition. The accompanying web interface for AdamSYNT has been published in [GHY21]. The author of this thesis is a co-author of said paper. How to use the certificates of the used QBF solver QuAbS to model a synthesis problem as a Petri game is one of the topics of publication [HT18]. This section constitutes the work of the author of this thesis, who wrote the corresponding section in said paper as a co-author. None of the mentioned works of the author of this thesis have or will appear in other theses by the authors of these papers.

The true concurrent encoding for bounded synthesis from Chapter 6 has been published in [HM19b]. The content is the work of the author of this thesis, who is the lead author of the paper. The content constitutes a significant improvement and extension of preliminary work, which has been published in the Bachelor's thesis [Met17] of the co-author of the paper. The improved and extended content from this paper has not and will not appear in other theses by the authors of this paper.

Steps towards liveness winning condition per player mentioned as future work in Chapter 7 have been published in [FGHO19; FGHO20a; FGHO20b]. The author of this work is one of the co-authors developing these ideas.

1.9. Structure of this Thesis

In Chapter 2, the main part of this thesis starts with an in-depth introduction to Petri nets and their extension to Petri games. We also focus on multisets to represent the bounded number of players in Petri games and give examples of how the unfolding of a Petri net represents the causal memory of players in a Petri game. Chapter 2 is followed

by Part I and Part II: Part I focuses on decidability and undecidability results for Petri games, whereas Part II focuses on bounded synthesis for Petri games. In Chapter 7, conclusions are drawn, and open problems are discussed.

1.9.1. Part I: Decidability and Undecidability

In Chapter 3, we prove that it is decidable whether a winning strategy exists in Petri games with a bounded number of system players, at most one environment player, and bad markings as global winning condition. This result is obtained by a reduction to two-player games with complete observation as in the case of bad places as local winning condition [FO17].

In the first section of Chapter 3, we give a motivating example to showcase the expressivity of the systems that can be synthesized by the new decidability result. In the next section, we introduce Büchi games as our formalism of two-player games with complete observation. In the subsequent section, we give an overview of the reduction. Therefore, we explain the similarities between the reduction for bad places and the reduction for bad markings in detail.

Afterward, we describe the parts of the Büchi games encoding the Petri games of our decidability result in individual subsections. We present the states in the Büchi game, describe which states are initial as well as which are accepting, and differentiate which states belong to Player 0 as well as which belong to Player 1. We also describe the edges in the Büchi game. In the following two subsections, we introduce backward moves in the Büchi game and how the NES-case is encoded directly in the Büchi game. As the last subsection of this section, we state the decidability result formally and present an intuition on how strategies between the Petri games from our decidability result and the encoding Büchi game can be translated in both directions.

In the next section, we formalize all the steps of the preceding section and present proofs for the correctness of the construction. To conclude the chapter, we discuss how the backward moves from the new decidability result for bad markings as global winning condition solve corner cases concerning the determinism requirement of strategies for the system players in the reduction for bad places as local winning condition.

In Chapter 4, we prove two undecidability results: First, we prove that it is undecidable whether a winning strategy exists in Petri games with at least two system players, one environment player, and good markings and bad markings as global winning condition. Second, we prove that it is undecidable whether a winning strategy exists in Petri games with at least one environment player, two players where each of the two players can change between being a system player and an environment player, and good markings as global winning condition.

In the first section of Chapter 4, we recall the root cause for undecidability in the synchronous setting of Pnueli and Rosner. In the remainder of the chapter, good markings as global winning condition are used to simulate this undecidable synchronous setting in the asynchronous setting of Petri games. In the next section, we give an intuition on how this simulation leads to undecidability and a general overview of the first undecidability result. This includes an introduction to the Post correspondence

problem (PCP), how strategies for a Petri game can represent solutions to the PCP, and how good markings can be used to focus on linear firing sequences to check solutions to the PCP for correctness.

In the subsequent section, we formalize this and present the proofs of both the first and the second undecidability result. The second undecidability result is obtained by a reduction from the first undecidability result. Here, bad markings from the first undecidability result are encoded by all players repeatedly becoming environment players without restricting the possible solutions to the PCP, output by the system players. To conclude the chapter, we discuss how these two undecidability results transfer to control games. Furthermore, we emphasize the difference between good markings used in this thesis and final markings used more commonly in control games.

1.9.2. Part II: Bounded Synthesis

Bounded synthesis allows us to find winning strategies in Petri games where the limitation to one environment player or one system player cannot be met [Fin15]. This includes the case of Petri games with multiple system players and with multiple environment players. In a nutshell, bounded synthesis increases a bound on the size of the unfolding and encodes the existence of a winning strategy for the system players in the bounded unfolding as a QBF until a winning strategy for the system players is found. Bounded synthesis produces winning strategies of minimal size but cannot disprove that a winning strategy for the system players exists. We extend previous work on bounded synthesis in the following ways.

In Chapter 5, we present the sequential encoding for bounded synthesis for Petri games via the following steps: In the first section, we outline a motivating example of two robots with independent sources of error. This exemplifies the necessity of solving Petri games with multiple system players and multiple environment players. In the subsequent section, we introduce bounded unfoldings and bounded strategies as the formal foundations for bounded synthesis for Petri games. Afterward, we introduce formal algorithms to obtain bounded unfoldings depending on whether the graph of reachable markings of the Petri game contains loops or not. For the case without loops, we deploy McMilian's unfolding algorithm [McM95]. For the case with loops, we utilize the explicit bound to obtain finite prefixes of the unfolding, which is inspired by using these prefixes in model checking [KKV03]. In the next section, we recall the sequential encoding for bounded synthesis for Petri games with bad markings as global winning condition [Fin15] and present some optimizations. Next, we formalize the sequential encoding for bounded synthesis for Petri games with good markings as global winning condition and explain how the concurrency between players is represented by all sequential firing sequences. To conclude the chapter, we present how bounded strategies are obtained from the output of the QBF solver for satisfiable formulas and how counterexamples from the QBF solver for unsatisfiable formulas can guide the development of a Petri game.

In Chapter 6, we showcase, for bad places as local winning condition, how the sequential encoding can be optimized to the true concurrent encoding for bounded syn-

thesis for Petri games by the following steps: In the first section, we present a motivating example of a production line in a factory. This exemplifies when the sequential encoding represents too many and too difficult problems We also highlight how this can be circumvented by the true concurrent encoding. In the following two sections, we introduce true concurrency in Petri games by recalling strategies for the system players, defining strategies for the environment players, and introducing the true concurrent flow semantics for Petri games. In a nutshell, the true concurrent flow semantics defines when and how concurrent transitions can be fired in one step. Afterward, we present the QBF encoding of true concurrency in Petri games via the true concurrent flow semantics. To conclude the chapter, we present how both the sequential encoding and the true concurrent encoding are implemented in our tool AdamSYNT [Ada20], which also includes a web interface [GHY21]. Furthermore, our experimental evaluation shows that the true concurrent encoding outperforms the sequential encoding on an extensive set of benchmarks.

Chapter 2

Foundations

In this chapter, we introduce the theoretical foundations and fix the notations for the remainder of this thesis. We introduce Petri nets [NPW81; Rei85; Old91] and Petri games [FO17; FG17] with illustrative examples. Petri games are played on an underlying Petri net as the game arena. Our presentation of the definitions follows [BFH19a]. We start with a short introduction to multisets as they are used in the definition of Petri nets and in the definition of Petri games to allow for several tokens in places.

Multisets can be thought of as an extension of sets allowing for repetitions of objects. Each object in a set is either in the set or not, whereas each object in a multiset can additionally occur repeatedly. In both sets and multisets, the order of the objects in the set or multiset does not matter.

Definition 1 (Multisets)

A *multiset* M over a set S is a function $M : S \to \mathbb{N}$.

We write $s \in M$ for the *membership* of object s in a multiset M if $M(s) > 0$. A *set* is a $\{0,1\}$-valued multiset, and vice versa. We use notations such as $\{a : 1, b : 2, c : 3\}$ to represent multisets. For example, the notation $\{a : 1, b : 2, c : 3\}$ encodes a multiset containing element a once, element b twice, element c thrice, and no other elements. Often, we abbreviate expressions such as $a : 1$ by a in these notations for elements occurring once in the multiset. The ordering of the elements in these notations is arbitrary.

We introduce some notations for multisets that might, in parts, look similar to notations for sets: The *empty multiset* $\emptyset$ is defined as $\emptyset(s) = 0$ for all $s \in S$. For two multisets M and N over S, the *multiset inclusion* $M \subseteq N$ is defined as $\forall s \in S : M(s) \leq N(s)$, the *multiset addition* $M + N$ is defined as $(M + N)(s) = M(s) + N(s)$ for all $s \in S$, the *multiset difference* $M - N$ is defined as $(M - N)(s) = \max(0, M(s) - N(s))$ for all $s \in S$, the *multiset intersection* $M \cap N$ is defined as $(M \cap N)(s) = \min(M(s), N(s))$ for all $s \in S$, and the *multiset union* $M \cup N$ is defined as $(M \cup N)(s) = \max(M(s), N(s))$ for all $s \in S$.

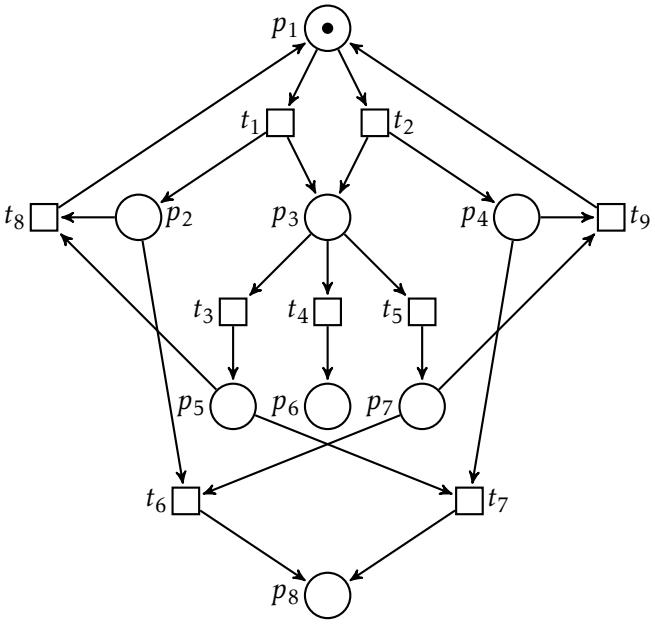


Figure 2.1.: An exemplary Petri net is depicted. It has eight places p_i with $i \in \{1,\dots,8\}$ and nine transitions t_j with $j \in \{1,\dots,9\}$. The initial marking puts one token in place p_1. The weights of all arcs are one. Therefore, they are omitted. When transition t_1 fires, one token is removed from the single place p_1 in t_1's precondition, and one token each is added to the places p_2 and p_3 in t_1's postcondition. For every reachable marking, each place can contain at most one token. The three markings $\{p_2, p_6\}$, $\{p_4, p_6\}$ and $\{p_8\}$ are both reachable and final. In total, this Petri net has ten reachable markings.

2.1. Petri Nets

We begin with the definition of Petri nets because they constitute the underlying game arena of Petri games.

Definition 2 (Petri nets [NPW81; Rei85; Old91])
A *Petri net* or simply *net* $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, In)$ consists of

- the disjoint sets of *places* $\mathcal{P}$ and of *transitions* $\mathcal{T}$,
- the *flow relation* $\mathcal{F}$ as a multiset over $(\mathcal{P} \times \mathcal{T}) \cup (\mathcal{T} \times \mathcal{P})$,
- and the *initial marking* In as a multiset over $\mathcal{P}$.

Generally speaking, Petri nets define the removal and addition of tokens from and to places according to transitions, starting from the distribution of tokens given by the initial marking. In Figure 2.1, we illustrate Petri nets and the formalisms introduced in the following with an example. Places are depicted as *circles* and transitions as *boxes*.

The flow relation defines the *arcs* and their *weight* w from places to transitions for pairs from $\mathcal{P} \times \mathcal{T}$ and from transitions to places for pairs from $\mathcal{T} \times \mathcal{P}$. For a place $p \in \mathcal{P}$ and a transition $t \in \mathcal{T}$, there is an arc from p to t annotated with weight w if $w = \mathcal{F}(p,t)$ and $\mathcal{F}(p,t) > 0$, and there is an arc from t to p annotated with weight w if $w = \mathcal{F}(t,p)$ and $\mathcal{F}(t,p) > 0$. When depicting Petri nets, weights equal to one can be omitted, as can be seen in Figure 2.1. Arcs are depicted by *arrows*. Arcs in both directions between a place and a transition having the same weight are depicted by double-headed arrows. When these arcs have different weights, then they are depicted by two arrows.

States of Petri nets are represented by multisets over $\mathcal{P}$, called *markings*. A marking M is represented by putting $M(p)$ *tokens* in every place $p \in \mathcal{P}$. Tokens are depicted as *dots* inside of places. The *initial marking* of a Petri net is In.

For a place p, the *precondition* is the set $pre(p) = \{t \in \mathcal{T} \mid \mathcal{F}(t,p) > 0\}$ and the *postcondition* is the set $post(p) = \{t \in \mathcal{T} \mid \mathcal{F}(p,t) > 0\}$. For a transition t, the *precondition* is the multiset $pre(t)$ over $\mathcal{P}$ defined as $pre(t)(p) = \mathcal{F}(p,t)$ for all $p \in \mathcal{P}$ and the *postcondition* is the multiset $post(t)$ over $\mathcal{P}$ defined as $post(t)(p) = \mathcal{F}(t,p)$ for all $p \in \mathcal{P}$.

If it is unclear to which Petri net $\mathcal{N}$ a precondition or a postcondition is referring, then we annotate the precondition or the postcondition with $\mathcal{N}$, i.e., we write $pre^{\mathcal{N}}(x)$ and $post^{\mathcal{N}}()$, for x being a place or a transition. By convention, superscripted names of Petri nets are also used for the components of the net, e.g., $\mathcal{N}^U = (\mathcal{P}^U, \mathcal{T}^U, \mathcal{F}^U, In^U)$. Here, we write $pre^U(x)$ and $post^U(x)$ as abbreviation for $pre^{\mathcal{N}^U}(x)$ and $post^{\mathcal{N}^U}(x)$.

A transition t is *enabled* at a marking M if $pre(t) \subseteq M$ (denoted by $M[t\rangle$), i.e., every place in M contains at least as many tokens as required by the precondition of t. If no transition is enabled from a marking M, then we call it *final*. An enabled transition t can *fire* from a marking M resulting in the successor marking $M' = (M - pre(t)) + post(t)$ (denoted by $M[t\rangle M'$), i.e., tokens for the precondition of t are removed from M and tokens for the postcondition of t are afterward added to obtain M'. For markings M and M', we write $M[t_0, \ldots, t_{n-1}\rangle M'$ if there exist markings $M_0, \ldots, M_n$ such that $M_0 = M$, $M_n = M'$, and $M_i[t_i\rangle M_{i+1}$ for all $0 \leq i < n$. The set of *reachable markings* of a Petri net $\mathcal{N}$ is defined as $\mathcal{R}(\mathcal{N}) = \{M \mid \exists n \in \mathbb{N}, t_0, \ldots, t_{n-1} \in \mathcal{T} : In[t_0, \ldots, t_{n-1}\rangle M\}$.

We call elements x in $\mathcal{P} \cup \mathcal{T}$ *nodes*. We say that a Petri net $\mathcal{N}$ is *finite* if the set of nodes of $\mathcal{N}$ is finite. For some $k \in \mathbb{N}$, a Petri net $\mathcal{N}$ is *k-bounded* if $M(p) \leq k$ holds for all reachable markings $M \in \mathcal{R}(\mathcal{N})$ and all places $p \in \mathcal{P}$. A Petri net is *bounded* if it is k-bounded for some given k; otherwise it is *unbounded*. A Petri net is *safe* if it is 1-bounded. For a finite k-bounded Petri net, we restrict the flow relation from multisets over the natural numbers $\mathbb{N}$ to multisets over $\{0, \ldots, k\}$ by removing transitions that can never be enabled. A Petri net $\mathcal{N}'$ is a *subnet* of a Petri net $\mathcal{N}$ (denoted by $\mathcal{N}' \sqsubseteq \mathcal{N}$) if $\mathcal{P}' \subseteq \mathcal{P}$, $\mathcal{T}' \subseteq \mathcal{T}$, $In' = In$, and $\mathcal{F}' = \mathcal{F} \upharpoonright (\mathcal{P}' \times \mathcal{T}') \cup (\mathcal{T}' \times \mathcal{P}')$, i.e., the flow relation is restricted to the remaining places and transitions. We enforce $In' = In$ to maintain all players when later defining strategies for Petri games.

For nodes x and y, we write $x \lessdot y$ if $x \in pre(y)$, i.e., if there is an arc from x to y. With $\leq$, we denote the reflexive, transitive closure of $\lessdot$. The *causal past* of node x is $past(x) = \{y \mid y \leq x\}$. Nodes x and y are *causally related* if $x \leq y$ or $y \leq x$. They are *in conflict* (denoted by $x \sharp y$) if there exists a place $p \in \mathcal{P} \setminus \{x, y\}$ and two distinct transitions

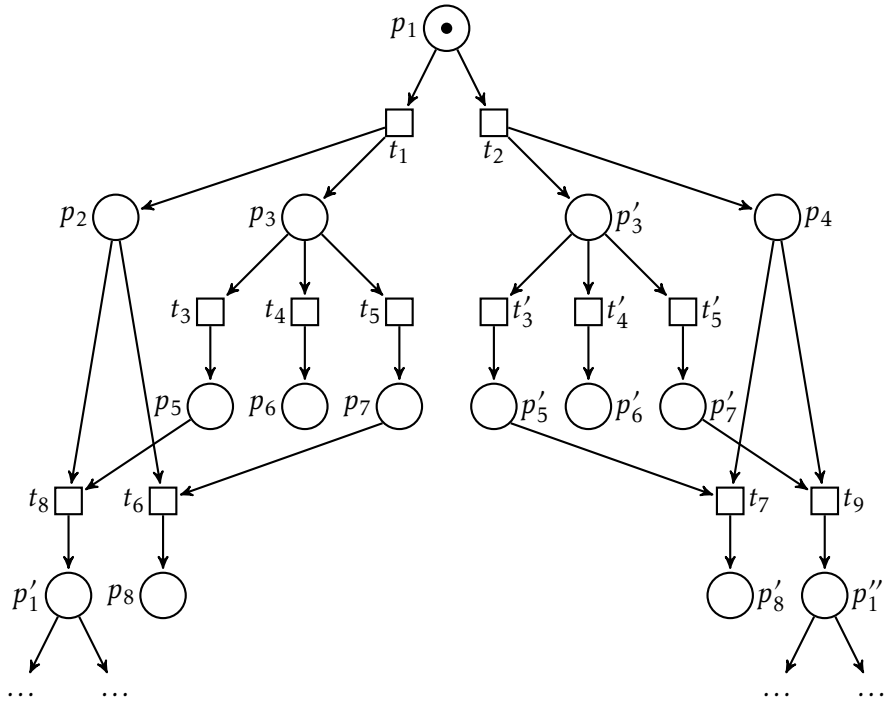


Figure 2.2.: The unfolding of the example Petri net from Figure 2.1 is depicted. Place p_3 in the original net is split into two places p_3 and p'_3 depending on whether transition t_1 or transition t_2 fired. This ensures that places p_3 and p'_3 in the unfolding have exactly one ingoing transition. Both places p_3 and p'_3 have distinct transitions and places to follow the behavior in the original net. Place p_8 in the original net is split into two places p_8 and p'_8 depending on whether transition t_6 or transition t_7 fired. From places p'_1 and p''_1, a copy of the depicted net follows recursively with unique names using the prime symbol $'$ for places and transitions. This fulfills the requirements posed by the definition of an occurrence net. The corresponding homomorphism removes the prime symbols of places and transitions. The unfolding is of infinite length and of infinite width. When the process of copying the net is stopped at one place or at several places, then a branching process is obtained.

$t_1, t_2 \in post(p)$ such that $t_1 \leq x$ and $t_2 \leq y$, i.e., nodes x and y can be reached exiting place p by two different transitions. Node x is in *self-conflict* if $x \sharp x$. We call x and y *concurrent* if they are neither causally related nor in conflict.

2.1.1. Occurrence Nets, Branching Processes, and Unfoldings

To represent the occurrences of transitions with both their causal dependency and conflicts (nondeterministic choices), we give the formal definition of occurrence nets, branching processes, and unfolding for Petri nets. An illustration of these concepts, which are introduced formally in the following, can be found in Figure 2.2.

Definition 3 (Occurrence nets)
An *occurrence net* is a Petri net $\mathcal{N}$, where

- the precondition and postcondition of all transitions are sets of places instead of multisets over places,
- the initial marking is a set of places and contains exactly the places without ingoing transitions, i.e., $\forall p \in \mathcal{P} : p \in In \Leftrightarrow |pre(p)| = 0$,
- all other places have exactly one ingoing transition, i.e., $\forall p \in \mathcal{P} \setminus In : |pre(p)| = 1$,
- $\leq$ is *well-founded*, i.e., starting from any given node, no infinite path following the inverse flow relation exists, and
- no transition is in self-conflict.

Note that an occurrence net is a safe net, i.e., places contain at most one token for all reachable markings in the net. If nodes $x \neq y$ of an occurrence net are in conflict, then they are mutually exclusive, i.e., there is a nondeterministic choice between x and y.

Definition 4 (Homomorphisms)
A *homomorphism* from $\mathcal{N}^1$ to $\mathcal{N}^2$ is a function $\lambda : \mathcal{P}^1 \cup \mathcal{T}^1 \to \mathcal{P}^2 \cup \mathcal{T}^2$ that

- respects node types, i.e., $\lambda(\mathcal{P}^1) \subseteq \mathcal{P}^2 \wedge \lambda(\mathcal{T}^1) \subseteq \mathcal{T}^2$, and
- is structure-preserving on transitions, i.e,
  $\forall t \in \mathcal{T}^1 : \lambda(pre^1(t)) = pre^2(\lambda(t)) \wedge \lambda(post^1(t)) = post^2(\lambda(t))$.

If λ additionally agrees on the initial markings, i.e., $\lambda(In^1) = In^2$, then λ is called an *initial homomorphism*.

A branching process [Eng91; MMS96; EH08] describes parts of possible behaviors of a Petri net. We use the *individual token semantics* [GR83].

Definition 5 (Branching processes [Eng91; MMS96; EH08])
An *(initial) branching process* of a Petri net $\mathcal{N}$ is a pair $\iota = (\mathcal{N}^\iota, \lambda^\iota)$ where

- $\mathcal{N}^\iota$ is an occurrence net and
- $\lambda^\iota : \mathcal{P}^\iota \cup \mathcal{T}^\iota \to \mathcal{P} \cup \mathcal{T}$ is an initial homomorphism from $\mathcal{N}^\iota$ to $\mathcal{N}$ that is injective on transitions with the same precondition, i.e.,
  $\forall t, t' \in \mathcal{T}^\iota : (pre^\iota(t) = pre^\iota(t') \wedge \lambda^\iota(t) = \lambda^\iota(t')) \Rightarrow t = t'$.

Intuitively, whenever a node can be reached on two distinct paths in a Petri net, then it is split up in the branching process of the Petri net. The initial homomorphism λ^ι can be thought of as a label of the copies into nodes of the Petri net. The injectivity condition avoids additional unnecessary splits: Each transition must either be labeled differently or occur from different preconditions.

Definition 6 (Unfoldings)
The *unfolding* $\iota_U = (\mathcal{N}^U, \lambda^U)$ of a Petri net $\mathcal{N}$ is a maximal branching process: Whenever there is a set of pairwise concurrent places C such that $\lambda^U(C) = pre^{\mathcal{N}}(t)$ for some transition $t \in \mathcal{T}$, then there exists $t' \in \mathcal{N}^U$ with $\lambda^U(t') = t$ and $pre^U(t') = C$.

The unfolding represents every possible behavior of a Petri net $\mathcal{N}$. Finite Petri nets may have infinite unfoldings due to loops in the graph of reachable markings.

Let $\iota_1 = (\mathcal{N}^1, \lambda^1)$ and $\iota_2 = (\mathcal{N}^2, \lambda^2)$ be two branching processes of the same Petri net. A homomorphism from ι_1 to ι_2 is a homomorphism h from $\mathcal{N}^1$ to $\mathcal{N}^2$ such that $\lambda^1 = \lambda^2 \circ h$. Notice that the homomorphism h does not necessarily need to exist for two arbitrary branching processes of $\mathcal{N}$. The homomorphism h is called *initial* if $h(In^1) = In^2$ holds. The homomorphism h is called an *isomorphism* if it is a bijection. Two branching processes ι_1 and ι_2 are *isomorphic* if there exists an initial isomorphism from ι_1 to ι_2. A branching process ι_1 *approximates* a branching process ι_2 if there exists an initial injective homomorphism from ι_1 to ι_2. A branching process ι_1 is a *subprocess* of a branching process ι_2 if ι_1 approximates ι_2 with the identity on $\mathcal{P}^1 \cup \mathcal{T}^1$ as the homomorphism. Thus, $\mathcal{N}^1 \sqsubseteq \mathcal{N}^2$ and $\lambda^1 = \lambda^2 \restriction (\mathcal{P}^1 \cup \mathcal{T}^1)$. The Petri net $\mathcal{N}^1$ is a subnet of the Petri net $\mathcal{N}^2$, and the homomorphism λ^1 is the restriction of the homomorphism λ^2 to the remaining places $\mathcal{P}^1$ and the remaining transition $\mathcal{T}^1$. If ι_1 approximates ι_2, then ι_1 is isomorphic to a subprocess of ι_2. In the literature [Eng91], it is shown that the unfolding $\iota_U = (\mathcal{N}^U, \lambda^U)$ of a net is unique up to isomorphism and that every initial branching process ι of $\mathcal{N}$ approximates ι_U. Thus, ι is a subprocess of ι_U up to isomorphism.

2.2. Petri Games

In this section, we introduce Petri games, strategies for Petri games, and when these strategies are winning for the different winning conditions.

Definition 7 (Petri games [FO17; FG17])
A *Petri game* or simply *game* is a tuple $\mathcal{G} = (\mathcal{P}_S, \mathcal{P}_E, \mathcal{T}, \mathcal{F}, In, \mathcal{W})$. The places of the *underlying Petri net* $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, In)$ are partitioned into *system places* $\mathcal{P}_S$ and *environment places* $\mathcal{P}_E$. The *winning condition* is given by the symbol $\mathcal{W}$ as the set of *bad places* $\mathcal{P}_B \subseteq \mathcal{P}$, the set of *bad markings* $\mathcal{M}_B \subseteq \mathcal{R}(\mathcal{N})$, the set of *good markings* $\mathcal{M}_G \subseteq \mathcal{R}(\mathcal{N})$, or the pair of disjoint sets of *good and bad markings* $(\mathcal{M}_G, \mathcal{M}_B) \in \mathbb{P}(\mathcal{R}(\mathcal{N})) \times \mathbb{P}(\mathcal{R}(\mathcal{N}))$.

We call tokens on system places *system players* and tokens on environment places *environment players*. The Petri game is played by firing transitions in the underlying Petri net. We say that players *synchronize* when a joint transition fires. Intuitively, a strategy controls the behavior of system players by deciding which transitions to allow. Environment players are uncontrollable and transitions only dependent on environment players cannot be restricted by a strategy. We illustrate Petri games and the formalisms introduced in the following with an example in Figure 2.3. This figure has been published in [HM19b]. We depict Petri games as Petri nets and color system places gray

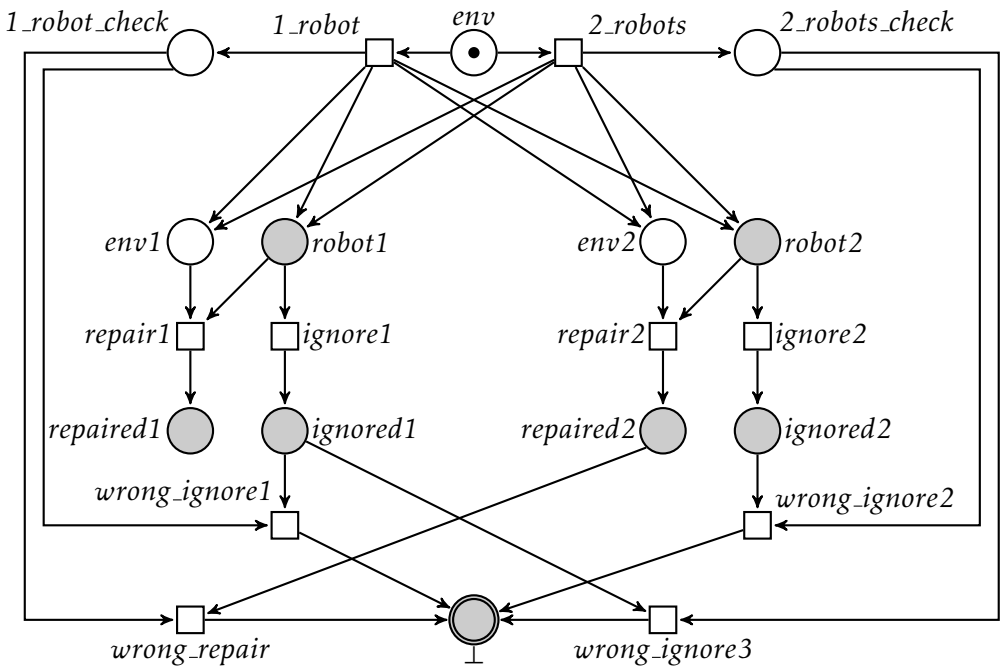


Figure 2.3.: An exemplary Petri game is depicted. It specifies a production line where two robots can repair a product. The product either requires repair by only one or by both robots. The system places *robot1* and *robot2* represent the robots. They are spawned after the environment player in environment place *env* decides by firing either transition *1_robot* or transition *2_robots* whether one or both robots need to repair the product. The decision is stored in the environment places *1_robot_check* and *2_robots_check*. Each robot can either *repair* or *ignore* the product. Afterward, the bad place $\bot$ can be reached when too few robots repaired the product or both robots repaired the product, although repair by only one robot is necessary. The corresponding unfolding and winning strategy are presented in Figure 2.4.

and environment places white. Transitions and tokens are depicted in Petri games as they are depicted in Petri nets.

We define strategies for the system players.

Definition 8 (Strategies)

A *strategy* for a Petri game $\mathcal{G}$ is a branching process $\sigma = (\mathcal{N}^\sigma, \lambda^\sigma)$ satisfying *justified refusal*: If there is a set of pairwise concurrent places C in $\mathcal{N}^\sigma$ and a transition $t \in \mathcal{T}$ with $\lambda^\sigma[C] = pre^{\mathcal{N}}(t)$, then there either is a transition $t' \in \mathcal{T}^\sigma$ with $\lambda^\sigma(t') = t$ and $C = pre^\sigma(t')$ or there is a system place $p \in C \cap (\lambda^\sigma)^{-1}[\mathcal{P}_S]$ with $t \notin \lambda^\sigma[post^\sigma(p)]$. Furthermore, the strategy σ has to be *deterministic*: For every reachable marking M of σ and system place $p \in M$, there is at most one transition from $post^\sigma(p)$ enabled in M.

As a branching process describes subsets of the behavior of a Petri net, a strategy is a restriction of possible transitions in the Petri game. Justified refusal enforces that only system places can prohibit transitions based on their causal past. From every situation in the game, a transition possible in the underlying net is either in the strategy or there is a system place that never allows it. Transitions involving only environment places are always possible. As a branching process encodes the causal memory of places, system players in the strategy base their decisions on their causal memory.

The requirement of justified refusal for the strategy σ can be formalized by $\forall t \in \mathcal{T}^U : t \notin \mathcal{T}^\sigma \wedge pre^\sigma(t) \subseteq \mathcal{P}^\sigma \Rightarrow (\exists p \in pre^\sigma(t) \cap \mathcal{P}_S^\sigma : \forall t' \in post^U(p) : \lambda^U(t) = \lambda^U(t') \Rightarrow t' \notin \mathcal{T}^\sigma)$, i.e., every transition from the unfolding which can be enabled in the strategy is either in the strategy or a system place forbids the transition and all its outgoing transitions which are based on the same transition in the original Petri game.

The requirement of deterministic decisions for the strategy σ can be formalized by the following formula $\forall M \in \mathcal{R}(\mathcal{N}^\sigma) : \forall p \in M \cap \mathcal{P}_S^\sigma : \exists^{\leq 1} t \in post^\sigma(p) : pre^\sigma(t) \subseteq M$. Notice that $post^\sigma(p)$ can contain more than one transition as long as at most one of them is enabled at the same reachable marking. This makes it possible that the environment player decides between different branches of the Petri game, and the system player, later on, reacts to every decision.

2.2.1. Winningness of Strategies

We define winningness for the strategies for the system players in a Petri game.

For safety winning conditions, we need the requirement of deadlock-avoidance: A strategy is *deadlock-avoiding* if, for every final, reachable marking M in the strategy, $\lambda^\sigma[M]$ is final as well, i.e., the strategy is only allowed to terminate if the underlying Petri net does so. The requirement of deadlock-avoidance can be formalized by the following formula $\forall M \in \mathcal{R}(\mathcal{N}^\sigma) : \exists t_U \in \mathcal{T}^U : pre^U(t_U) \subseteq M \Rightarrow \exists t_\sigma \in \mathcal{T}^\sigma : pre^\sigma(t_\sigma) \subseteq M$.

Now, we can define winningness for the local winning condition of bad places.

Definition 9 (Winning for bad places)
A strategy σ is *winning for bad places* $\mathcal{W} = \mathcal{P}_B \subseteq \mathcal{P}$ if it is deadlock-avoiding and no reachable marking in σ contains a place corresponding to a bad place, i.e., $\forall M \in \mathcal{R}(\mathcal{N}^\sigma) :$
$(\exists t_U \in \mathcal{T}^U : pre^U(t_U) \subseteq M \Rightarrow \exists t_\sigma \in \mathcal{T}^\sigma : pre^\sigma(t_\sigma) \subseteq M) \wedge (\forall p_{bad} \in \mathcal{P}_B : p_{bad} \notin \lambda^\sigma[M])$.

Similarly, we define winningness for the global winning condition of bad markings.

Definition 10 (Winning for bad markings)
A strategy σ is *winning for bad markings* $\mathcal{W} = \mathcal{M}_B \subseteq \mathcal{R}(\mathcal{N})$ if it is deadlock-avoiding and no reachable marking in σ corresponds to a bad marking, i.e., $\forall M \in \mathcal{R}(\mathcal{N}^\sigma) :$
$(\exists t_U \in \mathcal{T}^U : pre^U(t_U) \subseteq M \Rightarrow \exists t_\sigma \in \mathcal{T}^\sigma : pre^\sigma(t_\sigma) \subseteq M) \wedge (\forall M_{bad} \in \mathcal{M}_B : M_{bad} \neq \lambda^\sigma[M])$.

For liveness winning conditions, we need so-called covering firing sequences, which are based on so-called maximal plays: A *play* $\pi = (\mathcal{N}^\pi, \lambda^\pi)$ is a subprocess of a strategy $\sigma = (\mathcal{N}^\sigma, \lambda^\sigma)$ such that $\forall p \in \mathcal{P}^\pi : |post(p)| \leq 1$. A play is *maximal* when, for each set of

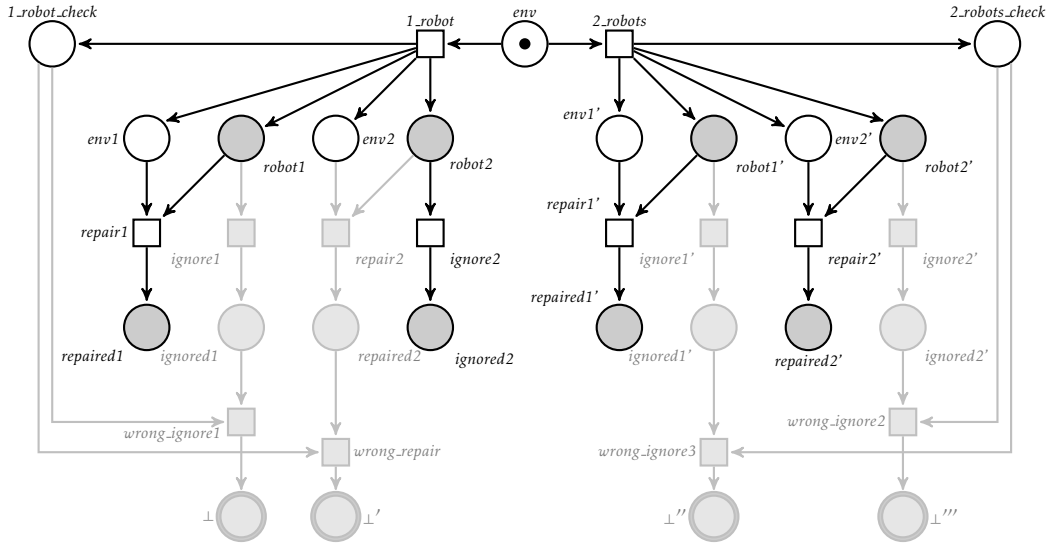


Figure 2.4.: The unfolding and the winning strategy for the exemplary Petri game in Figure 2.3 are depicted. The unfolding includes the grayed-out parts, whereas the winning strategy does not include the grayed-out parts. Copies of places and transitions due to splits are primed. The winning strategy realizes the following behavior: When the product requires repair by only one robot (cf. left-hand side of the figure), the first robot repairs the product while the second robot ignores the product. When the product requires repair by two robots (cf. right-hand side of the figure), both robots repair it. The strategy fulfills justified refusal because it only removes outgoing transitions of system places. By doing this, transitions with the environment player participating become unreachable and are removed. The strategy is deterministic because every system place has at most one outgoing transition. The strategy is deadlock-avoiding because no other enabled transitions exist. Because the strategy further avoids the bad places $\bot$, it is winning.

pairwise concurrent places C in $\mathcal{N}^\pi$ such that $C = pre^\sigma(t)$ for some transition $t \in \mathcal{T}^\sigma$, a place $p \in C$ and a transition $t' \in \mathcal{T}^\pi$ exist such that $t' \in post^\pi(p)$. A *covering firing sequence* of a play π is a sequence of subsequent markings and fired transitions such that each place and transition of π occurs.

Now, we can define winningness for the global winning condition of good markings.

Definition 11 (Winning for good markings)
A strategy σ is *winning for good markings* $\mathcal{W} = \mathcal{M}_G \subseteq \mathcal{R}(\mathcal{N})$ if, for all complete firing sequences $t_0 t_1 t_2 \ldots$ of all maximal plays π of σ with $M_0 = In^\pi$ and $M_0[t_0\rangle M_1[t_1\rangle M_2[t_2\rangle \ldots$, there exists $i \geq 0$ such that $\lambda^\pi[M_i] \in \mathcal{M}_G$.

In a similar manner, we define winningness for the global winning condition of good and bad markings.

Definition 12 (Winning for good and bad markings)
A strategy σ is *winning for good and bad markings* $\mathcal{W} = (\mathcal{M}_G, \mathcal{M}_B) \in \mathbb{P}(\mathcal{R}(\mathcal{N})) \times \mathbb{P}(\mathcal{R}(\mathcal{N}))$ if, for all complete firing sequences $t_0 t_1 t_2 \ldots$ of all maximal plays π of σ with $M_0 = In^\pi$ and $M_0[t_0\rangle M_1[t_1\rangle M_2[t_2\rangle \ldots$, there exists $i \geq 0$ such that

$$\lambda^\pi[M_i] \in \mathcal{M}_G \wedge \forall 0 \leq j < i : \lambda^\pi[M_j] \notin \mathcal{M}_B.$$

Note that terminating in a final marking as another possible winning condition is different from reaching a good marking because players are not required to terminate in a good marking. The corresponding unfolding and winning strategy for the Petri game with bad places from Figure 2.3 are given in Figure 2.4. The winning strategy has been published in [HM19b].

Example 2.2.1. The example Petri net from Figure 2.1 can be turned into a Petri game by defining place p_3 as the only system place, by defining all other places as environment places, and by defining the places p_6 and p_8 as bad places. A winning strategy is obtained from the unfolding depicted in Figure 2.2 by only choosing transition t with $\lambda(t) = t_3$ at all system places p that satisfy $t_1 \in \lambda(pre(p))$ and only choosing transition t with $\lambda(t) = t_5$ at all system places p that satisfy $t_2 \in \lambda(pre(p))$. Thereby, transition t_1 is immediately answered by transition t_3, and transition t_2 is immediately answered by transition t_5. Not chosen transitions from system places and all thereby unreachable places and transitions are removed. The bad places are avoided because the system player in system place p_3 never chooses transition t_4 and, for the two pairs of places from the preconditions of transitions t_6 and t_7, at least one place of each pair does not contain a token for every reachable marking. △

We define the restrictions on the number of system players and environment players used in the decidability results for Petri games. A Petri game has *a bounded number of players* if and only if it is played on a bounded Petri net, i.e., the number of tokens in every place of every reachable marking of the Petri net is not larger than a bound k. This translates to *a bounded number of system (environment) players* if the number of tokens in all system (environment) places is less or equal to a bound k for all reachable markings in the underlying Petri net. Notice that players can spawn and terminate during the play of a Petri game. Therefore, the same token can represent a new player for each iteration of a loop in a Petri game. The number of players existing at the same time in the Petri game is still bounded in this case. A Petri game has exactly (at most) a certain *fixed number of system (environment) players*, e.g., one, if the sum of tokens in all system (environment) places is exactly (at most) this number, e.g., one, for all reachable markings of the underlying Petri net.

The existence of a winning strategy for the system players in Petri games with a bounded number of system players, one environment player, and bad places as *local* winning condition is EXPTIME-complete [FO17]. The existence of a winning strategy for the system players in Petri games with a bounded number of environment players, one system player, and bad markings as *global* winning condition is

EXPTIME-complete [FG17]. In Chapter 3, we show that the existence of a winning strategy for the system players in Petri games with a bounded number of system players, at most one environment player, and bad markings as *global* winning condition is decidable [FGHO22; FGHO21]. Due to the formal connection [BFH19a; BFH19b; Beu19] between Petri games and *control games* [GGMW13] based on *asynchronous automata* [Zie87] (cf. Section 1.7), it is possible to transfer decidability results in acyclic communication architectures [GGMW13], which were originally obtained for control games, to Petri games.

2.3. Different Formalisms

We distinguish Petri games from formalisms concerned with the synthesis of Petri nets or with games played on Petri nets. In Section 1.7, we present related work concerning synthesis in general and *control games* [GGMW13] based on *asynchronous automata* [Zie87] specifically.

For a given behavioral description of a concurrent system, the *Petri net synthesis problem* transforms it into a structural description [ER90; DR96; BBD15]. The given behavioral description can be transition systems, formal languages, or execution traces. The obtained structural description can be Petri nets in all their flavors, including Petri nets with priorities and inhibitor arcs. The difference with Petri games is that only Petri games define the synthesis problem between system players and environment players on a given Petri net, i.e., there is no notion of system players and environment players in the Petri net synthesis problem.

Synthesis and control can be defined based on Petri nets [RSVB03; BDLV05; ABP21]. There are extensions enabling us to include time with continuous variables, as in hybrid automata [JLS16; JLS18], and to synthesize updates in software-defined networks [Did+21]. These approaches solve supervisory control problems or two-player games on the state space created by Petri nets. Thus, they are different from Petri games because Petri games can have multiple system players and multiple environment players with their individual causal memory.

For the synthesis of distributed systems, synchronous processes with shared-variable communication have been a focus of research [PR89a]. Here, *distributed games* represent a general game model [MW03]. As mentioned in Chapter 1, this setting is undecidable in general [PR90] but decidable for rings [KV01] specifically and generally for all architectures where processes can be ordered according to their informedness, i.e., for architectures without so-called information forks [FS05]. These decidability results have non-elementary complexity. Alternating-time temporal logics can also be interpreted over concurrent game structures [AHK02]. These approaches use a separate, static specification of the relative informedness of processes in an architecture. This is in contrast to Petri games, where the informedness of players is linked to causality.

Unfoldings have been utilized in order to connect Petri nets with event structures [NPW81; BF88; Eng91; Old91; MMS96] and to obtain algorithms for reachability. These algorithms construct a finite canonical prefix of the generally infinite unfold-

ing [Esp94; KKV03; EH08; Bon+14]. In Petri games, the unfolding is used to define strategies, their plays, and to represent the causal memory of players. The canonical prefixes are also of interest to define the bounded unfolding for bounded synthesis of Petri games. This will be discussed in Chapter 5. Partial order reduction and true concurrency have been studied thoroughly to speed up the model checking of finite distributed systems [Hel99; Hel02; FG05; MP09].

High-level Petri nets can represent some Petri nets in a more concise way [GL81; Jen92]. Here, components of the same form that occur multiple times, e.g., distributed robots in a factory, can be represented by one expression in the high-level Petri net instead of specifying each component separately. This approach has been extended to the setting of Petri games in the form of *high-level Petri games* [GO19]. To find winning strategies for the system players in high-level Petri games, the reduction with respect to bad places as local winning condition, a bounded number of system players, and one environment player [FO17] has been extended to solve high-level Petri games with the same local winning condition and the same restriction on the number of system players and environment players [GOW20].

Part I.

Decidability

Chapter 3

Decidability of Bad Markings

In this part of the thesis, we contribute three results concerning the (un)decidability of the synthesis of asynchronous distributed reactive systems with causal memory using Petri games. The general decidability or undecidability of this problem is a long-standing open problem [Mus15; FO17]. Our results are concerned with going from local winning conditions [MT01] to global winning conditions [CMT99]. Local winning conditions are limited as they cannot express global properties like mutual exclusion. In the first part of this chapter, we prove that it is decidable whether a winning strategy for the system players exists in Petri games with a bounded number of system players, at most one environment player, and *bad markings* as global winning condition.

In the second part of this chapter, we investigate whether global winning conditions beyond bad markings are decidable. We report on two undecidability results to further underline the significance of our decidability result: First, we prove that it is undecidable whether a winning strategy exists for the system players in Petri games with at least two system players, one environment player, and *good and bad markings* as winning condition. Notice that it is not required to terminate in a good marking. Good markings can be used to simulate the undecidable synchronous setting of Pnueli and Rosner [PR90] in the asynchronous setting of Petri games. This is realized by identifying executions as good if players deviate too much from the synchronous setting. Second, we prove that it is undecidable whether a winning strategy exists for the system players in Petri games with *good markings* and at least three players, out of which one is an environment player and each of the other two can change between being a system and an environment player. Here, bad markings from the first undecidability result are encoded by repeatedly changing all players to environment players.

In this chapter, we prove that it is decidable whether a winning strategy for the system players exists in Petri games with a bounded number of system players, at most one environment player, and *bad markings* as global winning condition. This is achieved by a reduction to a two-player game with complete observation and a Büchi winning condition. In the two-player game, it is encoded that transitions with the environment

player fire as late as possible, i.e., transitions *without* the environment player fire before transitions *with* it. This order of transitions encodes causal memory [FO17]. For every sequential play of the two-player game, we need to check that no bad marking is reached for the different orders of fired concurrent transitions. Here, the causal history of system players can grow infinitely large. We show that the finite causal history of each system player until its last transition with the environment player suffices to find bad markings and that it can be stored finitely.

The general decidability or undecidability of the synthesis problem for asynchronous distributed reactive systems with causal memory is a long-standing question [Mus15; FO17]. With these results, we obtain a clear picture regarding decidability and undecidability for global winning conditions.

From our decidability result and previous work [FG17], we obtain for bad markings as global winning condition that the question of whether the system players have a winning strategy is decidable for Petri games where the number of system players or the number of environment players is at most one and the number of players of the converse type can be bounded by some arbitrary number. For bad markings as global winning condition, this leaves the case of Petri games with two or more system players *and* two or more environment players open.

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

Thus, for the synthesis of asynchronous distributed reactive systems with causal memory, global safety winning conditions are decidable for a large class of such systems, whereas global liveness winning conditions are undecidable for almost all classes of such systems. In the future, one could combine the decidability results for bad markings as global safety winning condition with local liveness specifications per player as in Flow-LTL [FGHO19; FGHO20a].

The key contributions of this chapter are the following:

- We prove that deciding the existence of a winning strategy for the system players in Petri games with a bounded number of system players, one environment player, and *bad markings* as global winning condition is decidable.
- We highlight that backward moves, which represent one key ingredient of the reduction of the decidability result from above, solve an intricate problem in the case of bad places as local winning conditions concerning the requirement of *deterministic strategies* for the system players.
- We encode Petri games with at most one environment player in Petri games with one environment player. Thereby, we prove that the previous decidability result can be extended to Petri games with a bounded number of system players, *at most one environment player*, and bad markings as global winning condition.

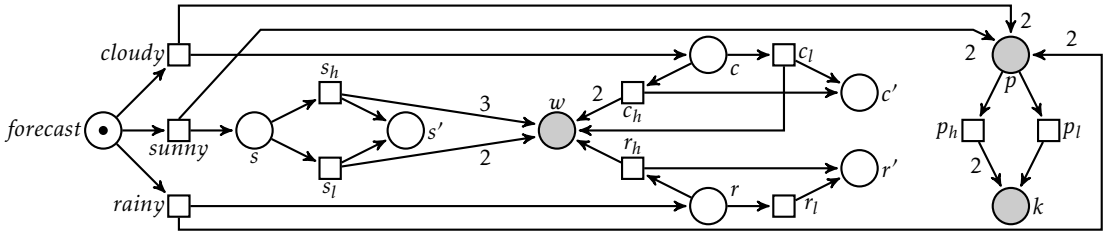


Figure 3.1.: Two power plants observe whether *sunny*, *cloudy*, or *rainy* weather is *forecast*. Depending on the actual weather, renewable sources produce up to three units of energy. The power plants produce one or two units of energy each and have to maintain the total energy production between four and five units of energy because all final markings with different energy production are bad markings.

This chapter is structured as follows: In Section 3.1, we motivate the expressive power of Petri games with bad markings as global winning condition. In Section 3.2, we define Büchi games formally. In Section 3.3, we explain the ideas behind the reduction from Petri games with a bounded number of system players, one environment player, and bad markings as global winning condition to Büchi games. In Section 3.4, we make the reduction from the previous section formal and prove its correctness. In Section 3.5, we highlight that backward moves from the aforementioned reduction solve an intricate problem in the reduction for Petri games with a bounded number of system players, one environment player, and bad places as local winning condition. In Section 3.6, we encode Petri games with at most one environment player as Petri games with exactly one environment player to extend our decidability result to Petri games with a bounded number of system players and at most one environment player, and bad markings as global winning condition.

3.1. Motivating Example

With the example in Figure 3.1, we introduce the intuition behind causal memory and behind bad markings as global winning condition for Petri games. There, we search for a strategy for two power plants, which should react to the energy production of renewable sources based on the weather forecast. For example, after transition *sunny* fires to indicate a sunny forecast, there are two system players in place p (each representing one power plant) and one environment player in place s. Causal memory implies that both system players know what the weather forecast predicts. They do not know whether the actual energy production is high (indicated by transition s_h firing) or low (indicated by transition s_l firing) producing three or two units of energy in place w. Nevertheless, each power plant has to decide whether to produce two or one unit of energy in place k by transition p_h or transition p_l firing.

The two power plants should produce together with the renewable sources either four or five units of energy. Therefore, any final marking resulting in a different energy

production is a bad marking, i.e., the set of bad markings is

$$\{M : \mathcal{P} \to \mathbb{N} \mid (M(k) + M(w) < 4 \vee M(k) + M(w) > 5) \wedge \\ \exists x \in \{s', c', r'\} : (M(x) = 1 \wedge \forall y \in \mathcal{P} \setminus \{k, w, x\} : M(y) = 0)\},$$

where the second line ensures that the marking is final. As discussed in Section 2.2, strategies for the system players are required to be *deadlock-avoiding* for safety winning conditions. This implies for this Petri game that all system players always choose one of their successors. The reasons for this are that there is no infinite behavior in the example and that the system players do not have to react directly to a decision by the environment player, i.e., transitions with an environment place in their precondition do not have a system place in their precondition.

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

3.2. Büchi Games

We introduce *Büchi games* on a finite game arena as representation of reactive synthesis as a game [KV05]. A Büchi game has two players: *Player 0* represents the *system* and *Player 1* represents the *environment*. Both players act on complete information about the game arena and the play so far. To win, Player 0 has to ensure that an accepting state is visited infinitely often. A winning strategy for Player 0 corresponds to a correct-by-construction implementation of the encoded reactive synthesis question. Deciding the existence of such a winning strategy can be performed in polynomial time in the number of edges in the graph [CH12]. Formally, a *Büchi game* $\mathbb{G} = (V, V_0, V_1, I, E, F)$ consists of the finite set of *states* V partitioned into the disjoint sets of states V_0 of Player 0 and of states V_1 of Player 1, the *initial state* $I \in V$, the *edge relation* $E \subseteq V \times V$, and the set of *accepting states* $F \subseteq V$. We assume that all states in a Büchi game have at least one outgoing edge. A *play* is a possibly infinite sequence of states which is constructed by letting Player 0 choose the next state from the successors in E whenever the game is in a state from V_0 and by letting Player 1 choose otherwise. An *initial play* is a play that starts from the initial state. A play is *winning* for Player 0 if it visits at least one accepting state infinitely often. Otherwise, the play is *winning* for Player 1. A *strategy* for Player 0 is a function $f : V^* \cdot V_0 \to V$ that maps plays ending in states of Player 0 to one possible successor according to E. A play *conforms* to a strategy f if all successors of states in V_0 are chosen in accordance with f. A strategy f is winning for Player 0 if all initial plays that conform to f are winning for Player 0.

3.3. Decidability in Petri Games with Bad Markings

We present a reduction from Petri games with a bounded number of system players, one environment player, and bad markings to Büchi games. In the following, we give an intuition for the main concepts of the reduction, before presenting the structure of the Büchi game in the remainder of this section. More details can be found in Section 3.4 and a running example can be found in Figure 3.2.

Petri games use unfoldings, which can be of infinite size, to encode the causal memory of players. By contrast, Büchi games have two players with complete information and a finite number of states. To overcome these differences when encoding Petri games, states in the corresponding Büchi games consist of a representation of the current marking and some additional information. Edges in the Büchi game mostly correspond to a transition firing in the Petri game. We say that a transition fires in the Büchi game when it fires in the encoded Petri game. Concurrency between transitions in the Petri game is encoded by having most possible interleavings in the Büchi game. Some interleavings are left out to encode causal memory of the players in Petri games: Causal memory is simulated in Büchi games by transitions with an environment place in their precondition firing as late as possible at *mcuts* [FO17]. An mcut is a situation in the Petri game where all system players have progressed maximally, i.e., the environment player can choose between all remaining possible transitions. Mcuts can only be defined for Petri games with at most one environment player. States corresponding to mcuts are the only states where Player 1 in the Büchi game makes decisions.

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

Second, we add the *NES-case* to handle system players playing infinitely without synchronizing with the environment player directly in the Büchi game. The abbreviation *NES* stands for *no more environment synchronization* and is necessary when some system players play infinitely but without synchronization with the environment player. In [FO17], this case is called the type-2 case because the situation where each system player either terminates or synchronizes with the environment player infinitely often is implicitly called the type-1 case. In [FO17], the situation equivalent to the NES-case in this chapter can be handled as a preprocessing step, because the local winning condition of bad places is considered. This is impossible for the global winning condition of bad markings, considered in this chapter. Throughout this chapter, the NES-case can be ignored by adding the restriction that each system player either terminates or synchronizes infinitely often with the environment player.

For the NES-case, every system player has a three-valued flag. As long as the system player will terminate or will synchronize with the environment player in the future,

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

3.3.1. States and Initial State in the Büchi Game

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

The decision is either $\top$ or the set of *allowed transitions* by the player. For system players, $\top$ indicates that a decision for a set of allowed transitions is missing and has to be chosen. The representation of the last mcut encodes the last known position of the environment player. There can be at most as many different such positions as there are system players. Thus, a number suffices to identify the last known mcut. Let $\max_S$ be the maximal number of system players in the Petri game which are visible at the same time. The set of *system decision tuples* is $\mathcal{D}_S = \{(id, p, b, T, K) \mid id, K \in \{1, \dots, \max_S\} \wedge p \in \mathcal{P}_S \wedge b \in \{false, true, end\} \wedge (T = \top \vee T \subseteq post(p))\}$, the set of *environment decision tuples* is $\mathcal{D}_E = \{(0, p, false, post(p), 0) \mid p \in \mathcal{P}_E\}$, and the set of all *decision tuples* is $\mathcal{D} = \mathcal{D}_S \cup \mathcal{D}_E$.

Example 3.3.1. In Figure 3.2, a branch of the Büchi game for the Petri game in Figure 3.1 is shown. States with decision tuples with positive NES-status are omitted because no infinite behavior occurs. The initial state v_0 has one decision tuple for the environment player in place *forecast* and empty information for the NES-case and the backward moves. After Player 1 plays the edge for transition *sunny* firing, state v_1 with three decision tuples is reached. The decision tuples for the two system players in place p have $\top$ as decision. There are 16 combinations of decisions by the two system players, out of which four are shown. The first system player always allows transition p_l and the second system player allows no transition in state v_2, only one of the two transitions p_l and p_h in states v_6 and v_8, or both transitions in state v_7. △

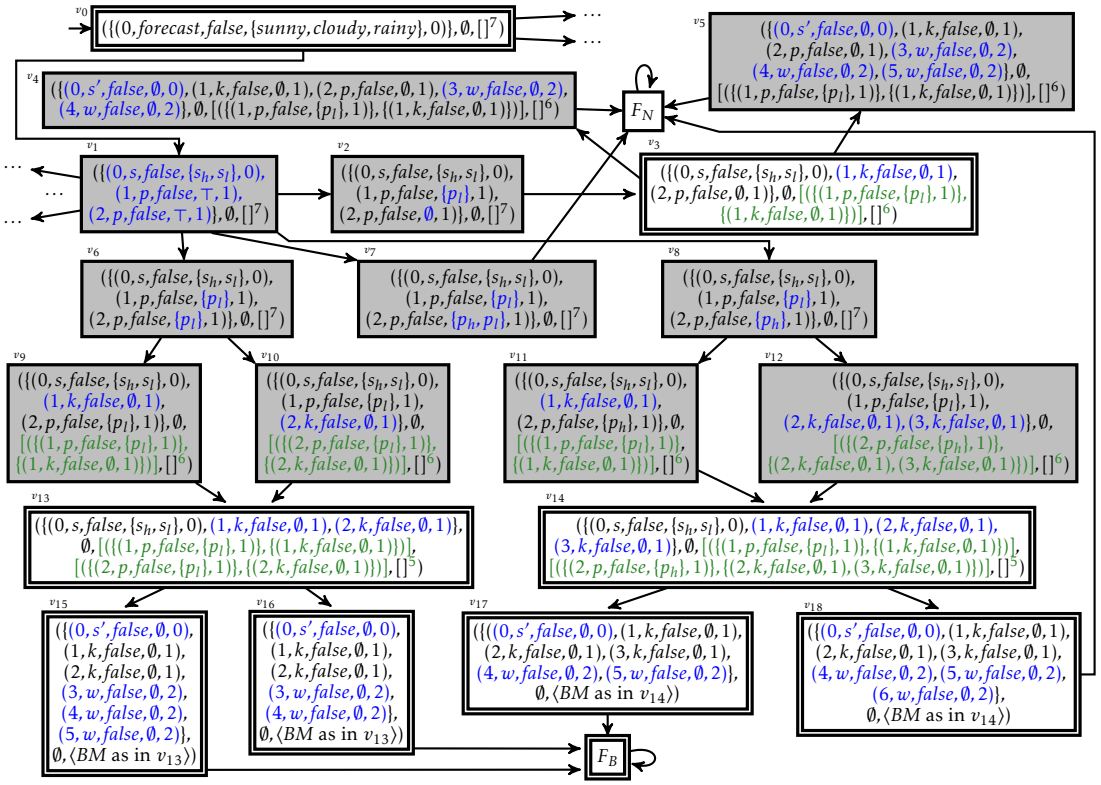


Figure 3.2.: A part of the Büchi game for the Petri game in Figure 3.1 is given. States of Player 0 are gray, states of Player 1 white. Most states are labeled for identification. Double squares are accepting states. Changes from previous states are blue for decision tuples and green for backward moves.

Almost all states in the Büchi game contain decision tuples and additional information for the NES-case and for backward moves. The *states in the Büchi game* are defined as $V = V_{BN} \cup \mathscr{D} \times (\mathcal{P}_S \to \{0,\dots,k\}) \times (\mathcal{B}^*)^{\max_S}$ with $V_{BN} = \{F_B, F_N\}$. Finite winning and losing behavior in the Petri game is represented in the Büchi game by the two unique states F_B and F_N in V_{BN}. A *decision marking* is a set of decision tuples corresponding to a reachable marking in the Petri game such that each identifier occurs at most once. $\mathscr{D}$ is the set of all such decision markings. The next element stores the underlying multiset over system places of the decision marking from the start of the NES-case restricted to system players with positive NES-status. In the NES-case, repeating this multiset proves that the system players with positive NES-status can play infinitely without firing a transition with an environment place in its precondition. This element is the empty multiset if not in the NES-case, i.e., no system player has positive NES-status. More details can be found in Section 3.3.5. $\mathcal{B} : \mathbb{P}(\mathcal{D}_S) \times \mathbb{P}(\mathcal{D}_S)$ is the set of backward moves to detect states corresponding to a bad marking or a nondeterministic decision. The remaining elements are $\max_S$ sequences of backward moves. Each identifier in a

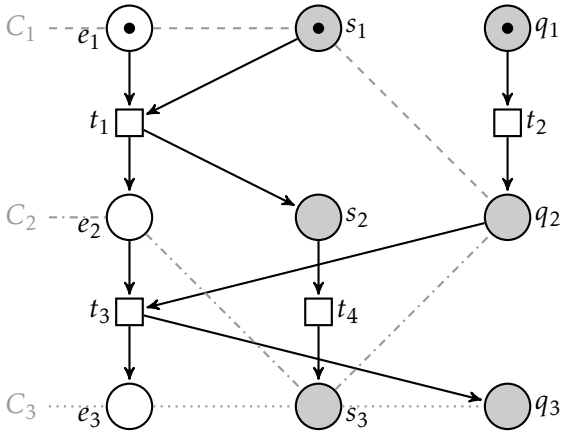


Figure 3.3.: A Petri game is depicted which is equal to its unfolding. The mcuts of the unfolding of this Petri game are $C_1 = \{e_1, s_1, q_2\}$, $C_2 = \{e_2, s_3, q_2\}$, and $C_3 = \{e_3, s_3, q_3\}$.

decision tuple maps to the position of a sequence of backward moves. More details can be found in Section 3.3.4.

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

3.3.2. States of Player 0, States of Player 1, and Accepting States

Causal memory in Petri games is encoded in Büchi games by letting Player 0 fix the decisions of allowed transitions for system players as early as possible and having Player 1 fire transitions with an environment place in their precondition as late as possible at mcuts. Formally, *cuts* are markings in unfoldings, and an *mcut* is a cut where all enabled transitions have an environment place in their precondition, i.e., all system players progressed maximally on their own. With Figure 3.3, we illustrate mcuts. The initial cut $\{e_1, s_1, q_1\}$ is *not* an mcut as the enabled transition t_2 has only the system place q_1 in its precondition. After t_2 fires, the cut $C_1 = \{e_1, s_1, q_2\}$ is an mcut as the only enabled transition t_1 has environment place e_1 in its precondition. Analog arguments lead to $\{e_2, s_2, q_2\}$ not being an mcut and $C_2 = \{e_2, s_3, q_2\}$ being an mcut. The final cut $C_3 = \{e_3, s_3, q_3\}$ is an mcut as there are no enabled transitions.

A decision marking $\mathbb{D}$ in the states in the Büchi game *corresponds to an mcut* when no $\top$ and no positive NES-status are part of $\mathbb{D}$ and every transition with only system places in its precondition is not enabled or not allowed by a participating system player in $\mathbb{D}$, i.e., $\mathbb{D}$ corresponds to an mcut if and only if $(\forall D \in \mathbb{D} : dec(D) \neq \top \wedge t2(D) \neq true) \wedge (\forall t \in \mathcal{T} : pre(t) \not\subseteq \mathscr{M}(\mathbb{D}_{pre(t)}) \vee pre(t) \cap \mathcal{P}_E \neq \emptyset)$. A state in the Büchi game can correspond

to an mcut although the cut in the unfolding of the Petri game is not an mcut as the decisions of the system players in the Büchi game can disallow transitions. *States of Player 1* are F_B, F_N, and states corresponding to an mcut. *States of Player 0* are all other states. *Accepting states* are F_B and states corresponding to an mcut.

Example 3.3.2. The Petri game from Figure 3.1 has $\{forecast\}$, $\{\{e,k:i\} \mid e \in \{s,c,r\} \wedge 2 \leq i \leq 4\}$, and $\{\{s',w:w_{s'},k:i\},\{c',w:w_{c'},k:i\},\{r',w:w_{r'},k:i\} \mid 2 \leq w_{s'} \leq 3 \wedge 1 \leq w_{c'} \leq 2 \wedge 0 \leq w_{r'} \leq 1 \wedge 2 \leq i \leq 4\}$ as mcuts, i.e., the initial cut, cuts where the power plants produced energy while the energy production by renewable sources was not selected, and all final, reachable cuts. In the Büchi game in Figure 3.2, the eight states v_0, v_3, and v_{13} to v_{18} of Player 1 have decision markings that correspond to an mcut. For states v_0, v_{13}, and v_{14}, all enabled transitions have an environment place in their precondition. For states v_{15} to v_{18}, each decision marking corresponds to a final cut. The decision marking of state v_3 corresponds to an mcut as the second system player in p decided to not allow any of its outgoing transitions. △

3.3.3. Edges in the Büchi Game

Edges in the Büchi game mostly connect states $\mathcal{V} = (\mathbb{D}, M_{T2}, BM_1, \ldots, BM_{\max_S})$ and $\mathcal{V}' = (\mathbb{D}', M'_{T2}, BM'_1, \ldots, BM'_{\max_S})$ where $\mathbb{D}$ is a decision marking, M_{T2} is a marking, and $BM_1, \ldots, BM_{\max_S}$ are as many sequences of backward moves as the maximum number $\max_S$ of system players in the Petri game. There are five sets of edges *TOP*, *SYS*, *NES*, *MCUT*, and *STOP*. In the following description of the five sets of edges, not mentioned elements of the connected states stay the same. The formal definitions can be found in Section 3.4.8. How backward moves identify states corresponding to a bad marking or a nondeterministic decision and how some edges collect and remove them is outlined in Section 3.3.4. Edges for the NES-case are explained in Section 3.3.5.

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

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

(3) Edges from *NES* are the NES-case and occur from states where a decision tuple in $\mathbb{D}$ has positive NES-status. To obtain $\mathbb{D}'$, Player 0 fires a transition as in (2) but only from decision tuples with positive NES-status resulting in new decision tuples with positive NES-status. This includes the storage of backward moves. The NES-case is successful if the marking M_{T2} is reached again and all players in it moved. Then, decision tuples with NES-status *true* are set to NES-status *end* and M'_{T2} becomes the empty marking. Otherwise, the marking M_{T2} is not changed. Decision tuples with ended NES-status never move.

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

(5) Edges from *STOP* occur from states with no transition enabled or corresponding to losing behavior. They replace other outgoing edges for losing behavior. States corresponding to termination lead to the winning state F_B. States corresponding to a deadlock but not termination lead to the losing state F_N. If backward moves detect a bad marking or a nondeterministic decision, the state leads to F_N. In the NES-case, a synchronization of decision tuples with positive and negative NES-status or a deadlock or vanishing of decision tuples with positive NES-status leads to F_N. Decision tuples with positive NES-status can vanish when transitions with empty postcondition fire. Without this case, a state without decision tuples with positive NES-status but with a marking to repeat in the NES-case can exist.

Example 3.3.3. In Figure 3.2, outgoing edges of state v_1 are in *TOP*. Outgoing edges of states v_2, v_6, and v_8 to v_{12} are in *SYS*. Outgoing edges of states v_0, v_3, v_{13}, and v_{14} are in *MCUT*. Other edges are in *STOP*. No edges in *NES* exist in the depicted part. Outgoing edges of states v_4 and v_5 represent the deadlock of the second system player in p disallowing both outgoing transitions while only they are enabled. The outgoing edge of state v_7 encodes a nondeterministic decision of the second system player, which allows two enabled transitions. Such a decision is only useful if another player ensures that at most one of the transitions becomes enabled. Outgoing edges of states v_{15} to v_{17} represent termination. The outgoing edge of state v_{18} represents a bad marking for six produced units of energy. △

If, as in our construction, (I) Player 0 immediately resolves $\top$ to the decisions of system players, (II) Player 0 decides which transitions with only system places in their precondition fire following the decisions of system players, and (III) Player 1 decides

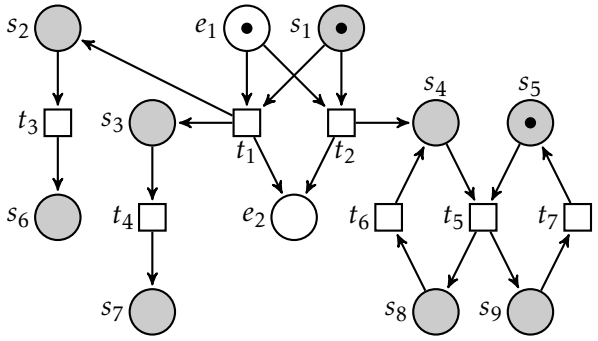


Figure 3.4.: A Petri game is depicted to illustrate backward moves and the NES-case. In this Petri game, all reachable markings containing both s_2 and s_7 are bad markings.

as late as possible at mcuts which transitions with an environment place in their precondition fire following the decisions of system players, then the corresponding Büchi games encode causal memory [FO17]. Allowed transitions with only system places in their precondition fire in an order determined by Player 0 until an mcut is reached. There, Player 1 decides for the environment player which allowed transition to fire. Afterward, this process repeats itself.

3.3.4. Backward Moves in the Büchi Game

In the Büchi game, Player 0 can avoid markings by picking the firing order for transitions with only system places in their precondition. In Figure 3.4, the two system players in s_2 and s_3 are reached after t_1 fires. One can fire t_3, the other t_4. This results in the firing sequences $t_1t_3t_4$ and $t_1t_4t_3$. If s_2 and s_7 are in a bad marking, then Player 0 can decide for edges corresponding to the first firing sequence and the bad marking is missed. Notice that this cannot occur with bad places as local winning condition. We introduce backward moves to avoid such problems. A *backward move* is a pair of decision markings. It stores the change to the decision tuples by edges from *SYS* and *NES*. For every such edge from $\mathcal{V} = (\mathbb{D}, M_{T2}, BM_1, \ldots, BM_{\max_S})$ to $\mathcal{V}' = (\mathbb{D}', M'_{T2}, BM'_1, \ldots, BM'_{\max_S})$, we obtain $\mathbb{D}_{pre}$ and $\mathbb{D}_{post}$ with $\mathbb{D}' = (\mathbb{D} \setminus \mathbb{D}_{pre}) \cup \mathbb{D}_{post}$ and add backward move $(\mathbb{D}_{pre}, \mathbb{D}_{post})$ to the end of BM_{id} of all participating players with identifier id.

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

a decision marking corresponding to a bad marking or a nondeterministic decision is detected, the current state $\mathcal{V}'$ only has an edge to F_N.

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

The sequence of backward moves can grow infinitely long when system players play infinitely without the environment player and without the NES-case. This would result in a Büchi game with infinitely many states. To avoid this, the Büchi game becomes losing for Player 0 when it plays in a way that corresponds to a strategy with a variant of useless repetitions [Gim17] for the system players in the Petri game. Our variant of useless repetitions identifies the repetition of a loop consisting only of transitions without the environment player in their precondition such that the last mcut of the system players does not change, i.e., the system players repeat a loop in which they do not exchange any new information about the environment player. Thus, winning strategies have to avoid playing a useless repetition more than once between the successor of an mcut and the next mcut. This can be achieved either by continuing to the next mcut or by setting some players to positive NES-status and completing the NES-case, i.e., playing infinitely without the environment player.

Example 3.3.4. In Figure 3.2, we include the collection of backward moves. State v_{13} represents each power plant producing one unit of energy after a sunny weather forecast. It is reached from state v_6 either via state v_9 or v_{10} depending on which power plant produces energy first. State v_{13} has a backward move for each power plant: $(\{(1,p,\mathit{false},\{p_l\},1)\},\{(1,k,\mathit{false},\emptyset,1)\})$ and $(\{(2,p,\mathit{false},\{p_l\},1)\},\{(2,k,\mathit{false},\emptyset,1)\})$. Because the three markings $\{s,k:2\}$ (underlying marking of v_{13}), $\{s,p,k\}$ (applying one backward move), and $\{s,p:2\}$ (applying both backward moves) are no bad markings and all decisions are deterministic, state v_{13} continues with edges for the transitions of the environment place s instead of having an edge to F_N. △

3.3.5. Encoding the NES-Case Directly in the Büchi Game

We handle the NES-case where system players play infinitely without firing a transition with an environment place in its precondition directly in the Büchi game as players in the NES-case might be in a bad marking. This is in contrast to the reduction for bad places [FO17].

In the Büchi game, Player 0 has to reach an accepting state infinitely often in order to win the game. Only F_B and states corresponding to an mcut are accepting states. Transitions with only system places in their precondition are fired between successors of mcuts and the following mcut. Thus, if the system players can fire transitions with only system places in their precondition infinitely often, eventually a useless repeti-

tion is reached which is losing. To overcome this, we give Player 0 the possibility to change the NES-status for decision tuples of system players from negative to positive. The underlying marking of this change is stored and afterward only transitions from decision tuples with positive NES-status can be fired. Firing these transitions maintains the positive NES-status for new decision tuples. Instead of firing infinitely many transitions, the NES-case is ended if the stored marking is reached again and all players in the marking have moved. In this case, the NES-status of all decision tuples with positive NES-status is changed to ended NES-status and the Büchi game continues with the remaining decision tuples with negative NES-status. The requirement to move is necessary as otherwise too many players could get ended NES-status. Decision tuples with ended NES-status are maintained as backward moves can be applicable to them, i.e., backward moves store the NES-status and allow us to reverse it in search for a bad marking. We can thus ensure that continuing with the case where all decision tuples have negative NES-status avoids bad markings that span the NES-case.

Player 0 has to disclose decision tuples with positive NES-status if system players fire infinitely many transitions with only system places in their precondition. Otherwise, they lose the game as no accepting state is reached infinitely often. It is losing if system players with positive and negative NES-status synchronize, if players with positive NES-status deadlock, if one such player is not moved and the marking from the start of the NES-case is repeated, if all such players vanish, or if another marking is repeated. Notice that at most one NES-case is necessary per branch in the strategy tree of the Büchi game. For a safety winning condition, possible NES-cases after the first successful one can simply terminate. One disclosure is necessary when the environment player can terminate. Otherwise, the system is responsible when the environment player terminates and players with possible positive NES-status deadlock.

Example 3.3.5. A Petri game with necessary NES-case in the encoding Büchi game is shown in Figure 3.4. After Player 0 allows transition t_2 and Player 1 fires it, a state is reached where the decision tuples for s_4 and s_5 can be set to positive NES-status by Player 0. After transitions t_5, t_6, and t_7 fire, the marking $\{s_4, s_5\}$ is repeated and the NES-case is successful, proving that t_5, t_6, and t_7 can fire infinitely often. No more transitions can be fired and the winning state F_B is the sole successor state because the environment player terminated. Player 0 can also set the decision tuples for $\{s_8, s_9\}$, $\{s_4, s_9\}$, or $\{s_5, s_8\}$ to positive NES-status and show a repetition of the respective marking. To win, Player 0 avoids deadlocks and sets one of the four pairs $\{s_4, s_5\}$, $\{s_8, s_9\}$, $\{s_4, s_9\}$, or $\{s_5, s_8\}$ to positive NES-status. Otherwise, they play infinitely without reaching an accepting state. △

3.3.6. Decidability Result

We analyze the traits of the constructed Büchi game. Full proofs are in Section 3.4.9.

Lemma 1 (From Büchi game strategies to Petri game strategies)**.** *If Player 0 has a winning strategy in the Büchi game, then there exists a winning strategy for the system players in the Petri game.*

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

Lemma 2 (From Petri game strategies to Büchi game strategies)**.** *If the system players have a winning strategy in the Petri game, then there exists a winning strategy for Player 0 in the Büchi game.*

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

Theorem 3 (Game solving)**.** *For Petri games with a bounded number of system players, one environment player, and bad markings, the question of whether the system players have a winning strategy is decidable in* 2-EXPTIME.

Proof Sketch. The complexity is based on the double exponential number of states in the Büchi game and polynomial solving of Büchi games. There are exponentially many states in the size of the Petri game to represent decision tuples and each of these states has to store sequences of backward moves of at most exponential length in the size of the Petri game. □

Remark 3.3.1. *In the presented construction, Player 0 in the Büchi game decides both the decisions of the system players in the Petri game and the order in which concurrent transitions with only system places in their precondition are fired between states after an mcut and states corresponding to the next mcut. We call the fact that Player 0 determines the order of these concurrent transitions the* system scheduling*. The natural question arises what happens if Player 1 determines this order while Player 0 still decides the decisions of the system player and whether we can thereby leave out backward moves from our construction. We call this idea the* environment scheduling*. The environment scheduling without backward moves has been pursued in some preliminary work [Spr15] on bad markings as global winning condition.*

We give a counterexample to the environment scheduling without backward moves being a suitable replacement of the system scheduling and backward moves. This somewhat surprising result is caused by allowing Player 0 to make different decisions for the system players in the Petri game depending on which scheduling of concurrent transitions is chosen by Player 1 without the possibility to roll fired transitions back via backward moves. We consider the extract of the Petri game in Figure 3.5. It consists of two system players and has the three

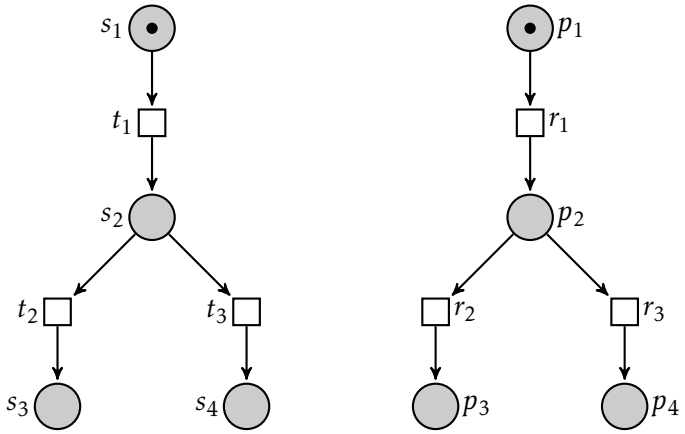


Figure 3.5.: A simple extract of a Petri game is depicted. It consists of two system players that can each make a decision between two transitions after firing one transition. All markings of the whole Petri game containing one of the following three pairs of places $\{s_1, p_3\}$, $\{s_3, p_1\}$, and $\{s_4, p_4\}$ are bad markings.

bad markings $\{s_1, p_3\}$, $\{s_3, p_1\}$, and $\{s_4, p_4\}$ in this extract. The third bad marking $\{s_4, p_4\}$ requires that the two system players not both allow transition t_3 and transition r_3, which would lead to this bad marking. When the two system players not both allow transition t_3 and transition r_3, the two system players cannot always avoid the one of the other two bad markings. When allowing transition r_2, firing transitions r_1 and r_2 before the system player in place s_1 moves leads to the bad marking $\{s_1, p_3\}$. Analogously, when allowing transition t_2, firing transitions t_1 and t_2 before the system player in place p_1 moves leads to the bad marking $\{s_3, p_1\}$. Therefore, the system players cannot win a Petri game where the extract of the Petri game in Figure 3.5 is reachable.

In Figure 3.6, we give a visualization of the, by Player 0 chosen, decisions of the system players in answer to the scheduling decisions by Player 1 for the environment scheduling. The vertices of the tree symbolize the decisions of the system players from Figure 3.5. Initially, the one system player in place s_1 allows transition t_1 and the other system player in place p_1 allows transition r_1. After the environment scheduling decided between firing transition t_1 or transition r_1 as the only two enabled and allowed transitions, the moved system player makes its next decision. As soon as one of the players moves, it becomes acceptable for this player to allow either transition t_3 or transition r_3. When the system player from places starting with s is moved first, then it decides for transition t_3 (cf. upper branch of Figure 3.6). When it is moved later on, then it decides for transition t_2 (cf. lower branch of Figure 3.6). An analog statement holds for the system player from places starting with r and transitions r_3 and t_2.

One can clearly see that a bad marking is never reached in Figure 3.6, which shows that the environment scheduling without backward moves determines the wrong winner for the extract of a Petri game shown in Figure 3.5. Only when using backward moves irrespective of whether the system scheduling or the environment scheduling is used, we can determine that no winning strategy can exist for the extract of a Petri game shown in Figure 3.5.

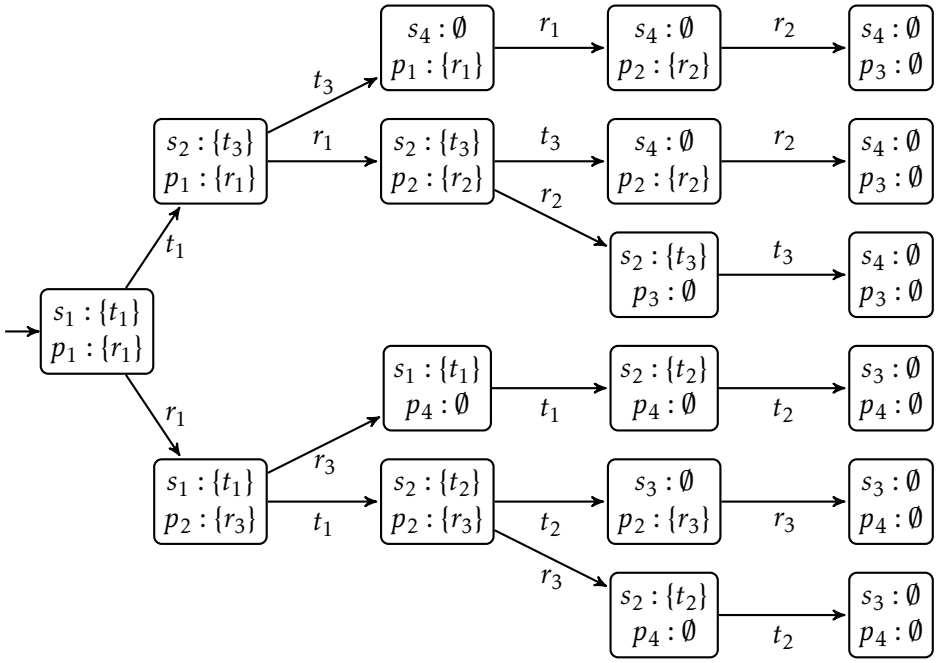


Figure 3.6.: A tree is depicted that has the decisions of the system players from Figure 3.5 as vertices in answer to the fired transitions chosen by the environment scheduling. The fired transitions are symbolized by the edges of the tree.

3.4. Formal Details

We give the formal reduction from Petri games with a bounded number of system players, one environment player, and bad markings to Büchi games. Let $\max_S$ be the *maximal number of system players* visible at the same time in the Petri game.

3.4.1. Decision Tuples

A *decision tuple* for a player consists of an *identifier*, a *position*, a *NES-status*, a *decision*, and a *representation of the last mcut*. It has type $\{1,\dots,\max_S\} \times \mathcal{P}_S \times \{\mathit{false}, \mathit{true}, \mathit{end}\} \times (\mathbb{P}(\mathcal{T}) \cup \{\top\}) \times \{1,\dots,\max_S\}$ for system places and type $\{0\} \times \mathcal{P}_E \times \{\mathit{false}\} \times \mathbb{P}(\mathcal{T}) \times \{0\}$ for environment places.

Definition 13 (Decision tuples)
The set $\mathcal{D}_S$ of *system decision tuples* is defined as $\mathcal{D}_S = \{(id, p, b, T, K) \mid id \in \{1,\dots,\max_S\} \wedge p \in \mathcal{P}_S \wedge b \in \{\mathit{false}, \mathit{true}, \mathit{end}\} \wedge (T \subseteq \mathit{post}(p) \vee T = \top) \wedge K \in \{1,\dots,\max_S\}\}$. The set $\mathcal{D}_E$ of *environment decision tuples* is defined as $\mathcal{D}_E = \{(0, p, \mathit{false}, \mathit{post}(p), 0) \mid p \in \mathcal{P}_E\}$. The set $\mathcal{D}_S$ of *decision tuples* is defined as $\mathcal{D} = \mathcal{D}_S \cup \mathcal{D}_E$.

We define the following functions to retrieve the respective elements of a decision tuple D: For $D = (id, p, b, T, K)$, *id* obtains the first element representing the identifier,

i.e., $id(D) = id$, pl obtains the second element representing the place, i.e., $pl(D) = p$, $t2$ the third element representing the NES-status, i.e., $t2(D) = b$, dec the fourth element representing the decision, i.e., $dec(D) = T$, and lmc the fifth element representing the last mcut, i.e., $lmc(D) = K$.

3.4.2. Enabledness of Transitions from Decision Markings

We use decision markings as subset of the set of decision tuples as representation of the marking of the Petri game. Thus, we are only interested in decision markings that correspond to a reachable marking in the Petri game and where each identifier of players occurs at most once. We define the underlying *marking* $\mathscr{M}(\mathbb{D})$ of a decision marking $\mathbb{D}$ as $\mathscr{M}(\mathbb{D})(p) = |\{D \in \mathbb{D} \mid pl(D) = p\}|$ for all places $p \in \mathcal{P}$. We stipulate that each decision marking $\mathbb{D}$ corresponds to a (k-bounded) Petri game $\mathcal{G}$, i.e., $\mathscr{M}(\mathbb{D})$ is a reachable marking in the Petri game ($\exists M \in \mathcal{R}(\mathcal{N}) : \mathscr{M}(\mathbb{D}) = M$). Therefore, each place occurs at most k times in $\mathscr{M}(\mathbb{D})$ ($\forall p \in \mathcal{P} : \mathscr{M}(\mathbb{D})(p) \leq k$). Formally, $\mathbb{D}$ is of type $((\{1,\dots,\max_S\} \times \mathcal{P}_S \times \{false, true, end\} \times (\mathbb{P}(\mathcal{T}) \cup \{\top\}) \times \{1,\dots,\max_S\}) \to \{0,\dots,k\}) \cup ((\{0\} \times \mathcal{P}_E \times \{false\} \times \mathbb{P}(\mathcal{T}) \times \{0\}) \to \{0,1\})$.

Definition 14 (Decision markings)
The set $\mathscr{D}$ of *decision markings corresponding to reachable markings in $\mathcal{G}$ and with unique identifiers of players* is defined as

$$\mathscr{D} = \{\mathbb{D} \subseteq \mathcal{D} \mid \exists M \in \mathcal{R}(\mathcal{N}) : \mathscr{M}(\mathbb{D}) = M \wedge \forall i \in \{1,\dots,\max_S\} : |\{D \in \mathbb{D} \mid id(D) = i\}| \leq 1\}.$$

We define the *enabledness* of a transition t from a decision marking $\mathbb{D}$. We first remove decision tuples from $\mathbb{D}$ that are not in the precondition of t, have $\top$ as their decision, disallow t, or have ended NES-status. Formally, we introduce the decision marking $\mathbb{D}_{pre(t)}$ to retain only decision tuples that represent a place in the precondition of t and that allow t as

$$\mathbb{D}_{pre(t)} = \{(id, p, b, T, K) \in \mathcal{D} \mid (id, p, b, T, K) \in \mathbb{D} \wedge t \in T \wedge p \in pre(t) \wedge b \neq end\}.$$

Now, we can check the enabledness of a transition t from a decision marking $\mathbb{D}$ by $pre(t) \subseteq \mathscr{M}(\mathbb{D}_{pre(t)})$. We also define the decision marking $\mathbb{D}_{t2}$ to retain only decision tuples with positive NES-status as

$$\mathbb{D}_{t2}(id, p, b, T, K) = \{(id, p, b, T, K) \in \mathcal{D} \mid (id, p, b, T, K) \in \mathbb{D} \wedge b = true\}.$$

For the enabledness of a transition in the NES-case, we introduce the decision marking $\mathbb{D}_{pre(t) \wedge t2}$ to retain only decision tuples with positive NES-status that represent a place in the precondition of t and that allow t as $\mathbb{D}_{pre(t) \wedge t2} = \mathbb{D}_{pre(t)} \cap \mathbb{D}_{t2}$.

3.4.3. Decision Markings corresponding to Mcuts

We call a decision marking $\mathbb{D}$ *corresponding to an mcut* when no $\top$ exists in $\mathbb{D}$, every transition with only system places in its precondition is either not enabled or not allowed by a participating system player in $\mathbb{D}$, and no positive NES-status exists in $\mathbb{D}$.

Definition 15 (Decision markings corresponding to an mcut)
A decision marking $\mathbb{D}$ *corresponds to an mcut* if and only if
$(\forall D \in \mathbb{D} : dec(D) \neq \top \wedge t2(D) \neq true) \wedge \forall t \in \mathcal{T} : pre(t) \nsubseteq \mathscr{M}(\mathbb{D}_{pre(t)}) \vee pre(t) \cap \mathcal{P}_E \neq \emptyset$.

This definition can be expressed in a simpler way than in the original paper on Petri games [FO17] because the NES-case is handled directly in the Büchi game.

3.4.4. Backward Moves

We define *backward moves* $\mathcal{B}$ as pairs of decision markings. Backward moves can be based on transitions firing including at the start and at the end of the NES-case.

Definition 16 (Backward moves for transitions firing)
The set $\mathcal{B}_T$ of *backward moves for transitions firing* is defined as

$$\begin{aligned}\mathcal{B}_T = \{(\mathbb{D},\mathbb{D}') \in \mathbb{P}(\mathcal{D}_S) \times \mathbb{P}(\mathcal{D}_S) \mid (\exists t \in \mathcal{T} : \mathscr{M}(\mathbb{D}) = pre(t) \wedge \mathscr{M}(\mathbb{D}') = post(t)) \wedge \\ (\forall D \in \mathbb{D} : t \in dec(D)) \wedge (\forall D \in \mathbb{D} \cup \mathbb{D}' : dec(D) \neq \top) \wedge \\ (((\forall D \in \mathbb{D} : t2(D) = false) \wedge (\forall D \in \mathbb{D}' : t2(D) = false \vee t2(D) = true)) \vee \\ ((\forall D \in \mathbb{D} : t2(D) = true) \wedge ((\forall D \in \mathbb{D}' : t2(D) = true) \vee (\forall D \in \mathbb{D}' : t2(D) = end))))\}.\end{aligned}$$

Each backward move has to be based on a transition t. Each decision tuple of a backward move does not have $\top$ as its decision. The decision tuples before the transition have to allow the transition. All decision tuples on the left side of a backward move can have NES-status *false* and all decision tuples on the right side have NES-status *false* or *true*. Alternatively, all decision tuples on the left side of a backward move can have NES-status *true* and all decision tuples on the right side have NES-status *true* or *end*.

Definition 17 (Backward moves for the start and the end of the NES-case)
The set $\mathcal{B}_{T2}$ of *backward moves for the start and the end of the NES-case* is defined as

$$\begin{aligned}\mathcal{B}_{T2} = \{(\mathbb{D},\mathbb{D}') \in \mathbb{P}(\mathcal{D}_S) \times \mathbb{P}(\mathcal{D}_S) \mid \\ ((\forall D \in \mathbb{D} : t2(D) = false) \wedge (\forall D \in \mathbb{D}' : t2(D) = true) \wedge \\ (\forall id \in \{1,\ldots,\max_S\}, p \in \mathcal{P}_S, d \subseteq post(t), K \in \{1,\ldots,\max_S\} : \\ (id,p,d,false,K) \in \mathbb{D} \Leftrightarrow (id,p,d,true,K) \in \mathbb{D}')) \vee \\ ((\forall D \in \mathbb{D} : t2(D) = true) \wedge (\forall D \in \mathbb{D}' : t2(D) = end) \wedge \\ (\forall id \in \{1,\ldots,\max_S\}, p \in \mathcal{P}_S, d \subseteq post(t), K \in \{1,\ldots,\max_S\} : \\ \mathbb{D}((id,p,d,true,K)) = \mathbb{D}'((id,p,d,end,K))))\}.\end{aligned}$$

Backward moves for the start and the end of the NES-case reverse the change of all decision tuples in a decision marking from negative to positive or from positive to ended NES-status without changing the identifier, the positions, decisions of the decision tuples, or the last mcut. This case becomes necessary because not all players of the NES-case are in the precondition of the transition starting the NES-case or the postcondition of the transitions ending the NES-case.

Definition 18 (Backward moves)
The set $\mathcal{B}$ of *backward moves* is defined as $\mathcal{B} = \mathcal{B}_T \cup \mathcal{B}_{T2}$.

The set of backward moves is defined as the union of the set of backward moves for transitions firing and the set of backward moves for the start and the end of the NES-case.

3.4.5. States in the Büchi Game

Definition 19 (States in the Büchi game)
The *states V in the Büchi game* are defined as

$$V = V_{BN} \cup \mathscr{D} \times (\mathcal{P}_S \to \{0,\dots,k\}) \times (\mathcal{B}^*)^{\max_S} \times \{1,\dots,\max_S\}$$

with $V_{BN} = \{F_B, F_N\}$ (B and N stand for *Büchi* and *non-Büchi*).

The sequences of backward moves $\mathcal{B}^*$ are of finite size because we do not allow useless repetition. We will later see that they are limited to be of at most single exponential length in terms of the size of the Petri game.

The two unique states F_B and F_N in V_{BN} are used to represent finite *winning and losing behavior* in the Petri game with a successor edge in the Büchi game. All other states have the following form: The first element is the decision marking storing the current players. The second element is the marking which Player 0 claims to repeat in the NES-case. It is the empty marking when the Büchi game is not in the NES-case. The next $\max_S$ elements represent the sequence of backward moves of each system player which are used to check for bad markings and for nondeterministic decisions.

We define the following functions to access the elements of a state $\mathcal{V} \in V \setminus V_{BN}$ in a Büchi game with $\mathcal{V} = (\mathbb{D}, M_{T2}, BM_1, \dots, BM_{\max_S})$: The function DS obtains the first element representing the decision tuple, i.e., $DS(\mathcal{V}) = \mathbb{D}$. The function $T2M$ obtains the second element representing the marking that Player 0 claims will repeat itself in the NES-case, i.e., $T2M(\mathcal{V}) = M_{T2}$. The function BRG_i obtains the next $\max_S$ elements representing the sequences of backward moves, i.e., $BRG_i(\mathcal{V}) = BM_i$ for $i \in \{1,\dots,\max_S\}$.

Definition 20 (States of Player 0 and states of Player 1)
The *states V_1 of Player 1* are defined as all states in V where the decision marking corresponds to an mcut:

$$V_1 = \{\mathcal{V} \mid DS(\mathcal{V}) \text{ corresponds to an mcut}\}$$

The *states V_0 of Player 0* are defined as all other states:

$$V_0 = V \setminus V_1$$

Definition 21 (Initial state in the Büchi game)
The *initial state I* in the Büchi game is defined as

$$I = (\mathbb{D}_{In}^S \cup \{(0, p_E, false, post(p_E), 0) \mid p_E \in In \cap \mathcal{P}_E\}, \emptyset, []^{\max_S})$$

with $\mathbb{D}_{In}^S \subseteq \mathbb{P}(\mathcal{D}_S)$ such that

$$(\mathscr{M}(\mathbb{D}_{In}^S) = In \setminus \mathcal{P}_E) \wedge (\forall D \in \mathbb{D}_{In}^S : t2(D) = false \wedge dec = \top \wedge lmc(D) = 1) \wedge$$

$$(\forall i \in \{1, \ldots, \max_S\} : |\mathbb{D}_{In}^S \cap \{D \in \mathcal{D}_S \mid id(D) = i\}| \leq 1)$$

and the system players in $\mathbb{D}_{In}^S$ are ordered arbitrarily but fixed (e.g., lexicographically) according to their places and identifiers with the lowest possible sum are used.

The *initial state in the Büchi game* contains decision tuples for all places of the initial marking, an empty marking to repeat, and $\max_S$ many empty sequences of backward moves. The constraints on the the selection of unique identifiers of the players are only used to obtain a single initial state in the Büchi game.

Definition 22 (Accepting states in the Büchi game)
The *accepting states F in the Büchi game* are defined as F_B and all states corresponding to an mcut:

$$F = \{F_B\} \cup \{\mathcal{V} \in V \mid DS(\mathcal{V}) \text{ corresponds to an mcut}\}$$

3.4.6. Finite Winning and Losing Behavior in the Büchi Game

We define finite winning and losing behavior in the Büchi game. Finite losing behavior includes bad markings and nondeterministic decisions. These two cases require to check the via backward moves reachable decision markings. Therefore, we define a Petri net that has the via backward moves reachable decision markings as reachable markings.

Definition 23 (Via backward moves reachable decision markings)
Given a set of sequences of backward moves $BM_1, \ldots, BM_n$ and a decision marking $\mathbb{D}_{In}$, the k-bounded Petri net $\mathcal{N}_{BM_1,\ldots,BM_n}[\mathbb{D}_{In}] = (\mathcal{P}, \mathcal{T}, \mathcal{F}, In)$ to calculate the *via backward moves reachable decision markings* starting from $\mathbb{D}_{In}$ is defined as follows.

- We initialize $\mathcal{N}_{BM_1,\ldots,BM_n}[\mathbb{D}_{In}]$ with $\mathcal{P} = \mathbb{D}_{In}$, $\mathcal{T} = \emptyset$, $\mathcal{F} = \emptyset$, and $In = \mathbb{D}_{In}$ and define the current marking $M = \mathbb{D}_{In}$.

- We apply the following step recursively: We check for the backward moves $(\mathbb{D}, \mathbb{D}')$ from the end of the sequences of backward moves $BM_1, \ldots, BM_n$ whether $\mathbb{D}'$ is contained in the current marking M and whether all participating players in $\mathbb{D}'$ with identifier *id* have backward move $(\mathbb{D}, \mathbb{D}')$ at the end of BM_{id}. If this is the case, then we add each decision tuple of $\mathbb{D}$ to $\mathcal{P}$, $t_{(\mathbb{D}', \mathbb{D})}$ to $\mathcal{T}$, and $\{(D', t_{(\mathbb{D}', \mathbb{D})}) \mid D' \in \mathbb{D}'\}$ and $\{(t_{(\mathbb{D}', \mathbb{D})}, D) \mid D \in \mathbb{D}\}$ to $\mathcal{F}$, we remove $(\mathbb{D}, \mathbb{D}')$ from the end of BM_{id} for all participating players in $\mathbb{D}'$ with identifier *id*, and we update the current marking M by setting it to $(M - \mathbb{D}') + \mathbb{D}$. We continue with the next backward move.

Notice that we reverse the backward moves to apply standard forward algorithms to calculate the set of reachable markings. Notice further that when more than one backward move is applicable from the current marking, then the second backward move is also applicable from the current marking obtained after adding the first backward move to the Petri net, because the backward moves are disjoint. The algorithm terminates as the sequences of backward moves are finite and each iteration of the algorithm makes the sequences smaller.

Definition 24 (Finite winning and losing behavior (non-NES-case))
Terminated states TERM, deadlocked states DL, states NDET corresponding to nondeterministic decisions, and *states BAD corresponding to a bad marking* are defined as follows:

$$
\begin{aligned}
V' =& \{\mathcal{V} \in V \mid \forall D \in DS(\mathcal{V}) : dec(D) \neq \top\} \\
TERM =& \{\mathcal{V} \in V' \mid \forall t \in \mathcal{T} : pre(t) \nsubseteq \mathscr{M}(DS(\mathcal{V}))\} \\
DL =& \{\mathcal{V} \in V' \mid \forall t \in \mathcal{T} : pre(t) \nsubseteq \mathscr{M}(DS(\mathcal{V})_{pre(t)})\} \\
NDET =& \{\mathcal{V} \in V' \mid \exists \mathbb{D} \in \mathcal{R}(\mathcal{N}_{BRG_1(\mathcal{V}),\ldots,BRG_{\max_S}(\mathcal{V})}[DS(\mathcal{V})]), D \in \mathbb{D} : pl(D) \in \mathcal{P}_S \wedge \\
& ((\exists t_1, t_2 \in post(pl(D)) : t_1 \neq t_2 \wedge pre(t_1) \subseteq \mathscr{M}(\mathbb{D}_{pre(t_1)}) \wedge \\
& pre(t_2) \subseteq \mathscr{M}(\mathbb{D}_{pre(t_2)})) \vee (\exists t \in post(pl(D)) : pre(t) \subset \mathscr{M}(\mathbb{D}_{pre(t)})))\} \\
BAD =& \{\mathcal{V} \in V' \mid \exists \mathbb{D} \in \mathcal{R}(\mathcal{N}_{BRG_1(\mathcal{V}),\ldots,BRG_{\max_S}(\mathcal{V})}[DS(\mathcal{V})]) : \mathscr{M}(\mathbb{D}) \in \mathcal{M}_B\}
\end{aligned}
$$

Only states where all $\top$ have been resolved are included. A state is *terminated* when no transition is enabled. A state is *deadlocked* when all transitions are not enabled or not allowed by enough decision tuples. A terminated state is also deadlocked. Terminated states correspond to winning behavior whereas deadlocked states that are not terminated correspond to losing behavior.

A state corresponds to a *nondeterministic decision* if there is a via backward moves reachable decision marking from the state such that either two different transitions are enabled having the same system place in their precondition or if a single transition with a system place in its precondition is enabled with more than enough players allowing the transition. In the second case, two different instances of the transition are enabled.

A state corresponds to a *bad marking* if there is a via backward moves reachable decision marking from the state such that the underlying marking is in the set of bad markings.

Definition 25 (Finite losing behavior (NES-case))
Deadlocked states DL_{t2} in the NES-case, states $SYNC_{t2}$ corresponding to synchronization between decision tuples with positive and negative NES-status, and *states corresponding to vanished decision tuples with positive NES-status* are defined as follows:

$$\begin{aligned}
V' =& \{\mathcal{V} \in V \mid \forall D \in DS(\mathcal{V}) : dec(D) \neq \top\} \\
DL_{t2} =& \{\mathcal{V} \in V' \mid (\exists D \in DS(\mathcal{V}) : t2(D) = true) \wedge \forall t \in \mathcal{T} : \\
& pre(t) \nsubseteq \mathscr{M}(DS(\mathcal{V})_{pre(t) \wedge t2})\} \\
SYNC_{t2} =& \{\mathcal{V} \in V' \mid \exists \mathbb{D} \in \mathcal{R}(\mathcal{N}_{BRG_1(\mathcal{V}),\ldots,BRG_{\max_S}(\mathcal{V})}[DS(\mathcal{V})]), D \in \mathbb{D} : \\
& t2(D) = true \wedge \exists t \in post(pl(D)) : \\
& pre(t) \nsubseteq \mathscr{M}(\mathbb{D}_{pre(t) \wedge t2}) \wedge pre(t) \subseteq \mathscr{M}(\mathbb{D}_{pre(t)})\} \\
VAN_{t2} =& \{\mathcal{V} \in V' \mid T2M(\mathcal{V}) \neq \emptyset \wedge \forall D \in DS(\mathcal{V}) : t2(D) \neq true\}
\end{aligned}$$

Only states where all $\top$ have been resolved are included. A state is *deadlocked in the NES-case* when at least one decision tuple has positive NES-status and all transitions are not enabled or not allowed from the decision marking with positive NES-status. Notice that this includes all *in the NES-case terminated* states. We do not need to differentiate between terminated and deadlocked in the NES-case because both cases are losing behavior.

A state corresponds to *synchronization between decision tuples with negative and positive NES-status* if there exists a via backward moves reachable decision marking from the state such that at least one decision tuple has positive NES-status and it exists a transition from the postcondition of that place that is not enabled or not allowed by only decision tuples with positive NES-status but is enabled and allowed when not regarding the distinction between decision tuples with negative and positive NES-status.

A state corresponds to *vanished decision tuples with positive NES-status* when a nonempty marking to repeat exists but no more decision tuples with positive NES-status exist. This can occur when at least one transition with an empty postcondition exists that removes all decision tuples with positive NES-status.

The definitions of states corresponding to nondeterministic decisions or to bad markings also apply to decision markings with positive NES-status.

3.4.7. Useless Repetitions in the Büchi Game

We define useless repetitions in the Büchi game. Therefore, we search for the repetition of loops in the graph of via backward moves reachable decision markings which

did not increase the information of any participating player. Such a useless repetition represents losing behavior to prevent the collection of sequences of backward moves of infinite length.

Definition 26 (Useless repetition in the Büchi game)
For $j \in \{1,2,3,4\}$, we define $BM_1^j, \ldots, BM_{\max_S}^j$ to be the corresponding backward moves to a decision marking $\mathbb{D}^j \in \mathcal{R}(\mathcal{N}_{BRG_1(\mathcal{V}),\ldots,BRG_{\max_S}(\mathcal{V})}[DS(\mathcal{V})])$ from the construction of $\mathcal{N}_{BRG_1(\mathcal{V}),\ldots,BRG_{\max_S}(\mathcal{V})}[DS(\mathcal{V})]$. States UR corresponding to a useless repetition in the Büchi game are defined as follows:

$$UR = \{\mathcal{V} \in V' \mid \exists \mathbb{D}^1, \mathbb{D}^2, \mathbb{D}^3, \mathbb{D}^4 \in \mathcal{R}(\mathcal{N}_{BRG_1(\mathcal{V}),\ldots,BRG_{\max_S}(\mathcal{V})}[DS(\mathcal{V})]) :$$
$$\mathbb{D}^1 = \mathbb{D}^2 = \mathbb{D}^3 = \mathbb{D}^4 \wedge \forall i \in \{1,\ldots,\max_S\} : BM_i^1 \subset BM_i^2 \subseteq BM_i^3 \subset BM_i^4 \wedge$$
$$BM_i^2 - BM_i^1 = BM_i^4 - BM_i^3\}$$

A useless repetition in the Büchi game occurs when the same loop happened twice without increasing the knowledge of the participating players. The first loops occurs from $\mathbb{D}_1$ to $\mathbb{D}_2$ and the second one from $\mathbb{D}_3$ to $\mathbb{D}_4$ where $\mathbb{D}_2$ and $\mathbb{D}_3$ can be the same position in $\mathcal{R}(\mathcal{N}_{BRG_1(\mathcal{V}),\ldots,BRG_{\max_S}(\mathcal{V})}[DS(\mathcal{V})])$. In the definition, the knowledge of the players does not increase because the decision markings have to be the same which includes the last mcut.

3.4.8. Edges in the Büchi Game

Winning behavior without a successor state (*TERM*) leads to the unique accepting state F_B with a self-loop. Losing behavior leads to the unique non-accepting state F_N with a self-loop.

Definition 27 (Edges *STOP* for finite winning and losing behavior)
Edges $STOP_B$ for *finite winning behavior* without a successor state are defined as

$$STOP_B = \{(\mathcal{V}, F_B) \in V \times \{F_B\} \mid \mathcal{V} \in TERM \setminus (BAD \cup UR \cup DL_{t2} \cup VAN_{t2})\}.$$

Edges $STOP_N$ for *finite losing behavior* are defined as

$$STOP_N = \{(\mathcal{V}, F_N) \in V \times \{F_N\} \mid \mathcal{V} \in (DL \setminus TERM) \cup NDET \cup BAD \cup UR \cup DL_{t2} \cup SYNC_{t2} \cup VAN_{t2}\}.$$

Edges *STOP* for *finite winning and losing behavior* are defined as $STOP = STOP_B \cup STOP_N$.

Again, *B* and *N* stand for *Büchi* and *non-Büchi*.
We introduce notation to calculate the successors of a decision marking:

Definition 28 (NES-status change and $\top$ removal)
Given a decision marking $\mathbb{D}$ without decision tuples with positive NES-status, the function $suc_{dec}(\mathbb{D})$ identifies the set of all possible decision markings corresponding to $\mathbb{D}$ where the NES-status can change and $\top$ is replaced by a set of allowed transitions for system places. It is defined as

$$
\begin{aligned}
suc_{dec}(\mathbb{D}) = \{\mathbb{D}' \in \mathscr{D} \mid \mathbb{D}' = \{(id, p, b', T', K) \in \mathcal{D} \mid (id, p, b, T, K) \in \mathbb{D} \wedge \\
(T \neq \top \Rightarrow T' = T) \wedge (T = \top \Rightarrow T' \subseteq post(p)) \wedge (p \in \mathcal{P}_E \Rightarrow b' = b) \wedge \\
((\exists id'', K'' \in \{1, \ldots, \max_S\}, p'' \in \mathcal{P}_S, T'' \subseteq post(p) : \\
(id'', p'', end, T'', K'') \in \mathbb{D}) \Rightarrow b' = b) \wedge \\
((\forall id'', K'' \in \{1, \ldots, \max_S\}, p'' \in \mathcal{P}_S, T'' \subseteq post(p) : \\
(id'', p'', end, T'', K'') \notin \mathbb{D}) \Rightarrow b' \in \{false, true\})\} \wedge \\
\forall id \in \{1, \ldots, \max_S\} : |\mathbb{D} \cap \{D \in \mathcal{D}_S \mid id(D) = id\}| = |\mathbb{D}' \cap \{D \in \mathcal{D}_S \mid id(D) = id\}|\}.
\end{aligned}
$$

We calculate all $\mathbb{D}'$ such that, for each element (id, p, b, T, K) in $\mathbb{D}$, there is exactly one element (id, p, b', T', K) in $\mathbb{D}$. This is ensured by the last requirement because each identifier occurs at most once in $\mathbb{D}$. For T', $\top$ is replaced by a set of allowed transitions and remains the same otherwise. For p being an environment place, the NES-status always remains *false*. For p being a system place, the NES-status can change from *false* to *true* unless there is a decision tuple in $\mathbb{D}$ with NES-status *end*. In this case, the NES-status remains the same.

All edges in the Büchi game go from a state of the form $\mathcal{V} = (\mathbb{D}, M_{T2}, BM_1, \ldots, BM_{\max_S})$ to a state $\mathcal{V}' = (\mathbb{D}', M'_{T2}, BM'_1, \ldots, BM'_{\max_S})$ in the following. We prevent states in $(DL \setminus TERM) \cup NDET \cup BAD \cup UR \cup DL_{t2} \cup SYNC_{t2} \cup VAN_{t2}$ to have further outgoing edges because they correspond to losing behavior.

Definition 29 (Edges *TOP*)
Edges *TOP* to let Player 0 make decisions and change NES-status are defined as

$$
\begin{aligned}
TOP = \{(\mathcal{V}, \mathcal{V}') \in V_0 \times V \mid \mathcal{V} = (\mathbb{D}, M_{T2}, BM_1, \ldots, BM_{\max_S}) \wedge \exists \mathbb{D}' \in suc_{dec}(\mathbb{D}) : \\
\mathcal{V}' = (\mathbb{D}', M'_{T2}, BM'_1, \ldots, BM'_{\max_S}) \wedge (\forall D \in \mathbb{D} : t2(D) \neq true) \wedge M_{T2} = \emptyset \wedge \\
(\exists D \in \mathbb{D} : dec(D) = \top) \wedge M'_{T2} = \mathscr{M}(\mathbb{D}'_{t2}) \wedge (\forall id \in \{1, \ldots, \max_S\} : BM'_{id} = BM_{id})\}.
\end{aligned}
$$

In $\mathcal{V}$, no decision tuple with positive NES-status exists in $\mathbb{D}$, the marking M_{T2} to repeat in the NES-case is empty, and a decision tuple in $\mathbb{D}$ has $\top$ as decision. To obtain $\mathbb{D}'$ from $\mathbb{D}$, $\top$ is removed and system decision tuples can be designated as having positive NES-status by choosing $\mathbb{D}'$ from $suc_{dec}(\mathbb{D})$. The underlying marking of $\mathbb{D}'$ restricted to decision tuples with positive NES-status is stored in M'_{T2}. It is the empty set when no

decision tuple in $\mathbb{D}'$ has positive NES-status. All sequences of backward moves stay the same.

A transit relation $\Upsilon(t) \subseteq (((pre(t) \cup \{\triangleright\}) \times (post(t) \cup \{\triangleleft\})) \to \mathbb{N})$ [FGHO19] (lifted to bounded Petri nets) relates places in the precondition with places in the postcondition of a transition. It also indicates that a new token is created with $\triangleright$ and that a token is removed with $\triangleleft$. In the following, we assume an arbitrary but fixed transit relation that preserves the type of players in the Petri game and represents the movement of players including their creation and removal. This is needed to maintain the correct identifier for players. In particular, the transit relation returns, for a decision marking $\mathbb{D}$, a player with identifier id at place p, and a transition t to fire, the unique next place p' for the player, written as $p\Upsilon(\mathbb{D}, id, t)p'$. We assume the existence of a function $nextID$ that, given a decision marking $\mathbb{D}$ and a transition t fired from $\mathbb{D}$, returns a unique identifier for each new player p' with $\triangleright\Upsilon(t)p'$.

Definition 30 (Making decisions for successor decision markings)
For a decision marking $\mathbb{D}$ and a from there enabled and allowed transition t with only system places in its precondition, the function $suc_S(\mathbb{D}, t)$ identifies the set of all possible decision markings corresponding to the postcondition of t where no $\top$ exists. It is defined as

$$
\begin{aligned}
&suc_S(\mathbb{D}, t) = \{\mathbb{D}' \in \mathscr{D} \mid \mathbb{D}' = \{(id, p', false, T', K') \in \mathcal{D}_S \mid (id, p, false, T, K) \in \mathbb{D} \wedge \\
&p\Upsilon(\mathbb{D}, id, t)p' \wedge T' \subseteq post(p') \wedge K' = \max(\{lmc(D) \mid D \in \mathbb{D}_{pre(t)}\})\} \cup \\
&\{(id', p', false, T', K') \in \mathcal{D}_S \mid \triangleright\Upsilon(t)p' \wedge T' \subseteq post(p') \wedge \\
&K' = \max(\{lmc(D) \mid D \in \mathbb{D}_{pre(t)}\}) \wedge id' = nextID(\mathbb{D}, t, \triangleright\Upsilon(t)p')\} \wedge \mathscr{M}(\mathbb{D}') = post(t) \wedge \\
&\forall id \in \{1, \ldots, \max_S\} : |\mathbb{D}' \cap \{D \in \mathcal{D}_S \mid id(D) = id\}| \leq 1\}.
\end{aligned}
$$

The NES-status is fixed to *false* as suc_{dec} is used afterward to give the possibility of changing the NES-status from negative to positive.

Definition 31 (Edges SYS)
Edges SYS to let Player 0 fire a transition with no environment places in its precondition and afterward make decisions and change NES-status are defined as

$$
\begin{aligned}
&SYS = \{(\mathcal{V}, \mathcal{V}') \in (V_0 \setminus NDET \cup BAD \cup UR) \times V \mid \mathcal{V} = (\mathbb{D}, M_{T2}, BM_1, \ldots, BM_{\max_S}) \wedge \\
&\exists t \in \mathcal{T}, pc \in suc_S(\mathbb{D}, t), \mathbb{D}' \in suc_{dec}((\mathbb{D} \setminus \mathbb{D}_{pre(t)}) \cup pc) : \\
&\mathcal{V}' = (\mathbb{D}', M'_{T2}, BM'_1, \ldots, BM'_{\max_S}) \wedge (\forall D \in \mathbb{D} : t2(D) \neq true) \wedge M_{T2} = \emptyset \wedge \\
&\mathbb{D} \text{ does } \textit{not} \text{ correspond to an mcut} \wedge pre(t) \cap \mathcal{P}_E = \emptyset \wedge pre(t) \subseteq \mathscr{M}(\mathbb{D}_{pre(t)}) \wedge \\
&\mathbb{D}'' = \mathbb{D}' \setminus \{(id, p, false, T, K), (id, p, true, T, K) \mid (id, p, b, T, K) \in pc\} \wedge \\
&M'_{T2} = \mathscr{M}(\mathbb{D}''_{t2}) \wedge \forall id \in \{1, \ldots, \max_S\} :
\end{aligned}
$$

$$((\exists p \in \mathcal{P}_S, b \in \{false, true, end\}, T \subseteq post(p), K \in \{1, \ldots, \max_S\} :$$
$$(id, p, b, T, K) \in pc \cup (\mathbb{D}'' \setminus \mathbb{D})) \Rightarrow ((\mathbb{D} \setminus \mathbb{D}_{pre(t)} \neq \mathbb{D}'' \Rightarrow BM'_{id} = BM_{id} \cup$$
$$[(\mathbb{D}_{pre(t)}, pc), (\mathbb{D} \setminus (\mathbb{D}_{pre(t)} \cup (\mathbb{D} \cap \mathbb{D}'')), \mathbb{D}'' \setminus \mathbb{D})]) \wedge$$
$$(\mathbb{D} \setminus \mathbb{D}_{pre(t)} = \mathbb{D}'' \Rightarrow BM'_{id} = BM_{id} \cup [(\mathbb{D}_{pre(t)}, pc)]))) \wedge$$
$$((\forall p \in \mathcal{P}_S, b \in \{false, true, end\}, T \subseteq post(p), K \in \{1, \ldots, \max_S\} :$$
$$(id, p, b, T, K) \notin pc \cup (\mathbb{D}'' \setminus \mathbb{D})) \Rightarrow BM'_{id} = BM_{id})\}.$$

In $\mathcal{V}$, no decision tuple with positive NES-status exists in $\mathbb{D}$, the marking M_{T2} to repeat in the NES-case is empty, and $\mathbb{D}$ does *not* correspond to an mcut. To obtain $\mathbb{D}'$ from $\mathbb{D}$, a transition t with no environment places in its preconditions, that is enabled and allowed from $\mathbb{D}$, is chosen. To simulate transition t firing, the decision tuples pc for the postcondition of t are obtained by first choosing decisions via suc_S. Afterward, decision tuples in $\mathbb{D} \setminus \mathbb{D}_{pre(t)}$ and pc can change their NES-status from negative to positive via suc_{dec} to obtain $\mathbb{D}'$. A corresponding backward move $(\mathbb{D}_{pre(t)}, pc)$ is added to BM_{id} of all participating players with identifier id to obtain BM'_{id}. If decision tuples in $\mathbb{D}$ have changed their NES-status, a further backward move for this change is added. Here, we use $\mathbb{D}''$ to remove pc because the NES-status of decision tuples in pc might have changed. All other backward moves are retained. As in *TOP*, the underlying marking of $\mathbb{D}'$ restricted to decision tuples with positive NES-status is stored in M'_{T2}.

We define the function suc_{mcut} that returns, for a decision marking $\mathbb{D}$ and a transition t, the decision marking with NES-status *false* and decision $\top$ corresponding to the system players participating in the transition.

Definition 32 (Successor decision markings from an mcut)
For a decision marking $\mathbb{D}$ and a from there enabled and allowed transition t with an environment place in its precondition, the function $suc_{mcut}(\mathbb{D}, t)$ returns the decision marking corresponding to the system players of the postcondition of t where *false* is the NES-status and $\top$ is the decision. It is defined as

$$suc_{mcut}(\mathbb{D}, t) = \{(id, p', false, \top, K') \in \mathcal{D}_S \mid (id, p, false, T, K) \in \mathbb{D} \wedge p \in \mathcal{P}_S \wedge$$
$$p \Upsilon(\mathbb{D}, id, t) p' \wedge K' = \max(\{lmc(D) \mid D \in \mathbb{D} \setminus \mathbb{D}_{pre(t)}\}) + 1\} \cup$$
$$\{(id', p', false, \top, K') \in \mathcal{D}_S \mid \triangleright \Upsilon(t) p' \wedge K' = \max(\{lmc(D) \mid D \in \mathbb{D}_{pre(t)}\}) + 1 \wedge$$
$$id' = nextID(\mathbb{D}, t, \triangleright \Upsilon(t) p')\}.$$

The transit relation is followed and a new highest number for the last mcut is given to all participating players of the transition. To stay in the range $\{1, \ldots, \max_S\}$ for the last mcut, we need to reduce higher values when all decision tuples with a specific value are removed when a transition with an environment player in its precondition

fires. Therefore, we assume a function $reduce_{mcut}(\mathbb{D})$ that only alters the last mcut in the decision marking $\mathbb{D}$ such that the order between decision tuples and their last mcut is maintained but the minimal number of last mcut numbers is used. For example, when $\mathbb{D}$ contains decision tuples with last mcut 1, 3, 6, and 7, because all decision tuples with last mcut 2, 4, 5, and 8 participate in the transition with an environment place in its precondition, then all decision tuples with last mcut 1 maintain their last mcut 1, all with last mcut 3 get last mcut 2, all with last mcut 6 get last mcut 3, and all with last mcut 7 get last mcut 4.

Definition 33 (Edges *MCUT*)
Edges *MCUT* to let Player 1 fire a transition with an environment place in its precondition from an mcut are defined as

$$\begin{aligned}
&\mathrm{M}CUT = \{(\mathcal{V},\mathcal{V}') \in (V_1 \setminus NDET \cup BAD) \times V \mid \exists t \in \mathcal{T} : \\
&\mathcal{V} = (\mathbb{D}, M_{T2}, BM_1, \ldots, BM_{\max_S}) \wedge \mathcal{V}' = (\mathbb{D}', M'_{T2}, BM'_1, \ldots, BM'_{\max_S}) \wedge \\
&(\forall D \in \mathbb{D} : t2(D) \neq true) \wedge \mathbb{D} \text{ corresponds to an mcut} \wedge M'_{T2} = M_{T2} = \emptyset \wedge \\
&pre(t) \subseteq \mathscr{M}(\mathbb{D}_{pre(t)}) \wedge \mathbb{D}'' = reduce(\mathbb{D} \setminus \mathbb{D}_{pre(t)}) \wedge \mathbb{D}' = \mathbb{D}'' \cup suc_{mcut}(\mathbb{D}'', t) \cup \\
&\{(0, p, false, post(p), 0) \mid p \in post(t) \cap \mathcal{P}_E\} \wedge \forall id \in \{1, \ldots, \max_S\} : \\
&BM'_{id} = \begin{cases} BM_{id} & \text{if } \exists p \in \mathcal{P}_S, b \in \{false, true, end\}, T \subseteq post(p), \\ & K \in \{1, \ldots, \max_S\} : (id, p, b, T, K) \in \mathbb{D}' \setminus \mathbb{D}'' \quad \}. \\ [\,] & \text{otherwise} \end{cases}
\end{aligned}$$

In $\mathcal{V}$, no decision tuple with positive NES-status exists in $\mathbb{D}$, $\mathbb{D}$ corresponds to an mcut, and the marking M_{T2} to repeat in the NES-case remains the empty set. To obtain $\mathbb{D}'$, a transition t that is enabled and allowed from $\mathbb{D}$ is chosen. This transition has an environment place p in its precondition because $\mathbb{D}$ corresponds to an mcut. The last mcut of decision tuples is *reduce*d accounting for the possible free numbers by the removal of decision tuples due to t firing to obtain $\mathbb{D}''$. To simulate transition t firing, decision tuples corresponding to the postcondition of t are added to $\mathbb{D}''$ to obtain $\mathbb{D}'$. All added decision tuples have negative NES-status and $\top$ as decision for decision tuples corresponding to system players (ensured by suc_{mcut}) and $post(p)$ as decision for decision tuples corresponding to the environment player. Backward moves for identifiers in $\mathbb{D}' \setminus \mathbb{D}''$ become empty.

We define three kinds of edges for the NES-case: all three fire a transition only from decision tuples with positive NES-status and the last two re-reach an already reached marking of the NES-case. For the first re-reaching, the marking is the marking to repeat and, for the second re-reaching, the marking is not. The first case finishes the NES-case, whereas the second case makes Player 0 lose the Büchi game immediately.

For the NES-case, we introduce the function suc_{t2} that works exactly like the function suc_S but it requires and produces decision tuples with positive NES-status.

Definition 34 (Edges NES_{fire})
Edges NES_{fire} let Player 0 fire a transition with no environment place in its precondition only from decision tuples with positive NES-status. They let Player 0 make decisions while maintaining positive NES-status and reach a new marking in the NES-case. Edges NES_{fire} are defined as

$$
\begin{aligned}
NES_{fire} = \{(\mathcal{V},\mathcal{V}') \in (V_0 \setminus NDET \cup BAD \cup SYNC_{t2} \cup VAN_{t2}) \times V_0 \mid \\
\mathcal{V} = (\mathbb{D}, M_{T2}, BM_1, \ldots, BM_{\max_S}) \wedge \exists t \in \mathcal{T}, pc \in suc_{t2}(\mathbb{D}, t) : \\
\mathcal{V}' = (\mathbb{D}', M'_{T2}, BM'_1, \ldots, BM'_{\max_S}) \wedge (\exists D \in \mathbb{D} : t2(D) = true) \wedge \\
pre(t) \cap \mathcal{P}_E = \emptyset \wedge pre(t) \subseteq \mathscr{M}(\mathbb{D}_{pre(t)\wedge t2}) \wedge M'_{T2} = M_{T2} \wedge \\
\mathbb{D}' = (\mathbb{D} \setminus \mathbb{D}_{pre(t)\wedge t2}) \cup pc \wedge (\forall \mathbb{D}^M \in \mathcal{R}(\mathcal{N}_{BM_1,\ldots,BM_{\max_S}}[\mathbb{D}_{t2}]) : \mathscr{M}(\mathbb{D}'_{t2}) \neq \mathscr{M}(\mathbb{D}^M_{t2})) \wedge \\
\forall id \in \{1,\ldots,\max_S\} : (((\exists p \in \mathcal{P}_S, T \subseteq post(p), K \in \{1,\ldots,\max_S\} : \\
(id, p, false, T, K) \in pc) \Rightarrow BM'_{id} = BM_{id} \cup [(\mathbb{D}_{pre(t)\wedge t2}, pc)]) \wedge \\
((\forall p \in \mathcal{P}_S, T \subseteq post(p), K \in \{1,\ldots,\max_S\} : \\
(id, p, false, T, K) \notin pc) \Rightarrow BM'_{id} = BM_{id}))\}.
\end{aligned}
$$

In $\mathcal{V}$, a decision tuple with positive NES-status exists in $\mathbb{D}$ and a transition t is chosen such that the firing of this transition has to reach a decision marking with a new underlying marking for the NES-case. The transition t cannot have an environment place in its precondition and is enabled and allowed from $\mathbb{D}$ restricted to decision tuples with positive NES-status. To simulate t firing, decision tuples $\mathbb{D}_{pre(t)\wedge t2}$ corresponding to the precondition of t are removed from $\mathbb{D}$ and then decision tuples corresponding to the postcondition of t are added. All added decision tuples have positive NES-status and a made decision via suc_{t2}. It is ensured via the reachability of decision markings $\mathbb{D}^M_{t2}$ and their underlying marking $\mathscr{M}(\mathbb{D}^M_{t2})$ from the backward moves that the marking in the NES-case is not repeated. The corresponding backward move $(\mathbb{D}_{pre(t)\wedge t2}, pc)$ is added to the backward move of all participating players. The remaining elements of the pair of states stay the same.

Next, we define edges to finish the NES-case successfully.

Definition 35 (Edges NES_{finish})
Edges NES_{finish} let Player 0 fire a transition with no environment place in its precondition only from decision tuples with positive NES-status. They re-reach the marking in which the NES-case started and thereby end the NES-case. Edges NES_{finish} are defined as

$$
\begin{aligned}
NES_{finish} = \{(\mathcal{V},\mathcal{V}') \in (V_0 \setminus NDET \cup BAD \cup UR \cup SYNC_{t2} \cup VAN_{t2}) \times V \mid \\
\mathcal{V} = (\mathbb{D}, M_{T2}, BM_1, \ldots, BM_{\max_S}) \wedge
\end{aligned}
$$

$$\exists t \in \mathcal{T}, pc \in suc_{t2}(\mathbb{D}, t) : \mathcal{V}' = (\mathbb{D}', M'_{T2}, BM'_1, \ldots, BM'_{\max_S}) \wedge$$

$$(\exists D \in \mathbb{D} : t2(D) = true) \wedge pre(t) \cap \mathcal{P}_E = \emptyset \wedge pre(t) \subseteq \mathscr{M}(\mathbb{D}_{pre(t)\wedge t2}) \wedge$$

$$\mathbb{D}'' = (\mathbb{D} \setminus \mathbb{D}_{pre(t)\wedge t2}) \cup pc \wedge \mathscr{M}(\mathbb{D}''_{t2}) = M_{T2} \wedge M'_{T2} = \emptyset \wedge$$

$$(\forall p \in M_{T2} : \exists \mathbb{D}^M \in \mathcal{R}(\mathcal{N}_{BM_1,\ldots,BM_{\max_S}}[\mathbb{D}_{t2}]) : p \notin \mathscr{M}(\mathbb{D}^M_{t2})) \wedge$$

$$\mathbb{D}' = \{(id, p, end, T, K) \in \mathcal{D}_S \mid (id, p, true, T, K) \in \mathbb{D}'\} \cup$$

$$\{(id, p, false, T, K) \in \mathcal{D}_S \mid (id, p, false, T, K) \in \mathbb{D}'\} \wedge$$

$$\mathbb{D}'_{BM} = \{(id, p, end, T, K) \in \mathcal{D}_S \mid (id, p, true, T, K) \in pc\} \wedge$$

$$\forall id \in \{1, \ldots, \max_S\} : ((\exists p \in \mathcal{P}_S, T \subseteq post(p), K \in \{1, \ldots, \max_S\} :$$

$$(id, p, true, T, K) \in ((\mathbb{D}' \setminus \mathbb{D}'') \cup \mathbb{D}'_{BM})) \Rightarrow (((\mathbb{D}'' \setminus pc) \neq (\mathbb{D}' \setminus \mathbb{D}'_{BM}) \Rightarrow$$

$$BM'_{id} = BM_{id} \cup \{(\mathbb{D}_{pre(t)\wedge t2}, \mathbb{D}'_{BM}), (\mathbb{D}'' \setminus (\mathbb{D}' \cup pc), \mathbb{D}' \setminus (\mathbb{D}' \cup \mathbb{D}'_{BM}))\}) \wedge$$

$$((\mathbb{D}'' \setminus pc) = (\mathbb{D}' \setminus \mathbb{D}'_{BM}) \Rightarrow BM'_{id} = BM_{id} \cup \{(\mathbb{D}_{pre(t)\wedge t2}, \mathbb{D}'_{BM})\}))) \wedge$$

$$(\forall p \in \mathcal{P}_S, T \subseteq post(p), K \in \{1, \ldots, \max_S\} :$$

$$(id, p, true, T, K) \notin ((\mathbb{D}' \setminus \mathbb{D}'') \cup \mathbb{D}'_{BM}) \Rightarrow BM'_{id} = BM_{id})\}.$$

In $\mathcal{V}$, a decision tuple with positive NES-status exists in $\mathbb{D}$ and a transition t is chosen such that the firing of this transition reaches a decision marking with the same underlying marking M_{T2} as when the NES-case started. The transition t cannot have an environment place in its precondition and is enabled and allowed from $\mathbb{D}$ restricted to decision tuples with positive NES-status. Simulating t firing would reach the decision marking $\mathbb{D}''$ by removing $\mathbb{D}_{pre(t)\wedge t2}$ from $\mathbb{D}$ and adding pc. The underlying marking of $\mathbb{D}''$ restricted to decision sets with positive NES-status has to be reached for the second time, i.e., it has to be M_{T2}. Every player p in M_{T2} has to be at least once not included in an underlying marking of a decision marking $\mathbb{D}^M_{t2}$ reachable by the backward moves, i.e., every player has to move at least once before completing the NES-case. To obtain $\mathbb{D}'$, all decision tuples with positive NES-status in $\mathbb{D}''$ are set to ended NES-status. The marking M'_{T2} is set to the empty set.

By $\mathbb{D}'_{BM}$, we identify pc with ended NES-status for decision tuples with positive NES-status. When the change from $\mathbb{D}''$ to $\mathbb{D}'$ changed decision tuples not corresponding to the postcondition of t, i.e., not in pc and $\mathbb{D}'_{BM}$, then the backward moves $(\mathbb{D}_{pre(t)\wedge t2}, \mathbb{D}'_{BM})$ and $(\mathbb{D}'' \setminus (\mathbb{D}' \cup pc), \mathbb{D}' \setminus (\mathbb{D}'' \cup \mathbb{D}'_{BM}))$ are added to BM_{id} to obtain BM'_{id} for participating players of the transition with identifier id. Otherwise, the second backward move relates the empty decision marking with the empty decision marking and therefore only the backward move $(\mathbb{D}_{pre(t)\wedge t2}, \mathbb{D}'_{BM})$ is added to BM_{id} to obtain BM'_{id}. The other sequences of backward moves stay the same. No new NES-cases are necessary as one successful disclosure for this branch suffices.

At last, we define edges to finish the NES-case unsuccessfully.

Definition 36 (Edges NES_{bad})
Edges NES_{bad} corresponding to a bad finish of the NES-case because the wrong marking is repeated or not all players moved are defined as

$$
\begin{aligned}
NES_{bad} = \{&(\mathcal{V}, F_N) \in (V_0 \setminus NDET \cup BAD \cup SYNC_{t2} \cup VAN_{t2}) \times \{F_N\} \mid \\
&\mathcal{V} = (\mathbb{D}, M_{T2}, BM_1, \ldots, BM_{\max_S}) \wedge \exists t \in \mathcal{T}, pc \in suc_{t2}(\mathbb{D}, t) : \\
&(\exists D \in \mathbb{D} : t2(D) = true) \wedge pre(t) \cap \mathcal{P}_E = \emptyset \wedge pre(t) \subseteq \mathscr{M}(\mathbb{D}_{pre(t)\wedge t2}) \wedge \\
&\mathbb{D}'' = (\mathbb{D} \setminus \mathbb{D}_{pre(t)\wedge t2}) \cup pc \wedge (\exists \mathbb{D}^M \in \mathcal{R}(\mathcal{N}_{BM_1,\ldots,BM_{\max_S}}[\mathbb{D}_{t2}]) : \mathscr{M}(\mathbb{D}''_{t2}) = \mathscr{M}(\mathbb{D}^M_{t2})) \wedge \\
&(\mathscr{M}(\mathbb{D}''_{t2}) \neq M_{T2} \vee \exists p \in M_{T2} : \forall \mathbb{D}^M \in \mathcal{R}(\mathcal{N}_{BM_1,\ldots,BM_{\max_S}}[\mathbb{D}_{t2}]) : p \in \mathscr{M}(\mathbb{D}^M_{t2}))\}.
\end{aligned}
$$

In $\mathcal{V}$, a transition t with only system places in its precondition has to exist that is enabled and allowed from the decision tuples with positive NES-status in $\mathbb{D}$. The firing of the marking would reach the decision marking $\mathbb{D}''$ by removing $\mathbb{D}_{pre(t)\wedge t2}$ corresponding to the decision tuples for the precondition of t and adding pc corresponding to the decision tuples for the postcondition of t. The underlying marking of the decision tuples with positive NES-status in $\mathbb{D}''$ is reached for the second time, i.e., it is equal to the underlying marking of $\mathbb{D}^M_{t2}$ reachable via the backward moves from $\mathbb{D}_{t2}$ ($\mathbb{D}$ restricted to decision tuples with positive NES-status). The negative behavior occurs because the underlying marking is not equal to M_{T2} or some place p in M_{T2} is part of the entire loop, i.e., part of all $\mathbb{D}^M_{t2}$. This makes NES_{finish} and NES_{bad} mutual exclusive when a marking is repeated in the NES-case.

Definition 37 (Edges NES)
The edges NES for the NES-case in the Büchi game are defined as $NES_{fire} \cup NES_{finish} \cup NES_{bad}$.

We obtain:

Definition 38 (Edges in the Büchi game)
The edges E in the Büchi game are defined as $E = TOP \cup SYS \cup MCUT \cup NES \cup STOP$.

3.4.9. Correctness

We prove assumptions correct that are used in the translation from bounded Petri games with one environment player and bad markings to Büchi games.

Lemma 4 (Sufficiently many identifiers)**.** *There is always a free identifier when the number of system players increases from a transition firing.*

Proof. There are at most $\max_S$ system players in the Petri game. When the number of system players can increase by x, then there are at most $\max_S - x$ system players in the Petri game. As there are $\max_S$ unique identifiers for system players, there are x free identifiers. □

Lemma 5 (Sufficiently many last mcuts)**.** *There is always a free last mcut when a transition with an environment place and a system place in its precondition fires.*

Proof. System players remember their last mcut and there are at most $\max_S$ of them in the Petri game. Therefore, at most $\max_S$ last mcuts can exist. Whenever a system player participates in a transition with the environment player, its last mcut becomes free, other last mcuts are reordered to keep their order, and a new last mcut can be added. □

We prove that winning strategies for the Büchi game can be translated to winning strategies for the system players in the encoded Petri game. First, we define the translation. Second, we define an equivalence relation between decision markings in the strategy for Player 0 in the Büchi game and cuts in the strategy for the system players in the Petri game. Third, we use this relation to prove the translation correct.

Definition 39 (Translation from Büchi game strategies to Petri game strategies) ——
In the following, we translate winning strategies for Player 0 in the Büchi game into winning strategies for the system players in the Petri game. By f, we identify the winning strategy for Player 0 in the Büchi game $\mathbb{G} = (V, V_0, V_1, I, E, F)$ which simulates the Petri game $\mathcal{G} = (\mathcal{P}_S, \mathcal{P}_E, \mathcal{T}, \mathcal{F}, In, \mathcal{M}_B)$. From the initial state I, the strategy f generates a tree T_f which is possibly infinite. In T_f, the nodes are labeled with states from V and the root is labeled with I. A node N labeled with a V_0-state $\mathcal{V}$ has a unique successor node labeled with the state $f(w)$ with w being the sequence of labels from the root to N. A node N labeled with a V_1-state $\mathcal{V}$ has for each successor state $(\mathcal{V}, \mathcal{V}') \in E$ a successor node in T_f labeled with $\mathcal{V}'$.

We traverse the nodes of T_f in a breadth-first order and inductively construct a strategy $\sigma = (\mathcal{N}^\sigma, \lambda^\sigma)$ for $\mathcal{G}$. We associate to each node labeled with a state $\mathcal{V}$ a cut C in the strategy σ under construction such that $\lambda[C] = \mathscr{M}(DS(\mathcal{V}))$ holds, i.e., the cut and the decision marking of the state represent the same marking. To I, we associate a cut C_0 labeled with the initial marking of $\mathcal{N}$, i.e., $\lambda[C_0] = In$. Then, $\lambda[C_0] = \mathscr{M}(DS(I))$ holds.

Suppose now the breadth-first traversal has reached a node in T_f labeled with a V_0-state $\mathcal{V}$ to which the cut C is associated. Then, there is a unique successor node in T_f labeled with $\mathcal{V}'$. If the edge from $\mathcal{V}$ to $\mathcal{V}'$ resolves $\top$, nothing is added to the strategy, and C is associated with $\mathcal{V}'$ as well. If the edge from $\mathcal{V}$ to $\mathcal{V}'$ corresponds to a transition t with $\lambda(C_{pre}) = pre(t)$ for $C_{pre} \subseteq C$ and $\mathcal{V}$ has less or equally many decision tuples with positive NES-status than $\mathcal{V}'$ (i.e., $\sum_{D \in \mathcal{D} \wedge t2(D)=true} DS(\mathcal{V})(D) \leq \sum_{D \in \mathcal{D} \wedge t2(D)=true} DS(\mathcal{V}')(D)$), then we extend the strategy with a new transition t' and new places NP such that $\lambda[NP] = post(t)$, $\lambda(t') = t$, $pre(t') = C_{pre}$, and $post(t') = NP$. We associate to $\mathcal{V}'$ the cut $C' = NP + (C - C_{pre})$ containing the new places NP and the places in C that did not participate in t.

If the edge from $\mathcal{V}$ to $\mathcal{V}'$ corresponds to a transition t with $\lambda(C_{pre}) = pre(t)$ for $C_{pre} \subseteq C$ and $\mathcal{V}$ has a lower number of decision tuple with ended NES-status than $\mathcal{V}'$ because the marking to repeat in the NES-case is reached for the second time, i.e., $\sum_{D \in \mathcal{D} \wedge t2(D)=end} DS(\mathcal{V})(D) < \sum_{D \in \mathcal{D} \wedge t2(D)=end} DS(\mathcal{V}')(D)$, then a unique predecessor state $\mathcal{V}^{1st}$ exists such that $\mathscr{M}(DS(\mathcal{V})) - pre(t) + post(t) = \mathscr{M}(DS(\mathcal{V}^{1st}))$. The state $\mathcal{V}^{1st}$ represents the first occurrence of the marking $\mathscr{M}(DS(\mathcal{V}^{1st}))$ that is reached for a second time when firing t from $\mathcal{V}$. By construction, only transitions from decision tuples with positive NES-status fired between $\mathcal{V}^{1st}$ and $\mathcal{V}$. By C^{1st}, we identify the cut associated with $\mathcal{V}^{1st}$. We extend the strategy with a new transition t' and new places NP such that $\lambda[NP] = post(t)$, $\lambda(t') = t$, $pre(t') = C_{pre}$, and $post(t') = NP$. From $C' = NP + (C - C_{pre})$ containing the new places NP and the places in C that did not participate in t, we repeat the strategy from C^{1st} to C' infinitely often with new places and new transitions. The breadth-first search needs to continue with the situation before the NES-case. Therefore, we associate to $\mathcal{V}'$ the cut C^{1st}.

Suppose now the breadth-first traversal has reached a node in T_f labeled with a V_1-state $\mathcal{V}$ to which the cut C is associated. Then, for each successor node in T_f labeled with $\mathcal{V}'$, a corresponding environment transition t exists with $\lambda(C_{pre}) = pre(t)$ for $C_{pre} \subseteq C$ and we extend the strategy with a new transition t' and new places NP such that $\lambda[NP] = post(t)$, $\lambda(t') = t$, $pre(t') = C_{pre}$, and $post(t') = NP$. We associate to $\mathcal{V}'$ the cut $C' = NP + (C - C_{pre})$ containing the new places NP and the places in C that did not participate in t.

To prove that the constructed strategy from Definition 39 for the system players in the Petri game is winning, we prove that the considered decision markings of every winning strategy for Player 0 in the Büchi game are equivalent to the reachable cuts in the corresponding strategy for the system players in the Petri game. Notice that we no longer assume winningness of strategies. Notice further that all non-winning strategies in the Büchi game are by construction *minimal* in the sense that the strategy reaches a self-loop in F_N for losing behavior (a deadlock without termination, a bad marking, firing a transition from a nondeterministic decision, or a losing behavior in the NES-case). A decision marking is *considered* if it is directly reachable or indirectly reachable via backward moves in a state in the Büchi game. By definition, reaching F_B or F_N does not consider a decision marking. The *equivalence relation* between a decision marking and a cut checks that a place p with decision d is in the decision marking if and only if a place p' in the cut exists such that p' represents p in the original Petri game, $post(p')$ represents d, and the causal past of both p and p' is equivalent.

Definition 40 (Equivalence relation between decision markings and cuts)
The equivalence relation $\equiv$ between decision markings and cuts is defined as

$$\begin{aligned}\mathbb{D} \equiv C \iff ((&\forall (p,b,T) \in \mathbb{D} : \exists^{=1} c \in C : p = \lambda(c) \wedge T = \lambda(post(c)) \wedge \\ &past^{\mathcal{T}}((p,b,T)) = past^{\mathcal{T}}(c)) \wedge (\forall c \in C : \exists^{=1} (p,b,T) \in \mathbb{D} : \\ &p = \lambda(c) \wedge T = \lambda(post(c)) \wedge past^{\mathcal{T}}((p,b,T)) = past^{\mathcal{T}}(c)))\end{aligned}$$

where the term $past^{\mathcal{T}}(c)$ is the causal past represented by a sequence of transitions and the term $past^{\mathcal{T}}((p, b, T))$ is the causal past represented by a sequence of transition subtracted by the used backward moves.

The cuts and the decision markings can come from the strategies for the Petri game and the corresponding Büchi game. The function $past^{\mathcal{T}}(c)$ returns the sequence of transitions that leads from the initial marking In in $\mathcal{N}^{\sigma}$ to C following the order of the breadth-first search in Definition 39. We apply λ to each transition to obtain the original transitions. The function $past^{\mathcal{T}}((p, b, T))$ returns the sequence of transitions minus the used backward moves that match the edges that lead to the state $\mathcal{V}$ from which (p, b, T) was reached via backward moves. The equivalence of sequences is up to reordering of transitions with disjoint preconditions which can be necessary for $past^{\mathcal{T}}((p, b, T))$ as the order of $past^{\mathcal{T}}(c)$ is fixed by the breadth-first search.

Lemma 1 (From Büchi game strategies to Petri game strategies)**.** *If Player 0 has a winning strategy in the Büchi game, then there exists a winning strategy for the system players in the Petri game.*

Proof. We translate a winning strategy for Player 0 in the Büchi game into a winning strategy for the system players in the Petri game as defined in Definition 39. To prove that the constructed strategy is winning, we prove that the considered decision markings of every winning strategy for Player 0 in the Büchi game are equivalent as defined in Definition 40 to the reachable cuts in the corresponding strategy for the system players in the Petri game. Then, losing behavior in the strategy for Player 0 in the Büchi game occurs if and only if losing behavior in the strategy for the system players in the Petri game occurs. The translation in Definition 39 also applies to non-winning strategies for Player 0 in the Büchi game. Reaching a state corresponding to a deadlock but no termination, a bad marking, a nondeterministic decision, or losing behavior in the NES-case leads to a deadlock in the Petri game. We consider both directions for a winning strategy f for Player 0 in the Büchi game and the corresponding strategy σ for the system players in the Petri game:

- We show that if a decision marking $\mathbb{D}$ is considered in a by f reachable state of the Büchi game, then a reachable cut C exists in σ such that $\mathbb{D} \equiv C$ holds. If $\mathbb{D}$ is not considered via backward moves, then the word w exists to reach the state of $\mathbb{D}$. The word w^{-} is defined as w without symbols $\top$. Firing transitions in the order of w^{-} in the strategy leads to a cut C. By the strategy translation from Definition 39, w^{-} is applicable to σ and, by the construction of the Büchi game, the precondition and postcondition of the used transitions are the same in both strategies. Therefore, $\mathbb{D} \equiv C$ holds. If $\mathbb{D}$ is reached via backward moves, then the word w to reach the state of $\mathbb{D}$ and the word w^{back} of applied backward moves exist. We define the subtraction of w^{back} from w^{-} from the end of both words. Firing transitions in the order of $w^{-} - w^{back}$ in the strategy leads to a cut C. By the strategy translation from Definition 39, the backward moves can be subtracted

from the word, $w^- - w^{back}$ is applicable to σ, and, by the construction of the Büchi game, the precondition and postcondition of the used transitions are the same in both strategies. Therefore, $\mathbb{D} \equiv C$ holds.

- We show that if a reachable cut C exists in σ, then a decision marking is considered in a by f reachable state of the Büchi game. The cut C is reached via the sequence of transitions T. We only retain transitions with an environment place in their precondition in T. The choices of system players are already represented by σ. We start from the initial marking of σ and obtain a sequence of transitions w that is applicable to f and reaches C by recursively applying the following steps: If a NES-case starts from the current cut (i.e., a minimal set of system players in the cut can play infinitely together without synchronizing with the environment player) and the underlying marking of the cut is the first marking to repeat in this NES-case, then we add the transitions until the repetition of the marking occurs to w and update the current cut accordingly. By the strategy translation from Definition 39, at most on NES-case can occur. Otherwise, we try to fire a transition with only system places in its precondition. If this is not possible, we try to fire a transition with an environment places in its precondition. We pick the environment transition corresponding to the first element of T and afterward remove it from T. In both cases, the current cut becomes the cut reached after firing the transition.

  When none of these cases apply and there is a transition left in T, we need to add transitions with only system places in their precondition which are rolled back by backward moves to reach an equivalent decision marking to the cut C. In this case, we fire a maximal sequence of transitions with only system places in their precondition such that only transitions with an environment place in the precondition are enabled. The enabled transitions include the remaining transition in T. These fired transitions and all following transitions with only system places in their precondition are collected to be used as backward moves. We apply the recursive steps from before again. The first iteration removes the last transition from T and afterward a finite number of transitions with only system places in their precondition are added. Notice that at most one transition can be left in T because for all but the last transition we fire it as late as possible at an mcut. This procedure obtains w'. We add steps to w' to make explicit decisions removing $\top$ according to the postcondition of the respective players and indicate the beginning of the NES-case by changing the corresponding decision sets from negative to positive NES-status to obtain w. By the construction of the Büchi game, w is applicable to the Büchi game. Either w directly leads to a decision marking $\mathbb{D}$ such that $\mathbb{D} \equiv C$ holds or some of the collected backward moves and of the backward moves in the NES-case are applied to the state in the Büchi game leading to a decision marking $\mathbb{D}'$ such that $\mathbb{D}' \equiv C$ holds.

It remains to show that it suffices to only apply backward moves from one state, i.e., at most one transition can be left in T. By way of contradiction, assume that there are

two states when backward moves are applied. Either the first earlier application is also possible at the second later application because the corresponding system players have not moved or performing the first backward moves makes it impossible to continue to the second backward move because the environment player is part of a transition in between and the used backward move is negated.

It also remains to show that the constructed strategy for the Petri game is, in fact, a strategy, i.e., satisfies justified refusal and is deterministic and deadlock-avoiding. The strategy is deterministic and deadlock-avoiding because *all* reachable markings are tested to exactly fulfill these requirements by construction. The usage of backward moves ensures that all reachable markings are checked. Justified refusal is fulfilled because system players can only disallow transitions from the postcondition in the original Petri game. Meanwhile, transitions with only an environment place in their precondition are always added. Therefore, every possible transition not in the Petri game has at least one system place in its precondition that universally forbids this transition in the original Petri game. Because every reachable marking in the strategy is checked to have only deterministic decisions, we know that it is impossible to overlook transitions that could violate justified refusal when building the strategy from the Büchi game. Remember that the Büchi game fires transitions with the environment player as late as possible at mcuts, restricting the order in which transitions are added.

As we showed that a strategy for Player 0 in a Büchi game and the corresponding strategy for the system players in a Petri game visit equivalent decision markings and cuts, we can conclude that for a winning strategy for Player 0 in a Büchi game no losing situation in the strategy for the system players in the Petri game can occur. Therefore, the strategy for the system players in the Petri game is winning. □

Before we can prove the reverse direction, we have to show that it suffices to only consider *minimal* winning strategies for the system players in Petri games when translating them into strategies for Player 0 in the Büchi game. This is necessary because for not minimal but winning strategies for the system players in Petri games, the backward moves in the corresponding Büchi game might falsely classify the corresponding strategy for Player 0 in the Büchi game as losing. We prove that this cannot occur for minimal winning strategies and that all winning strategies can be made minimal while maintaining their winningness.

We define *minimal strategies for the system players in Petri games* based on *useless repetitions* [Gim17] which are defined for control games [GGMW13] played on asynchronous automata [Zie87]. Control games and Petri games can be translated into the converse formalism at an exponential blow-up [BFH19a]. More information on control games can be found in Section 1.7.2.

Before defining our useless repetitions formally, we recall the prefix [KKV03] of a Petri net or a Petri game and define the suffix of a prefix.

Definition 41 (Prefix [KKV03])
A branching process $\iota_{pre} = (\mathcal{N}^{\iota}_{pre}, \lambda^{\iota}_{pre})$ is a *prefix* of a branching process $\iota = (\mathcal{N}^{\iota}, \lambda^{\iota})$, denoted $\iota_{pre} \sqsubseteq_{pre} \iota$, if $\mathcal{N}^{\iota}_{pre}$ is a subnet of $\mathcal{N}^{\iota}$ (i.e., $\mathcal{P}^{\iota}_{pre} \subseteq \mathcal{P}^{\iota}$, $\mathcal{T}^{\iota}_{pre} \subseteq \mathcal{T}^{\iota}$, $In^{\iota}_{pre} = In^{\iota}$, and

$\mathcal{F}^\iota_{pre} = \mathcal{F}^\iota \restriction \mathcal{P}^\iota_{pre} \times \mathcal{T}^\iota_{pre} \cup \mathcal{T}^\iota_{pre} \times \mathcal{P}^\iota_{pre})$ such that if $t \in \mathcal{T}^\iota_{pre}$ and $(p,t) \in \mathcal{F}^\iota$ or $(t,p) \in \mathcal{F}^\iota$, then $p \in \mathcal{P}^\iota_{pre}$, if $p \in \mathcal{P}^\iota_{pre}$ and $(t,p) \in \mathcal{F}^\iota$, then $t \in \mathcal{T}^\iota_{pre}$, and λ^ι_{pre} is the restriction of λ^ι to $\mathcal{P}^\iota_{pre} \cup \mathcal{T}^\iota_{pre}$.

Definition 42 (Suffix)
A branching process $\iota_{suf} = (\mathcal{N}^\iota_{suf}, \lambda^\iota_{suf})$ is a *suffix* of a branching process $\iota = (\mathcal{N}^\iota, \lambda^\iota)$ if there exists a subset M of a reachable marking in $\mathcal{N}^\iota$, $\mathcal{N}^\iota_{suf} = \mathcal{N}^\iota[M]$ without unreachable places, transitions, and flows, and λ^ι_{suf} is the restriction of λ^ι to $\mathcal{P}^\iota_{suf} \cup \mathcal{T}^\iota_{suf}$.

We require that λ^σ, λ^ι_{pre}, and λ^ι_{suf} are restricted corresponding to the Petri net they belong to.

The *future* in $\mathcal{N}$ of a node x in $\mathcal{N}$ is defined as the set $fut(x) = \{y \in \mathcal{P} \cup \mathcal{T} \mid x \leq y\}$.

Definition 43 (Useless repetition)
A finite prefix $\iota_{pre} = (\mathcal{N}^\iota_{pre}, \lambda^\iota_{pre})$ is a *useless repetition* of a winning strategy $\sigma = (\mathcal{N}^\sigma, \lambda^\sigma)$ if and only if there exists a suffix $\iota_{suf} = (\mathcal{N}^\iota_{suf}, \lambda^\iota_{suf})$ of the prefix ι_{pre} such that

(1) the suffix contains only system places and the definition of the prefix has either removed all or no outgoing transitions of the places in the suffix (i.e., $\forall p \in \mathcal{P}^\iota_{suf} : \forall t \in \mathcal{T} : (p,t) \in \mathcal{F} \Rightarrow ((p,t) \in \mathcal{F}^\iota_{pre} \vee post^{\iota_{pre}}(p) = \emptyset)$),

(2) the initial marking and the final marking of the suffix represent the same marking in the original Petri game (i.e., $\exists^{=1} M \in \mathcal{R}(\mathcal{N}^\iota_{suf}) : (\forall t \in \mathcal{T}^\iota_{suf} : pre^{\iota_{suf}}(t) \nsubseteq M) \wedge \lambda^\iota_{suf}[In^\iota_{suf}] = \lambda^\iota_{suf}[M]$),

(3) the decisions of the system players in the initial marking of the suffix differ from the decisions of the system players in the final marking of the suffix (i.e., $\exists^{=1} M \in \mathcal{R}(\mathcal{N}^\iota_{suf}) : (\forall t \in \mathcal{T}^\iota_{suf} : pre^{\iota_{suf}}(t) \nsubseteq M) \wedge \exists p^\iota_{suf} \in In^\iota_{suf}, p \in M : \lambda(p^\iota_{suf}) = \lambda(p) \wedge \exists t^\iota_{suf} \in post^\sigma(p^\iota_{suf}) : pre^{\iota_{suf}}(t^\iota_{suf}) \subseteq In^\iota_{suf} \wedge \forall t \in post^\sigma(p) : \lambda(t^\iota_{suf}) = \lambda(t) \Rightarrow pre^\sigma(t) \nsubseteq M$), and

(4) the suffix only contains synchronizations between the players when the synchronization also occurred before the suffix but after the last synchronization with the environment player (i.e., $\forall t \in \mathcal{T}^\iota_{suf} : pre^{\iota_{suf}}(t) > 1 \Rightarrow \forall : p_1, p_2 \in In^\iota_{suf} \cap past^{\iota_{pre}}(t) : \exists t^\iota_{pre} \in \mathcal{T}^\iota_{pre} : fut^{\mathcal{N}^\iota_{pre}}(t^\iota_{pre}) \cap \mathcal{P}_E^{\iota_{pre}} = \emptyset \Rightarrow t^\iota_{pre} \in past^{\iota_{pre}}(p_1) \wedge t^\iota_{pre} \in past^{\iota_{pre}}(p_2)$).

We utilize useless repetitions to simplify the behavior of system players in the NES-case and between successors of mcuts and the next mcut (cf. Condition (1)). Therefore, our application of useless repetitions is independent of the environment. Condition (2) carries over from [Gim17] to our usage. We add Condition (3) to handle infinite strategies in the case of Petri games. It ensures that the suffix is not infinitely often repeated immediately after the suffix. This would lead to infinitely many applications of useless repetitions at one position. We prevent this by ensuring that the transition after the

suffix is *not* the same as at the beginning of the suffix. We add Condition (4) to require that a useless repetition only contains exchange of causal past via synchronization between the system players if this exchange already occurred before the useless repetition but after the respective last successor of an mcut of the system players. Notice that a suffix of a useless repetition has a unique final marking and the removal of outgoing transitions can only happen at the unique final marking of the suffix.

Our usage of useless repetitions is far more restricted than for the original definition [Gim17]. There it is used to decide the existence of a winning strategy in *decomposable games* by bounding the size of strategies without useless repetitions. The original definition of useless repetitions further requires that the future is the same when skipping and not skipping the suffix. For our approach, this is fulfilled by construction.

Definition 44 (Minimal strategies in Petri games)
A *minimal strategies for the system players in Petri games* has no useless repetitions.

We skip useless repetitions in winning strategies by shortcuts.

Definition 45 (Shortcut)
A strategy $\iota_{short} = (\mathcal{N}^\iota_{short}, \lambda^\iota_{short})$ is a *shortcut* via useless repetition $\iota_{pre} = (\mathcal{N}^\iota_{pre}, \lambda^\iota_{pre})$ with suffix $\iota_{suf} = (\mathcal{N}^\iota_{suf}, \lambda^\iota_{suf})$ of a strategy $\iota = (\mathcal{N}^\iota, \lambda^\iota)$ if and only if $\mathcal{P}^\iota_{short} = \mathcal{P}^\iota \setminus (\mathcal{P}^\iota_{suf} \setminus In^\iota_{suf})$, $\mathcal{T}^\iota_{short} = \mathcal{T}^\iota \setminus \mathcal{T}^\iota_{suf}$, $\mathcal{F}^\iota_{short}$ satisfies $\mathcal{F}^\iota_{suf} \cap \mathcal{F}^\iota_{short} = \emptyset \wedge \exists^{=1} M \in \mathcal{R}(\mathcal{N}^\iota_{suf}) : (\forall t \in \mathcal{T}^\iota_{pre} : pre^{\iota_{pre}}(t) \not\subseteq M) \wedge \forall p \in M : \forall (p,t) \in \mathcal{F}^\iota : (p,t) \notin \mathcal{F}^\iota_{short} \wedge \exists^{=1} p^\iota_{suf} \in In^\iota_{suf} : \lambda(p^\iota_{suf}) = \lambda(p) \wedge (p^\iota_{suf}, t) \in \mathcal{F}^\iota_{short}$, $In^\iota_{short} = In^\iota$, and λ^ι_{short} is the restriction of λ^ι to $\mathcal{P}^\iota_{short} \cup \mathcal{T}^\iota_{short}$.

A shortcut is obtained by removing the suffix of a useless repetition from a given strategy. We identify the reachable final marking of the suffix by M. The behavior following this marking in the original strategy is pulled forwarded to the initial marking of the suffix. All places, transitions, and flows of the suffix between its initial marking and the marking M are removed.

In [Gim17], removing useless repetitions by shortcuts is applied only to finite strategies because the winning condition is termination in final states whereas we also apply it to infinite strategies. This is no problem because it is only applied to the finite parts between successors of mcuts and the next mcut and to system players in the NES-case. Therefore, we might require infinitely many shortcuts but each of them is well-defined. We prove that every winning strategy can be reduced to a minimal winning strategy by removing all useless repetitions.

Lemma 6 (Minimal winning strategies)**.** *Removing a useless repetition via a shortcut from a winning strategy in a Petri game results again in a winning strategy in the Petri game.*

Proof. The construction of the shortcut only removes transitions with only system places in their precondition, that are not synchronizing with other players and have exchanged their history before the removed part, and then continues with the existing future. Therefore, the strategy is still deterministic and satisfies justified refusal. Fewer

markings (no new markings) are reached and therefore the shortcut also avoids bad markings. □

Removing all useless repetitions and updating the winning strategy accordingly in each step gives a minimal winning strategy.

Before we can prove the reverse direction, we also have to show that unnecessary NES-cases in strategies for the system players in Petri games can be removed. The Büchi game allows at most one NES-case per branch by preventing the change from negative to positive NES-status for decision tuples from the point onwards that a decision tuple with ended NES-status exists in the decision marking. We use the following definition to identify unnecessary NES-cases in strategies for the system players in Petri games.

Definition 46 (System place for the NES-case)
Given a strategy σ for the system players in a Petri game and a cut C in the Petri game, the function $nes_\sigma(C)$ calculates all minimal sets of system places from C that satisfy the NES-case in the Büchi game and is defined as

$$
\begin{aligned}
nes_\sigma(C) = \{C_{t2} \subseteq C \mid (\forall p \in C_{t2} : (\forall p_E \in \mathcal{P}_E^\sigma : p_E \not\leq p \Rightarrow \forall q \in fut^\sigma(p) : \\
p_E \not\leq q) \wedge fut^\sigma(p) \text{ is infinite}) \wedge (\exists t_1, \ldots, t_n \in \mathcal{T}^\sigma : \exists C_1, \ldots, C_n \subseteq \mathcal{P}^\sigma : \\
C_{t2}[t_1\rangle C_1[t_2\rangle \ldots [t_n\rangle C_n \wedge \mathscr{M}(C_{t2}) = \mathscr{M}(C_n) \wedge \forall i \in 1, \ldots, n-1 : \\
\mathscr{M}(C_{t2}) \neq \mathscr{M}(C_i)) \wedge (\forall C'_{t2} \subseteq C_{t2} : C_{t2} \setminus C'_{t2} \notin nes_\sigma(C)) \wedge \\
(\forall p \in C_{t2} : \exists i \in \{1, \ldots, n\} : p \notin C_i)\}.
\end{aligned}
$$

We use n to search for repetitions of markings in the underlying Petri net. Therefore, n can be upper bounded by the exponential size of the reachability graph of the underlying Petri net. The cut C_{t2} corresponding to the NES-case cannot synchronize with the environment player and has to have an infinite future. Further, there has to be a sequence of transitions such that the marking of C_{t2} is repeated after firing the sequence but not during firing it. Additionally, C_{t2} has to be minimal in the sense that no places can be removed from C_{t2} while C_{t2} remains a cut corresponding to the NES-case. At last, each place has to be left at least once between C and C_{t2} at a position i.

We remove unnecessary NES-cases in strategies for the system players in Petri games as follows:

Lemma 7 (One NES-case per branch)**.** *For each winning strategy σ for the system players in a bounded Petri game with one environment player and bad markings, there exists a winning strategy σ' for the system players with, for each reachable cut, at most one minimal set of system players playing infinitely together without synchronizing with the environment player. The winning strategy σ' has* at most one NES-case per branch *in the reachability graph.*

Proof. We initialize σ' with the initial marking of σ. We fire enabled transitions from the initial marking in σ in breadth-first search manner to obtain all reachable cuts. For

each reached cut C (including the initial marking), we calculate $nes_\sigma(C)$. We copy fired transitions t and their flow and postcondition to σ' unless either the precondition of t is not part of σ' or $nes_\sigma(C)$ contains at least one set, the precondition of t is a subset or equal to one set in $nes_\sigma(C)$, and one of the two following conditions holds:

1. The fired transition t is not added if there exists a disjoint NES-case in the past of cut C.

2. The fired transition t is not added if there exists no disjoint NES-case in the past of cut C but an outgoing transition from another set in $nes_\sigma(C)$ is already added to strategy σ'.

This construction allows for each reachable cut at most one minimal set of system players playing infinitely together without synchronizing with the environment player from the point onward that the terminated system players have the same position of the environment player in their causal past, i.e., at the latest after the first repetition of a marking for this set of system players playing infinitely together without synchronizing with the environment player. The winningness of σ translates to σ' because the termination of the system players is deadlock-avoiding as another set of system players plays infinitely. Furthermore, fewer markings are reached which is winning for the winning condition of bad markings. □

In the following, we assume that minimal strategies for the system players in a Petri game have at most one NES-case per branch in the reachability graph. We can now prove the reverse direction from strategies for the players in Petri games to strategies for Player 0 in the corresponding Büchi game. We define the translation from strategies for the players in Petri games to strategies for Player 0 in the corresponding Büchi game and prove it correct.

Definition 47 (Translation from Petri game strategies to Büchi game strategies) —— In the following, we translate minimal winning strategies for the system players in the Petri game into winning strategies for Player 0 in the Büchi game. Given a winning strategy $\sigma = (\mathcal{N}^\sigma, \lambda^\sigma)$ of $\mathcal{G}$ and a prefix $w \in V^* \cdot V_0$ of a play of $\mathbb{G}$ simulating $\mathcal{G}$, we compute the choice $f(w)$ of a strategy f for $\mathbb{G}$. Let w^- result from w by deleting all states containing $\top$. Starting with the initial marking In^σ of $\mathcal{N}^\sigma$ and firing the sequence of transitions corresponding to w^- in σ, we arrive at a cut C of σ, which by definition of V_0 does not correspond to an mcut. We assume a function $cut(id, p, C)$ that returns for an identifier $id \in \{1, \ldots, \max_S\}$ and a place p in the Petri game the corresponding place p_{cut} in C of σ.

There are three cases: the first one resolves $\top$, the second one fires a transition with only system places in its precondition not in the NES-case, and the third one does so in the NES-case. For the three cases, we apply the definitions of *TOP*, *SYS*, and *NES* for a given fired transition and a by $nes_\sigma(C)$ given (possibly empty) set of multisets over system places for the NES-case. Due to Lemma 7, $nes_\sigma(C)$ contains at most one set and if it contains a set no NES-case occurred in the past of the current cut C.

Case 1 (*TOP*). If $last(w) \in V_0$ contains $\top$ in the decision marking, we make decisions and start the NES-case if $nes_\sigma(C)$ is not empty. The state $last(w)$ does not contain any decision tuples with positive NES-status in the decision marking. Formally, we are at a state $last(w) = (\mathbb{D}, M_{T2}, BM_1, \ldots, BM_{\max_S})$ with corresponding cut C. We build the successor state $(\mathbb{D}', M'_{T2}, BM'_1, \ldots, BM'_{\max_S})$ that our strategy picks. We pick the unique $C_{t2} \in nes_\sigma(C)$ if $nes_\sigma(C)$ is not empty. Otherwise, $C_{t2} = \emptyset$. We define $\mathbb{D}' = \{(id, p, b, \lambda(post^\sigma(p_{cut})), K) \in \mathcal{D} \mid (id, p, b, T, K) \in \mathbb{D} \wedge cut(p, id, C) = p_{cut} \wedge p_{cut} \in C \setminus C_{t2}\} \cup \{(id, p, true, \lambda(post^\sigma(p_{cut})), K) \in \mathcal{D} \mid (id, p, false, T, K) \in \mathbb{D} \wedge cut(p, id, C) = p_{cut} \wedge p_{cut} \in C_{t2}\}$. If C_{t2} is not the empty set, then $M'_{T2} = \mathscr{M}(\mathbb{D}'_{t2})$ and the remaining elements stay the same. If C_{t2} is the empty set, then all elements except for $\mathbb{D}'$ stay the same.

Case 2 (*SYS*). If $last \in V_0$ contains no $\top$ and no decision tuple in the decision marking has positive NES-status, an enabled transition fires to the cut C' and the NES-case is started if $nes_\sigma(C)$ is not empty. As C is not an mcut, some transition with only system places in its precondition is enabled in C. We choose one of the enabled transitions and call it t. Let t lead to a successor cut C'. Formally, we are at a state $last(w) = (\mathbb{D}, M_{T2}, BM_1, \ldots, BM_{\max_S})$ with corresponding cut C. We build the successor state $(\mathbb{D}', M'_{T2}, BM'_1, \ldots, BM'_{\max_S})$ that our strategy picks. We pick the unique $C_{t2} \in nes_\sigma(C')$ if $nes_\sigma(C')$ is not empty. Otherwise, $C_{t2} = \emptyset$. We define $\mathbb{D}' = \{(id, p, b', T, K) \in \mathcal{D} \mid (id, p, b, T, K) \in \mathbb{D} \wedge (p \notin pre(t) \vee t \notin T) \wedge cut(p', id) = p_{cut} \wedge (p_{cut} \in (C \setminus C_{t2}) \Rightarrow b' = b) \wedge (p_{cut} \in C_{t2} \Rightarrow b' = true)\} \cup \{(id, p', b', \lambda(post^\sigma(p_{cut})), K') \in \mathcal{D} \mid (id, p, b, T, K) \in \mathbb{D} \wedge p\Upsilon(\mathbb{D}, id, t)p' \wedge p \in pre(t) \wedge t \in T \wedge cut(p', id) = p_{cut} \wedge (p_{cut} \in (C \setminus C_{t2}) \Rightarrow b' = false) \wedge (p_{cut} \in C_{t2} \Rightarrow b' = true) \wedge K' = \max(\{lmc(D) \mid D \in \mathbb{D}_{pre(t)}\})\} \cup \{(id', p', b', \lambda(post^\sigma(p_{cut})), K') \in \mathcal{D} \mid \rhd\Upsilon(t)p' \wedge id' = nextID(\mathbb{D}, t, \rhd\Upsilon(t)p') \wedge cut(p', id) = p_{cut} \wedge (p_{cut} \in (C \setminus C_{t2}) \Rightarrow b' = false) \wedge (p_{cut} \in C_{t2} \Rightarrow b' = true) \wedge K' = \max(\{lmc(D) \mid D \in \mathbb{D}_{pre(t)}\}$. The backward moves of all participating players are updated accordingly, i.e., we add $(\mathbb{D}_{pre(t)}, \mathbb{D}'_{post(t)})$ and potentially the backward move for all changes of NES-status of decision tuples that did not participate in t. If C_{t2} is not the empty set, then $M'_{T2} = \mathscr{M}(\mathbb{D}'_{t2})$ and the remaining elements stay the same. If C_{t2} is the empty set, then all remaining elements stay the same.

Case 3 (*NES*). If $last \in V_0$ contains a decision tuple with positive NES-status in the decision marking, an enabled transition fires and we finish the NES-case if the indicated marking to repeat in the NES-case is reached. The state $last(w)$ does not contain any $\top$ in the decision marking. As at least one decision tuple in the decision marking from *last* corresponding to cut C has positive NES-status, some transition with only system places in its precondition is enabled in C and has a future of infinite length and without synchronization with the environment player. We choose one of the these transitions and call it t. Let t lead to a successor cut C'. Formally, we are at a state $last(w) = (\mathbb{D}, M_{T2}, BM_1, \ldots, BM_{\max_S})$ with corresponding cut C. We build the successor state $(\mathbb{D}', M'_{T2}, BM'_1, \ldots, BM'_{\max_S})$ that our strategy picks. Two cases can occur: either M_{T2} is reached for the second time or not.

- If M_{T2} is *not* reached for the second time, then we define $\mathbb{D}' = (\mathbb{D} \setminus \mathbb{D}_{pre(t) \wedge t2}) \cup \{(id, p', true, \lambda(post^\sigma(p_{cut})), K') \in \mathcal{D} \mid (id, p, b, T, K) \in \mathbb{D} \wedge p\Upsilon(\mathbb{D}, id, t)p' \wedge p \in pre(t) \wedge t \in T \wedge cut(p', id) = p_{cut} \wedge K' = \max(\{lmc(D) \mid D \in \mathbb{D}_{pre(t)}\})\} \cup$

$\{(id', p', true, \lambda(post^{\sigma}(p_{cut})), K') \in \mathcal{D} \mid \triangleright\Upsilon(t)p' \wedge K' = \max(\{lmc(D) \mid D \in \mathbb{D}_{pre(t)}\}) \wedge cut(p', id) = p_{cut} \wedge id' = nextID(\mathbb{D}, t, \triangleright\Upsilon(t)p')\}$, i.e., transition t fires, decision tuples for places in the postcondition of t retain positive NES-status, and these decision tuples make their decision according to the structure of the strategy. In this case, we update the backward moves, i.e., we add $(\mathbb{D}_{pre(t)}, \mathbb{D}'_{post(t)})$ to BM_{id} to obtain BM'_{id} for participating player with identifier id. The remaining elements stay the same.

- If M_{T2} is reached for the second time, we perform the same step as for not reaching M_{T2} for the second time and update all decision tuples with positive NES-status to ended NES-status, i.e., for $\mathbb{D}''$ obtained from the previous step, $\mathbb{D}' = \{(id, p, b', T, K) \in \mathcal{D} \mid (id, p, b, T, K) \in \mathbb{D}'' \wedge (b = false \Rightarrow b' = false) \wedge (b = true \Rightarrow b' = end)\}$. We set M'_{T2} to the empty set and update the backward moves as in the definition of NES_{finish}. The remaining elements stay the same.

Lemma 2 (From Petri game strategies to Büchi game strategies)**.** *If the system players have a winning strategy in the Petri game, then there exists a winning strategy for Player 0 in the Büchi game.*

Proof. By making strategies for the system players in the Petri game minimal, this proof is analogous to the proof of Lemma 1. We translate a minimal winning strategy for the system players in the Petri game into a winning strategy for Player 0 in the Büchi game as defined in Definition 47. To prove that the constructed strategy is winning, we prove that the reachable cuts of every minimal winning strategy for the system players in the Petri game are equivalent as defined in Definition 40 to the considered decision markings in the corresponding strategy for Player 0 in the Büchi game. Then, losing behavior in the strategy for the system players in the Petri game occurs if and only if losing behavior in the strategy for Player 0 in the Büchi game occurs. We consider both directions for a minimal winning strategy σ for the system players in the Petri game and the corresponding strategy f for Player 0 in the Büchi game:

- We show that if a reachable cut C exists in σ, then a decision marking is considered in a by f reachable state of the Büchi game. The cut C is reached via the sequence of transitions T. We only retain transitions with an environment place in their precondition in T. The choices of system players are already represented by σ. We start from the initial marking of σ and obtain a sequence of transitions w that is applicable to f and reaches C by recursively applying the following steps: If a NES-case starts from the current cut (i.e., a minimal set of system players in the cut can play infinitely together without synchronizing with the environment player) and the underlying marking of the cut is the first marking to repeat in this NES-case, then we add the transitions until the repetition of the marking occurs to w and update the current cut accordingly. By Lemma 7, at most on NES-case can occur. Otherwise, we try to fire a transition with only system places in its precondition. If this is not possible, we try to fire a transition with an environment

places in its precondition. We pick the environment transition corresponding to the first element of T and afterward remove it from T. In both cases, the current cut becomes the cut reached after firing the transition.

When none of these cases apply and there is a transition left in T, we need to add transitions with only system places in their precondition which are rolled back by backward moves to reach an equivalent decision marking to the cut C. In this case, we fire a maximal sequence of transitions with only system places in their precondition such that only transitions with an environment place in the precondition are enabled. The enabled transitions include the remaining transition in T. These fired transitions and all following transitions with only system places in their precondition are collected to be used as backward moves. We apply the recursive steps from before again. The first iteration removes the last transition from T and afterward a finite number of transitions with only system places in their precondition are added. Notice that at most one transition can be left in T because for all but the last transition we fire it as late as possible at an mcut. This procedure obtains w'. We add steps to w' to make explicit decisions removing $\top$ according to the postcondition of the respective players and indicate the beginning of the NES-case by changing the corresponding decision sets from negative to positive NES-status to obtain w. By the construction of the Büchi game, w is applicable to the Büchi game. Either w directly leads to a decision marking $\mathbb{D}$ such that $\mathbb{D} \equiv C$ holds or some of the collected backward moves and of the backward moves in the NES-case are applied to the state in the Büchi game leading to a decision marking $\mathbb{D}'$ such that $\mathbb{D}' \equiv C$ holds.

- We show that if a decision marking $\mathbb{D}$ is considered in a by f reachable state of the Büchi game, then a reachable cut C exists in σ such that $\mathbb{D} \equiv C$ holds. If $\mathbb{D}$ is not considered via backward moves, then the word w exists to reach the state of $\mathbb{D}$. The word w^- is defined as w without symbols $\top$. Firing transitions in the order of w^- in the strategy leads to a cut C. By the strategy translation from Definition 47, w^- is applicable to σ and, by the construction of the Büchi game, the precondition and postcondition of the used transitions are the same in both strategies. Therefore, $\mathbb{D} \equiv C$ holds. If $\mathbb{D}$ is reached via backward moves, then the word w to reach the state of $\mathbb{D}$ and the word w^{back} of applied backward moves exist. We define the subtraction of w^{back} from w^- from the end of both words. Firing transitions in the order of $w^- - w^{back}$ in the strategy leads to a cut C. By the strategy translation from Definition 47, the backward moves can be subtracted from the word, $w^- - w^{back}$ is applicable to σ, and, by the construction of the Büchi game, the precondition and postcondition of the used transitions are the same in both strategies. Therefore, $\mathbb{D} \equiv C$ holds.

The same argument as in the proof of Lemma 1 applies to show that it suffices to only apply backward moves from one state, i.e., at most one transition can be left in T.

The constructed strategy for the Büchi game is a strategy as it chooses one successor for every state of Player 0. Also, by construction, all successors are added to states of Player 1.

As we showed that a strategy for the system players in a Petri game and the corresponding strategy for Player 0 in a Büchi game visit equivalent cuts and decision markings, we can conclude that for a minimal winning strategy for the system players in the Petri game an accepting state in the strategy for Player 0 in a Büchi game is visited infinitely often. This is because F_B is visited infinitely often for finite plays and for infinite plays at most one NES-case occurs needing only finitely many states and a state representing an mcut is visited infinitely often as the environment player is part of fired transitions infinitely often. Therefore, the strategy for the system players in the Petri game is winning. □

Theorem 3 (Game solving)**.** *For Petri games with a bounded number of system players, one environment player, and bad markings, the question of whether the system players have a winning strategy is decidable in* 2-EXPTIME*. If a winning strategy for the system players exists, it can be constructed in double exponential time.*

Proof. We establish the complexity by estimating the size of the set V of states in the Büchi game. We recall from Definition 19 that the states V in the Büchi game are defined as $V = V_{BN} \cup \mathscr{D} \times (\mathcal{P}_S \to \{0,\dots,k\}) \times (\mathcal{B}^*)^{\max_S} \times \{1,\dots,\max_S\}$. The underlying Petri net is k-bounded by some $k \geq 1$. We use binary encodings to identify places. This results in $\log_2$ occurring in the size estimate. Here, we assume that $\log_2$ returns numbers greater or equal to one, i.e., $\log_2(x) = \max(1, log_2^{usual}(x))$ where $\log_2^{usual}(x)$ is the usual logarithm with base two. We represent a decision tuple (id, p, b, T, K) by $\lceil \log_2(|\max_S|) \rceil$ Boolean variables to identify the identifier id, $\lceil \log_2(|\mathcal{P}|) \rceil$ Boolean variables to identify the place p, two Boolean variables for the three-valued flag b, a Boolean variable indicating the presence of $\top$, and, for each transition $t \in \mathcal{T}$, a Boolean variable indicating the presence of t in the decision T, and $\lceil \log_2(|\max_S|) \rceil$ Boolean variables to identify the last mcut K. There are at most $\max_S + 1$ players. The additional player is the environment player. The maximal number of system players $\max_S$ includes the possibility of k players in one place. This gives us $(\max_S + 1) \cdot (\lceil \log_2(|\mathcal{P}|) \rceil + 3 + |\mathcal{T}| + 2 * \lceil \log_2(|\max_S|) \rceil)$ Boolean variables. Considering all valuations of these variables, the size of the set $\mathscr{D}$ of decision markings can be bounded by $A = 2^{(\max_S+1) \cdot (\lceil \log_2(|\mathcal{P}|) \rceil + 3 + |\mathcal{T}| + 2 * \lceil \log_2(|\max_S|) \rceil)}$.

The marking to repeat in the NES-case can be represented with $\max_S \cdot \lceil \log_2(|\mathcal{P}_S|) \rceil$ Boolean variables. Thus, the size of the set of markings to repeat in the NES-case can be bounded by $B = 2^{\max_S \cdot \lceil \log_2(|\mathcal{P}_S|) \rceil}$.

Backward moves are pairs of decision markings. We need $2 \cdot \max_S \cdot (\lceil \log_2(|\mathcal{P}_S|) \rceil + 3 + |\mathcal{T}|)$ Boolean variables to represent a backward move. The number of backward moves for each of the $\max_S$ sequences can be bound by the triangular number of the maximum number of system players $\max_S \cdot (\max_S + 1)/2$ times the number of markings $2^{|\mathcal{P}|}$. The maximal length of a loop can be bounded by the number of markings $2^{|\mathcal{P}|}$ and in each loop at least one system player learns about at least one new last mcut. After $(\max_S \cdot (\max_S + 1)/2) \cdot |\mathcal{T}|$ loops, all system players have to be maximally informed and

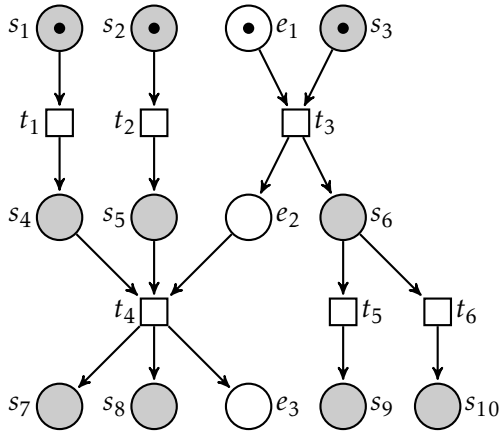


Figure 3.7.: As $\{s_1, s_5, e_2, s_9\}$ and $\{s_2, s_4, e_2, s_{10}\}$ are bad markings, no deadlock-avoiding strategy for the system players can avoid the bad markings.

further loops lead to a useless repetition, i.e., the collection of backward moves can be stopped. Therefore, the number of backward moves in the sets of backward moves can be bounded by $C = 2^{(\max_S \cdot (\max_S+1)/2) \cdot 2^{|\mathcal{P}|} \cdot 2 \cdot \max_S \cdot (\lceil \log_2(|\mathcal{P}_S|) \rceil + 3 + |\mathcal{T}|)}$.

The size of the set V of states in the Büchi game can be bounded by $A \cdot B \cdot C$. This equals a double exponential number of states in the size of the Petri game dominated by the factor C. As Büchi games can be solved in polynomial time [CH12], the total time required to construct and solve the Büchi game is double exponential in the size of the Petri game.

Büchi games are memoryless determined. The winning strategy can therefore be represented by a finite graph G_f whose size is bounded by the size of the Büchi game. We construct a finite deadlock-avoiding winning strategy for the system players in the Petri game following the construction from Definition 39, using G_f instead of the infinite strategy tree T_f. In the NES-case, we represent the strategy for the system players finitely. When the marking to repeat in the NES-case is reached for the second time, we do not add an infinite unrolling of the strategy for the system players in the NES-case. Instead, we change the flow between transitions and places where the infinite unrolling would be added to the first occurrence of the place in the NES-case. We remove all thereby unreachable places and represent the NES-case with a finite strategy. When G_f reaches a state for the second time, i.e., a loop occurs with the environment player, we change the flow from transitions to places, where the strategy would continue as above, such that they reach the cut corresponding to the first occurrence of the state in the Büchi game. We remove all thereby unreachable places and represent a loop with the environment player with a finite strategy. □

3.4.10. Necessity of Backward Moves

We illustrate the necessity of backward moves in our reduction with an exemplary Petri game. For this exemplary Petri game in Figure 3.7, the Büchi game encoding it is won by Player 1 because the Petri game does not have a winning strategy for the system players. To show that the bad markings cannot be avoided by a strategy for the system players while being deadlock-avoiding, the use of backward moves is essential: Both transitions t_1 and t_2 fire before transition t_3 fires in the Büchi game encoding the Petri game in Figure 3.7 because t_1 and t_2 have only system places in their precondition. Only after t_3 fires, Player 0 can decide between transitions t_5 and t_6. To reach one of the bad markings $\{s_1, s_5, e_2, s_9\}$ and $\{s_2, s_4, e_2, s_{10}\}$, it is required that only one of the transitions t_1 and t_2 fired when deciding between t_5 and t_6. Backward moves allow from either reachable marking $\{s_4, s_5, e_2, s_9\}$ or $\{s_4, s_5, e_2, s_{10}\}$ to apply the backward move for t_1 or t_2 to prove that the bad markings cannot be avoided.

3.5. Requiring Deterministic Decisions for Strategies

We elaborate on the requirement of deterministic decisions for strategies for the system players in Petri games. The reduction in the case of bad places as local winning conditions from the original paper on Petri games [FO17] is not quite correct for a very small class of Petri games[1]. We explain this problem in detail and give two solutions: First, we slightly restrict the class of Petri games that can be solved by the original paper. Second, we show that backward moves from the reduction for bad markings as global winning condition presented in this chapter can be used to extend the original paper to exactly decide the existence of winning (and therefore deterministic) strategies.

In Figure 3.8, a Petri game is depicted that has no winning strategy but for which the reduction in the original paper falsely gives a strategy. The Petri game can be considered with an empty set of bad places and with an empty set of bad markings. In both cases, there does not exist a winning strategy for the system players in the Petri game. The two markings $\{s_2, s_4, e_5\}$ (reachable by transitions t_1, t_3, and then t_5 firing) and $\{s_1, s_4, e_4\}$ (reachable by transition t_2 firing) are reachable. Thus, Player 0 has to allow for the system player in place s_4 both transitions t_6 and t_7 in order to avoid deadlocks.

This decision is nondeterministic for the marking $\{s_2, s_4, e_4\}$ (reachable by transitions t_1, t_3, and t_4 firing). When following the reduction in the original paper, transition t_6 always fires before transition t_5 and the marking $\{s_2, s_4, e_4\}$ is not reached. This results in the strategy in Figure 3.9 which violates justified refusal at system place s_4. The definition of justified refusal implies that either both transitions t_7 and t_7' are allowed or both are disallowed. Allowing both leads to a nondeterministic decision and not allowing both leads to a deadlock. Therefore, no winning strategy for the system players can exist.

[1] We thank Paul Gölz for alluding us to this problem.

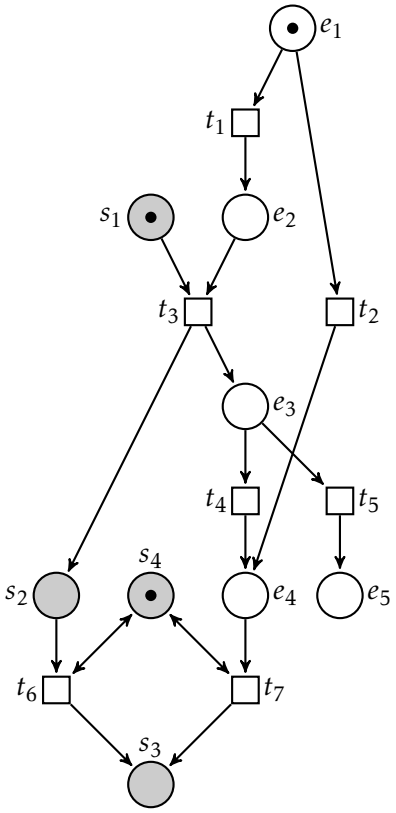


Figure 3.8.: A Petri game without bad places is depicted. The three markings $\{s_2, s_4, e_5\}$ (by t_1, t_3, and then t_4 firing), $\{s_1, s_4, e_4\}$ (by t_2 firing), and $\{s_2, s_4, e_4\}$ (by t_1, t_3, and then t_5 firing) are reachable. This implies that no deterministic and deadlock-avoiding strategy for the system player in place s_4 can exist.

Structural Restriction

As a first possible solution, we maintain the original definition of deterministic decisions for system players on page 188 of [FO17], but restrict the class of games for which Section 6 on pages 191 ff. is applicable. We add to the very last part on page 192 that $\mathcal{G}$ has to fulfill the following condition: $\forall p \in \mathcal{P}_E : \neg(p \sharp p) \vee \forall t \in pre(p) : M \in \mathcal{R}(M) : pre(t) \subseteq M \Rightarrow pre(t) = M$. It has to hold that every environment place is not in self-conflict, i.e., the environment place is not reachable from any other place by exiting the other place with two different arcs, or all ingoing transitions of the environment place are only enabled from a reachable marking when the marking is equal to the precondition of the transition. Note that the conditions are evaluated on the original Petri game and not on the unfolding or on strategies. In the proof of Lemma 6.6 on page 196, the argument for requirement (S3) now holds because skipped environment transitions due to preference of system transitions have to be represented explicitly and therefore can

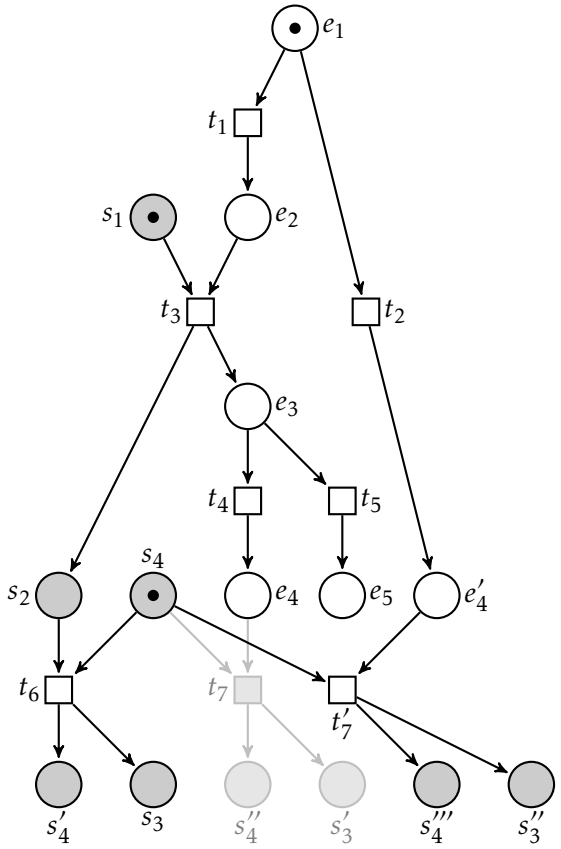


Figure 3.9.: The unfolding and a strategy for the Petri game in Figure 3.8 are depicted. The unfolding includes the grayed-out parts, whereas the strategy does not include them. Copies of places and transitions due to splits from the unfolding are primed. The strategy violates justified refusal because it disallows t_7 while allowing t'_7.

be forbidden. In this thesis, requirement (S3) for strategies for the system players is included in the requirement of justified refusal for strategies for the system players.

Extended Reduction

As a second possible solution, we explain how *backward moves* from the reduction introduced in this chapter solve the problem concerning deterministic decisions described before. The reduction from this chapter solves the synthesis problem for Petri games with a bounded number of system players and one environment player for the global winning condition of bad markings. This global winning condition is an extension compared with the local winning condition of bad places from the original paper on Petri games [FO17]. Using backward moves as described in Section 3.3.4 for the Petri game in Figure 3.8, a backward move for transition t_6 is applicable from the state encoding the marking $\{s_3, s_4, e_4\}$, resulting in the previous state corresponding to the marking

$\{s_2, s_4, e_4\}$. In this state, the made decisions of the players make the nondeterministic decision of system player s_4 visible. The system player s_4 allows both its outgoing transitions t_6 and t_7. Therefore, reaching the original state encoding the marking $\{s_3, s_4, e_4\}$ is losing for Player 0, and correctly no winning strategy for Player 0 exists in the Büchi encoding the Petri game.

Backward moves can be added to the original reduction for bad places as winning condition in order to find nondeterministic decisions of the system players. They are not necessary in the NES-case that is handled before generating the Büchi game. The higher complexity due to the length of the stored sequences of backward moves transfers to the extended reduction for bad places as local winning condition and for always preventing nondeterministic decisions. We conjecture that the construction can be optimized in such a way that its complexity is in EXPSPACE (instead of 2-EXPTIME) by guessing and checking the necessary sequence of backward moves instead of collecting all backward moves. It is an intriguing open problem whether this construction is really possible and whether it is optimal in terms of complexity. It is also a fascinating question whether there is a necessary gap in terms of complexity between bad places as local winning condition and bad markings as global winning condition or whether the somewhat global requirement of deterministic decisions prevents such a gap. We conjecture that the guessing and checking of backward moves is, in fact, necessary and optimal in terms of complexity for both the local winning condition of bad places and the global winning condition of bad markings.

3.6. From At Most One to Exactly One Environment Player

We show that every Petri game with at most one environment player can be extended into a Petri game with exactly one environment player such that both Petri games have equivalent strategies. The strategies are equivalent in the sense that the set of strategies obtained by removing the additions of our construction from the strategies for the extended Petri game is equal to the set of strategies of the original Petri game. The extension widens the decidability result for Petri games with a bounded number of system players, one environment player, and bad markings as global winning condition from the previous section to Petri games with a bounded number of system players, *at most one* environment player, and bad markings as global winning condition.

The construction works as follows: Given a Petri game $\mathcal{G}$ with at most one environment player in every reachable marking, we add a unique place env_{idle} to the set of environment places $\mathcal{P}_E$ of $\mathcal{G}$. When the initial marking In of $\mathcal{G}$ does not place a token into an environment place, then we extend In in such a way that it puts one token into the unique environment place env_{idle}. Otherwise, the unique environment place env_{idle} remains empty initially. For every transition t of $\mathcal{G}$ that does not require a token in an environment place by its precondition but puts a token into an environment place according to its postcondition, we extend the precondition of t in such a way that it also requires one token in the unique environment place env_{idle}. The postcondition of t remains unchanged. Analogously, for every transition t of $\mathcal{G}$ that does not put a token

in an environment place according to its postcondition but requires a token in an environment place by its precondition, we extend the postcondition of t in such a way that it also puts one token in the unique environment place env_{idle}. In this case, the precondition of t remains unchanged. When the winning condition is concerned with markings (i.e., it is a set of bad markings, a set of good markings, or a pair of good and bad markings), then we extend every marking that does not require a token in an environment place in such a way that it requires a token in the unique environment place env_{idle}.

By these steps, we obtain a Petri game with exactly one environment player in all reachable markings. When initially there was no environment player before, then it will now be in the unique environment place env_{idle}. Every transition, spawning the one environment player before, now moves the token out of the unique environment place env_{idle}. Every transition, terminating the one environment player before, now moves the token into the unique environment place env_{idle}. Maintaining one environment player leads to a larger causal past being accumulated and being shared upon synchronization with system players. The accumulation is no problem because the strategy for the system players has to work against all behaviors of the environment player, even against those that behave as if it accumulated the larger causal past. Sharing the larger causal past with some system players is no problem, either, because there has to be at least one system player that also shares it when ensuring the sequentiality between terminating and spawning the environment player. Otherwise, there could be more than one environment player.

3.7. Summary

We proved that the existence of a winning strategy for the system players is decidable for Petri games with a bounded number of system players, at most one environment player, and bad markings as global winning condition. Before, such a result was only known for the local winning condition of bad places [FO17]. Global winning conditions are a significant improvement over local winning conditions because they allow expressing global properties like mutual exclusion. The decidability result for bad markings is based on a reduction to Büchi games as in the decidability result for bad places. In both cases, system players explicitly fix their decisions of which of their outgoing transitions to allow as early as possible while decisions by the environment player occur as late possible at mcuts. The Büchi winning condition requires that at least one state corresponding to an mcut is reached infinitely often. We make two major extensions to encode bad markings as global winning condition. These extensions also avoid an intricate problem of the original decidability result for bad places as local winning condition.

The first extension deals with the case of some system players playing infinitely without synchronizing with the environment player. This so-called NES-case is handled directly in the Büchi game (instead of in a preprocessing step as in the decidability result for bad places). Handling it directly in the Büchi game is necessary to check markings

that include players, that play infinitely without the environment player, and players, that synchronize infinitely often with the environment player. The NES-case is handled directly in the Büchi game by allowing (and requiring) the system players to announce that they want to play infinitely without synchronizing with the environment player. The can prove their announcement by repeating a marking, after which the usual order of the Büchi game of repeating mcuts resumes. The system players are required to announce the NES-case because otherwise they cannot reach a Büchi state corresponding to an mcut infinitely often, which is losing for Player 0 in the Büchi game encoding the system players in the Petri game.

The second extension checks for each sequential play in the Büchi game that no bad markings and no nondeterministic decisions occur for all possible markings due to the reordering of concurrent transitions in the encoded play of the Petri game. This is achieved by backward moves which store the local past of each system player until its last synchronization with the environment player. We prove that this history can be stored finitely and suffices to check all reachable markings both to not be a bad marking and to not encode a nondeterministic decision. Such nondeterministic decisions that are only visible at markings reachable when reordering concurrent transitions are the intricate problem of the decidability result for bad places. This can be solved by extending the reduction in the case of bad places with backward moves.

Chapter 4

Undecidability of Good Markings

In this chapter, we investigate global winning conditions beyond bad markings. We report two undecidability results to motivate our focus on the global winning condition of bad markings in the previous chapter and to show that undecidability is quickly reached after the global winning condition of bad markings. We first recall the undecidability of synthesis of distributed systems in the synchronous setting and then explain on an intuitive level how these ideas can be transferred to the asynchronous setting of Petri games. Here, incomplete information between two system players allows us to force them to simulate the halting problem of a Turing machine. For the asynchronous setting of Petri games, we show how good markings can define firing sequences that deviate too much from the synchronous setting as winning. Thereby, only equivalent cases to the synchronous setting remain. For the formal reduction, we use the Post correspondence problem instead of the halting problem of Turing machines only for simpler notation. By the formal reduction, we obtain our first undecidability result: We prove that it is undecidable whether a winning strategy exists for the system players in Petri games with at least two system players, at least one environment player, and good and bad markings as global winning condition.

Afterward, we prove that bad markings from this undecidability result can be represented by having all players repeatedly become environment players. Bad markings and the increase in the number of environment players is used to detect errors by one or both system players in the solution to the PCP. Thereby, we obtain our second undecidability result: We prove that it is undecidable whether a winning strategy exists for the system players in Petri games with at least three players, out of which one player is always an environment player and two players can change between being a system and environment player and good markings as global winning condition. In the end, a transfer of our undecidability proofs to control games based on asynchronous automata is outlined. Because of the close relationship between Petri games and control games

discussed in Section 1.7.2, the key ideas from our undecidability proofs can be applied to control games and we achieve similar undecidability results for control games.

The key contributions of this chapter are the following:

- We prove that it is undecidable whether a winning strategy exists for the system players in Petri games with at least two system players, at least one environment player, and *good and bad markings* as global winning condition.
- We prove that it is undecidable whether a winning strategy exists for the system players in Petri games with at least three players, out of which one player is always an environment player and two players can change between being a system and environment player and *good markings* as global winning condition.
- We outline how these undecidability proofs can be transferred to *control games* based on asynchronous automata. Theses results show that one quickly hits undecidability when using good markings as winning condition.

This chapter is structured as follows: In Section 4.1, we present the undecidability proof for the synthesis of distributed systems in the synchronous setting that is the main inspiration for our undecidability proofs. In Section 4.2, we outline an in-depth overview of our undecidability proofs and show how the synchronous setting can be simulated in the asynchronous setting of Petri games. In Section 4.3, we give our undecidability proofs in their full formality. In Section 4.4, we outline how our undecidability proofs can be transferred to control games based on asynchronous automata.

4.1. Undecidability in the Synchronous Setting

Synthesis of distributed systems in the synchronous setting of Pnueli and Rosner is undecidable [PR90; Sch14]. In the proof, the specification forces two synchronous system players P_0 and P_1 with incomplete information on the environment *Env* (cf. Figure 4.1) to simulate a Turing machine by outputting sequences of tape configurations and to eventually reach a halting tape configuration. The environment can change the distance between the simulation at the two system players and can check the synchronously output tape configurations for correctness. Notice that the two system players cannot delay their output as they could in the asynchronous setting. Each system player has no information about the distance to the simulation of the other one. Thus, simulating the Turing machine at both players is the only possible winning strategy and synthesis of synchronous distributed systems can decide whether this strategy reaches a halting tape configuration. The specification of the two players can be encoded in the temporal logic LTL [Sch14].

In the next section, we show how good markings can restrict the asynchronous setting of Petri games to an equivalent a synchronous setting as in the setting of Pnueli and Rosner. Then, we encode the Post correspondence problem (PCP) into Petri games with good markings to prove their undecidability. We use the PCP instead of simulating a Turing machine in order to obtain simpler notation.

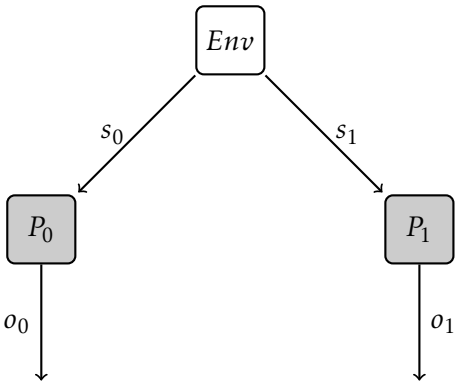


Figure 4.1.: The architecture of the undecidability proof for synthesis of synchronous distributed systems is depicted. There is one environment player *Env* and two system players P_0 and P_1. System player P_0 can base its output o_0 only on input s_0 from the environment player whereas system player P_1 can base its output o_1 only on input s_1 from the environment player. This implies that both system players have incomplete information about the other system player.

4.2. Undecidability of Petri Games with Good Markings

We prove that it is undecidable whether a winning strategy exists for the system players in Petri games with at least two system and one environment player and good and bad markings. For this winning condition, no bad marking should be reached *until* a good marking is reached, which can be expressed in LTL. Notice that good markings do not require players to terminate in a good marking but only to reach one. Notice also that, after a good marking has been reached, it is allowed to reach a bad marking. We show how good markings can enforce the undecidable synchronous setting [PR90; Sch14] from the previous section in Petri games. Afterward, we prove that it is undecidable whether a winning strategy exists for the system players in Petri games with only good markings and at least three players, out of which one is an environment player and each of the other two changes between system and environment player. Bad markings from the previous result are encoded by system players repeatedly changing to environment players and back. All formal details can be found in Section 4.3.

4.2.1. Petri Game for the Post Correspondence Problem

The undecidability proof uses the Post correspondence problem [Pos46]. The *Post correspondence problem* (PCP) is to determine, for a finite alphabet Σ and two finite lists $r_0, r_1, \ldots, r_n$ and $v_0, v_1, \ldots, v_n$ of non-empty words over Σ, if there exists a non-empty sequence $i_1, i_2, \ldots, i_l \in \{0, 1, \ldots, n\}$ such that $r_{i_1} r_{i_2} \ldots r_{i_l} = v_{i_1} v_{i_2} \ldots v_{i_l}$. This problem is undecidable (at least for five or more pairs of words) which is proven by a reduction from the halting problem of Turing machines [Nea15].

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

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

4.2.2. Linear Firing Sequences via Good Markings

We use MOD-3 counters to restrict the asynchronous setting of Petri games to firing sequences equivalent to the synchronous setting of Pnueli and Rosner [PR90; Sch14]. For each system player, we introduce two *MOD-3 counters* to count the number of output indices and of output letters modulo three. Whenever a player outputs an index, the respective index counter is increased by one, and accordingly for output letters and the letter counter. If a counter would reach value three, it is reset to zero. We define good markings based on the two MOD-3 index counters and the two MOD-3 letter counters. In a *linear firing sequence for indices (letters)*, the two system players output the indices (letters) alternately with the first system player preceding the second one at each turn. With good markings, we ensure that the environment player first decides that either the output indices or letters are checked for equality. Afterward, a good marking is reached when a firing sequence is not a linear firing sequence for indices or letters, based on the decision by the environment player.

In Figure 4.2, we visualize the reachability graph for the two system players when only considering either the values of their MOD-3 counters for indices or letters. Markings are differentiated in the reachability graph depending on if a good marking is

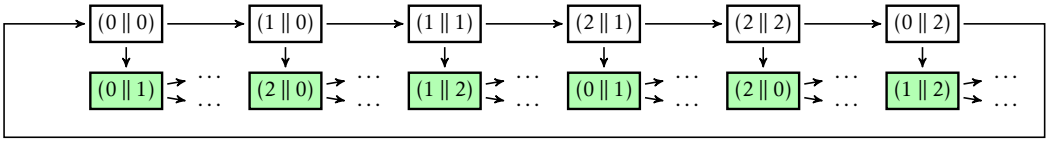


Figure 4.2.: The reachability graph for the two system players is depicted when only considering either the values of their MOD-3 index counters or of their MOD-3 letter counters and differentiating markings depending on if a good marking is reached before. Good markings are colored green. All behavior after a good marking (including reaching a bad marking) is winning by definition. To compare output indices or letters, only the specific firing sequence in white has to be considered.

reached before, e.g., position $(0 \parallel 1)$ does not lead to position $(1 \parallel 1)$ as the path to $(1 \parallel 1)$ does not include a good marking. With linear firing sequences, we only consider firing sequences where the first system player outputs the first index or letter before the second system player as the opposite cases are good markings. For firing sequences not reaching a good marking, equality of output indices or letters is checked at positions $(0 \parallel 0)$, $(1 \parallel 1)$, and $(2 \parallel 2)$. Thereby, equality of output indices or letters at the same position can be checked without storing all outputs and it is ensured that solutions have the same length.

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

4.2.3. Preventing Untruthful Termination

The good markings to only consider linear firing sequences introduce new possibilities for the system players to be winning. These possibilities arise when the system players can enforce all firing sequences to reach a good marking. They occur when a system player terminates without the end symbol ($\#_1$ or $\#_2$) and are called *untruthful termination*. Untruthful termination is prevented by letting the environment player decide which system player it believes to not terminate with the end symbol or that everything is okay. This decision happens together with the initial choice of the environment player between checking equality of indices or letters. Due to the independence of the players, each system player has to behave as if the environment player is anticipating it to untruthfully terminate and has to output the end symbol to avoid this. Therefore, no untruthful termination can occur.

4.2.4. Undecidability Results

A winning strategy exists in the Petri game if and only if there exists a solution to the instance of the PCP. The only strategy with a chance to be winning for the two system players is to output the same solution to the PCP and we can translate solutions between both cases. Therefore, we can encode the PCP in Petri games with good markings and bad markings which shows that the synthesis problem for such Petri games is undecidable. This gives us:

Theorem 8 (One environment player and good and bad markings)**.** *For Petri games with at least two system players, one environment player, and good and bad markings as global winning condition, it is undecidable whether the system players have a winning strategy.*

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

Theorem 9 (Three environment players and good markings)**.** *For Petri games with good markings as global winning condition and at least three players, out of which one is an environment player and two change between being a system player and an environment player, it is undecidable whether the system players have a winning strategy.*

4.3. Formal Details

In this section, we give the formal reduction from the Post correspondence problem to Petri games with good and bad markings and from Petri games with good and bad markings to Petri games with good markings. First, we present the first reduction by formally defining the two system players, the environment player, the good markings, and the bad markings. Note that we define a Petri game based on a safe Petri net which allows us to use sets of places instead of multisets over places for simpler notation. Second, we prove the first reduction correct. Third, we present the second reduction by formally defining the translation of Petri games with both good and bad markings to Petri games with good markings. Fourth, we prove the second reduction correct.

Throughout the section, we use unique variables for parts of our reduction. The number of indices is n and i is used to iterate over the possible indices. To differentiate between the two system player, we use $k \in \{1, 2\}$. The word of an index is identified with w_i for both system players, with r_i specifically for the first system player and with v_i specifically for the second system player. We use j to iterate over words letter-by-letter. Labels of transitions and letters are identified by l. We use x to either identify an index counter or its value for both system players, y specifically for an index counter or its value for the first system player, and z specifically for an index counter or its value for the second system player. The same applies to the letter counter with a, b, and c.

4.3.1. First Reduction

We present the reduction from the Post correspondence problem to Petri games with good and bad markings.

The Two System Players

The two system players without their MOD-3 counters are presented on the left side of Figure 4.3. We display transitions with their label which is equal to the transition when the label is unique. This means, for example, that τ or the letters to represent words w_i are only labels because they occur more than once, whereas $\#_k$ is both the transition and its label because it occurs only once in each system player. Let $k \in \{1,2\}$ differentiate between the first system player and the second system player and w identify the words of the players (i.e., $w = r$ if $k = 1$ and $w = v$ if $k = 2$). For each of the two system players, the system place p^k_{start} with the token initially has a choice between the indices $0,\dots,n$. Afterward, the corresponding word $w_0,\dots,w_n$ is output letter-by-letter, finished by a transition labeled by τ. This is illustrated on the right side of Figure 4.3. In all cases, the system place p^k_{choice} is reached which presents the choices with the same label to the system player as from system place p^k_{start} with the additional option to terminate with transition $\#_k$.

The player has $4 + n + \sum_{i\in\{0,\dots,n\}} |w_i|$ places. There are two places (p^k_{start} and p^k_{choice}) to decide between the indices, only the latter including the option $\#_k$, the termination place p^k_{term}, and $n+1$ places $p^k_0,\dots,p^k_n$ for the chosen index from $0,\dots,n$. After each index, the sequence of letters w_i is followed by a transition labeled by τ returning to the decision place including the option $\#_k$. Notice that the decisions for indices are labeled with numbers $0,\dots,n$ and transitions for outputting words are labeled with letters from Σ. The remaining transitions are either $\#_k$ or labeled by τ.

Next, we add the index counter $co^k_{index} = \{0^k_{index}, 1^k_{index}, 2^k_{index}\}$ and letter counter $co^k_{letter} = \{0^k_{letter}, 1^k_{letter}, 2^k_{letter}\}$. Places have the form $ID^k \times co^k_{index} \times co^k_{letter}$ where ID^k are the places from Figure 4.3. Transitions labeled with numbers increase the MOD-3 counter for indices by one and transitions labeled with letters increase the MOD-3 counter for letters by one. The termination transition $\#_k$ and transitions labeled by τ leave the counters unchanged and initially the counters are set to zero.

For $k \in \{1,2\}$, the Petri games for each of the two system players has the form $\mathcal{G}^k_S = (\mathcal{P}^k_S, \emptyset, \mathcal{T}^k, \mathcal{F}^k, In^k, (\emptyset,\emptyset))$ with $\mathcal{P}^k_S$, $\mathcal{T}^k$, $\mathcal{F}^k$, and In^k defined as follows. For words w_i of length at least one, we introduce the notation $w_i[j]$ for $0 \le j \le |w_i| - 1$ to obtain the letters of the word. A place $p^k_{i:j:l}$ with $0 \le i \le n$ and $0 \le j \le |w_i| - 1$ is reached after the j-th letter $l \in \Sigma$ of word w_i has been output. We define the IDs of places as $ID^k = \{p^k_{start}, p^k_{choice}, p^k_{term}, p^k_0, \dots, p^k_n\} \cup \{p^k_{i:j:w_i[j]} \mid 0 \le i \le n \wedge 0 \le j \le |w_i| - 1\}$ and the set of system places as $\mathcal{P}^k_S = \{(p^k_{start}, 0^k_{index}, 0^k_{letter})\} \cup \{(id^k, x^k, a^k) \mid id^k \in ID^k \setminus \{p^k_{start}\} \wedge x^k \in co^k_{index} \wedge a^k \in co^k_{letter}\}$. The initial marking In^k is $\{(p^k_{start}, 0^k_{index}, 0^k_{letter})\}$.

We introduce the set of *labels* $\mathscr{L}$ for transitions as $\mathscr{L} = \{\#_k, \tau\} \cup \{0,\dots,n\} \cup \Sigma$ and the notation $p\ [l\rangle^k\ p'$ to define a unique transition with label $l \in \mathscr{L}$ from place p to place p'

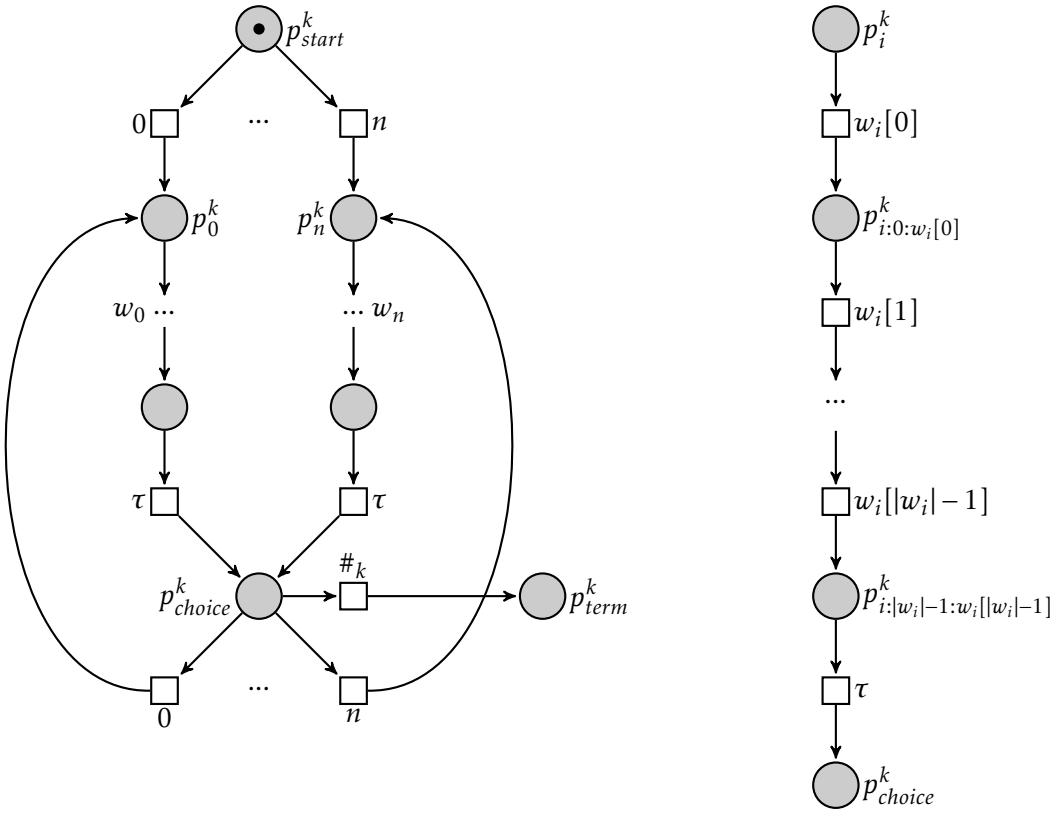


Figure 4.3.: Let $k = 1$ identify the first system player with $w = r$ and $k = 2$ the second one with $w = v$. On the left, each of the two system players is shown without MOD-3 counters. It is abbreviated how words $w_0, \ldots, w_n$ are output letter-by-letter. On the right, this abbreviation between places p^k_i for $i \in \{0, \ldots, n\}$ and p^k_{choice} is made explicit.

as transition $t^k_{p[l\rangle p'} \in \mathcal{T}^k$ with the corresponding arcs $(p, t^k_{p[l\rangle p'}), (t^k_{p[l\rangle p'}, p') \in \mathcal{F}^k$. In the following, we use this notation to define the transitions $\mathcal{T}^k$ and the flow $\mathcal{F}^k$ step-by-step. The outgoing transitions of the initial place and their flow are defined as

$$\{(p^k_{start}, 0^k_{index}, 0^k_{letter})[i\rangle^k(p^k_i, 1^k_{index}, 0^k_{letter}) \mid 0 \leq i \leq n\}.$$

The outgoing transitions of the choice place and their flow are defined for outputting an index as

$$\{(p^k_{choice}, x^k_{index}, a^k_{letter})[i\rangle^k(p^k_i, ((x+1) \bmod 3)^k_{index}, a^k_{letter}) \mid 0 \leq i \leq n \wedge 0 \leq x, a \leq 2\}$$

and for terminating with the special symbol as

$$\{(p^k_{choice}, x^k_{index}, a^k_{letter})[\#_k\rangle^k(p^k_{term}, x^k_{index}, a^k_{letter}) \mid 0 \leq x, a \leq 2\}.$$

The letter-by-letter output of words is outlined on the right in Figure 4.3. We define the letter-by-letter output of words in the following.

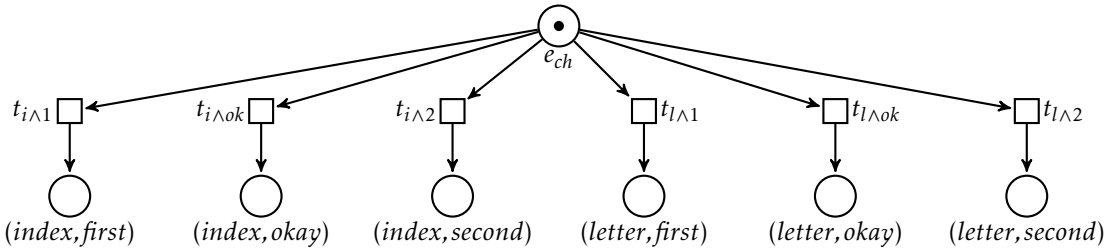


Figure 4.4.: The possible decisions of the one environment player are depicted. It decides at the same time which output to check and whether or not it anticipates untruthful termination by one of the two system players.

The transitions for the beginning of words are defined as

$$\{(p_i^k, x_{index}^k, a_{letter}^k)[w_i[0]\rangle^k(p_{i:0:w_i[0]}^k, x_{index}^k, ((a+1) \bmod 3)_{letter}^k) \mid 0 \leq i \leq n \wedge |w_i| > 0 \wedge 0 \leq x, a \leq 2\}.$$

The transitions for the remainder of words are defined as

$$\{(p_{i:j-1:w_i[j-1]}^k, x_{index}^k, a_{letter}^k)[w_i[j]\rangle^k(p_{i:j:w_i[j]}^k, x_{index}^k, ((a+1) \bmod 3)_{letter}^k) \mid 0 \leq i \leq n \wedge |w_i| > 0 \wedge 0 \leq x, a \leq 2 \wedge 1 \leq j \leq |w_i| - 1\}.$$

We add transitions labeled by τ to return to place p_{choice}^k after words have been output completely by

$$\{(p_{i:|w_i|-1:w_i[|w_i|-1]}^k, x_{index}^k, a_{letter}^k)[\tau\rangle^k(p_{choice}^k, x_{index}^k, a_{letter}^k) \mid 0 \leq i \leq n \wedge |w_i| > 0 \wedge 0 \leq j, m \leq 2\}.$$

The One Environment Player

The environment player is depicted in Figure 4.4 and makes two decisions in one step: It decides whether it wishes to check the output indices ($t_{i\wedge *}$) or letters ($t_{l\wedge *}$). Furthermore, the environment player decides whether it suspects the first ($t_{*\wedge 1}$) or second ($t_{*\wedge 2}$) system player to terminate untruthfully or whether the termination of the system players is fine ($t_{*\wedge okay}$). The result is stored in six pairs of the form $\{index, letter\} \times \{first, okay, second\}$. As all places and transition are unique in Figure 4.4, this formally defines the Petri game $\mathcal{G}_E$ for the environment player.

Good Markings and Bad Markings

The good and bad markings cover the two system players and the environment player. They are implicitly ordered as the first system player, second system player, and then the environment player. We use the notation $*$ to indicate that all places of the player

at the respective position are part of the good or bad markings. This notation can be applied to one element of the pair representing the decision by the environment player meaning that only the result of the other decision is important. We ensure that the environment player makes its decision first and therefore can ignore the place e_{ch} for $*$.

We start by defining good markings which indicate that both system players terminated with solutions of the same length (modulo three) as

$$\mathcal{M}_G^{finish} = \{\{(p^1_{term}, x^1_{index}, a^1_{letter}), (p^2_{term}, x^2_{index}, a^2_{letter}), (*, okay)\} \mid 0 \leq x, a \leq 2\}.$$

We start by defining good markings which indicate that both system players terminated with solutions of the same length (modulo three) as

$$\mathcal{M}_G^{finish} = \{\{(p^1_{term}, x^1_{index}, a^1_{letter}), (p^2_{term}, x^2_{index}, a^2_{letter}), (*, okay)\} \mid 0 \leq x, a \leq 2\}.$$

Next, we define good markings to focus on linear firing sequences for indices as

$$\begin{aligned}\mathcal{M}_G^{index} = \{&\{(p^1_{i_1}, y^1_{index}, b^1_{letter}), (p^2, z^2_{index}, c^2_{letter}), (index, *)\},\\ &\{(p^1, y^1_{index}, b^1_{letter}), (p^2_{i_2}, z^2_{index}, c^2_{letter}), (index, *)\} \mid\\ &p^1 \in ID^1 \setminus \{p^1_{term}\} \wedge p^2 \in ID^2 \setminus \{p^2_{term}\} \wedge 0 \leq i_1, i_2 \leq n \wedge\\ &(y \parallel z) \in \{(0 \parallel 1), (2 \parallel 0), (1 \parallel 2)\} \wedge 0 \leq b, c \leq 2\}\end{aligned}$$

and for letters as

$$\begin{aligned}\mathcal{M}_G^{letter} = \{&\{(p^1_{i_1:j_1:w_{i_1}[j_1]}, y^1_{index}, b^1_{letter}), (p^2, z^2_{index}, c^2_{letter}), (letter, *)\},\\ &\{(p^1, y^1_{index}, b^1_{letter}), (p^2_{i_2:j_2:w_{i_2}[j_2]}, z^2_{index}, c^2_{letter}), (letter, *)\} \mid\\ &p^1 \in ID^1 \setminus \{p^1_{term}\} \wedge p^2 \in ID^2 \setminus \{p^2_{term}\} \wedge\\ &0 \leq i_1, i_2 \leq n \wedge 0 \leq j_1 \leq |w_{i_1}| - 1 \wedge 0 \leq j_2 \leq |w_{i_2}| - 1 \wedge\\ &0 \leq y, z \leq 2 \wedge (b \parallel c) \in \{(0 \parallel 1), (2 \parallel 0), (1 \parallel 2)\}\}.\end{aligned}$$

The pairs of numbers for the respective MOD-3 counters are motivated in Figure 4.2. The index counter only increases at places $p^k_{i_k}$ whereas the letter counter only increases at places $p^k_{i_k:j_k:w_{i_k}[j_k]}$. Therefore, the place of one system player is fixed.

We introduce good markings for situations where the environment player claimed that one system player does not terminate but this system player terminated as

$$\begin{aligned}\mathcal{M}_G^{term} = &\{\{(p^1_{term}, y^1, b^1), *, (*, first)\} \mid y^1 \in co^1_{index} \wedge b^1 \in co^1_{letter}\} \cup\\ &\{\{*, (p^2_{term}, z^2, c^2), (*, second)\} \mid z^2 \in co^2_{index} \wedge c^2 \in co^2_{letter}\}.\end{aligned}$$

We define further good markings to only consider firing sequences starting with the environment player's decisions. As the system players never learn about the decision of the environment, this only simplifies our proof later. We define the good marking

$$\mathcal{M}_G^{envfirst} = \{\{(p_i^1, 1_{index}^1, 0_{letter}^1), *, e_{ch})\} \mid 0 \leq i \leq n\} \cup \\ \{\{*, (p_i^2, 1_{index}^2, 0_{letter}^2), e_{ch})\} \mid 0 \leq i \leq n\}.$$

We define bad markings for different output indices or letters at the same counter positions depending on the choice of the environment player as

$$\mathcal{M}_B^{index} = \{\{(p_{i_1}^1, x_{index}^1, b^1), (p_{i_2}^2, x_{index}^2, c^2), (index, *)\} \mid \\ 0 \leq i_1, i_2 \leq n \wedge i_1 \neq i_2 \wedge 0 \leq x \leq 2 \wedge b^1 \in co_{letter}^1 \wedge c^2 \in co_{letter}^2\}$$

and

$$\mathcal{M}_B^{letter} = \{\{(p_{i_1:j_1:w_{i_1}[j_1]}^1, y^1, a_{letter}^1), (p_{i_2:j_2:w_{i_2}[j_2]}^2, z^2, a_{letter}^2), (letter, *)\} \mid \\ 0 \leq i_1, i_2 \leq n \wedge 0 \leq j_1 \leq |w_{i_1}| - 1 \wedge 0 \leq j_2 \leq |w_{i_2}| - 1 \wedge \\ w_{i_1}[j_1] \neq w_{i_2}[j_2] \wedge y^1 \in co_{index}^1 \wedge z^2 \in co_{index}^2 \wedge 0 \leq a \leq 2\}.$$

When one system player terminated, the other system player can make at most one legal step and afterward should terminate as well. Therefore, we define markings where one system player terminated and the other one makes more than one additional step as bad with

$$\mathcal{M}_B^{term_{index}} = \{\{(p_{term}^1, y_{index}^1, b^1), (p_i^2, z_{index}^2, c^2), (index, *)\}, \\ \{(p_i^1, y_{index}^1, b^1), (p_{term}^2, z_{index}^2, c^2), (index, *)\} \mid \\ 0 \leq i \leq n \wedge (y \parallel z) \in \{(0 \parallel 1), (2 \parallel 0), (1 \parallel 2)\} \wedge c^1 \in co_{letter}^1 \wedge z^2 \in co_{letter}^2\}$$

and

$$\mathcal{M}_B^{term_{letter}} = \{\{(p_{term}^1, y^1, b_{letter}^1), (p_{i:j:v_i[j]}^2, z^2, c_{letter}^2), (letter, *)\} \mid \\ 0 \leq i \leq n \wedge 0 \leq j \leq |v_i| - 1 \wedge (b \parallel c) \in \{(0 \parallel 1), (2 \parallel 0), (1 \parallel 2)\} \wedge \\ y^1 \in co_{index}^1 \wedge z^2 \in co_{index}^2\} \cup \\ \{\{(p_{i:j:r_i[j]}^1, y^1, b_{letter}^1), (p_{term}^2, z^2, c_{letter}^2), (letter, *)\} \mid \\ 0 \leq i \leq n \wedge 0 \leq j \leq |r_i| - 1 \wedge (b \parallel c) \in \{(0 \parallel 1), (2 \parallel 0), (1 \parallel 2)\} \wedge \\ y^1 \in co_{index}^1 \wedge z^2 \in co_{index}^2\}.$$

Both system players terminating with different counter values for index or letter counter is also bad behavior but from there no good marking is reachable anymore and we do not need to define it as bad marking.

The constructed Petri game has the form $\mathcal{G} = \mathcal{G}^1 \| \mathcal{G}^2 \| \mathcal{G}^E$ where $\|$ defines the parallel composition as the disjoint union over system and environment places, transitions, flows and initial markings with the good markings $\mathcal{M}_G = \mathcal{M}_G^{finish} \cup \mathcal{M}_G^{index} \cup \mathcal{M}_G^{letter} \cup \mathcal{M}_G^{term} \cup \mathcal{M}_G^{envfirst}$ and the bad markings $\mathcal{M}_B = \mathcal{M}_B^{index} \cup \mathcal{M}_B^{letter} \cup \mathcal{M}_B^{term_{index}} \cup \mathcal{M}_B^{term_{letter}}$.

4.3.2. Correctness of the First Reduction

It is an easy check to see that our usage of $*$ results in disjoint good and bad markings. Our reduction further only requires two system players and a single environment player.

Lemma 10 (Linear firing sequence)**.** *Bad markings for different output indices or letters are reached without a good marking being reached before if and only if the letters are output at the same position.*

Proof. For a strategy with different output indices or letters at position q and $q+3$, a good marking is reached at one of the positions $(q \parallel q+1)$, $(q+2 \parallel q)$, or $(q+1 \parallel q+2)$ as indicated in Figure 4.2. Therefore, positions q and $q+3$ are never compared and no bad marking can be reached for inequality without reaching a bad marking before. □

Lemma 11 (Same number of output indices/letters)**.** *For the constructed Petri game $\mathcal{G}$, strategies outputting a different number of indices or letters at the two system players are not winning.*

Proof. Assume the strategy has output q indices at the first player and $q+1$ indices at the second player (letters analogous). From the structure of the system players, we know that both strategies have to terminate after outputting the letters of the output indices. After both players have terminated, no good marking from $\mathcal{M}_G^{finish}$ is reached because the index counters differ. No good marking was reached before for the linear firing sequence.

Assume the strategy outputs q indices at the first player and $q+2$ indices at the second player (letters analogous). A corresponding bad marking from $\mathcal{M}_B^{term_{index}}$ is reached. □

This proves the absence of untruthful termination.

Lemma 12 (Finite winning strategies)**.** *For the constructed Petri game $\mathcal{G}$, all infinite strategies are not winning.*

Proof. The only possibility for an infinite strategy is to not fire $\#_1$ or $\#_2$ at one or at both system players. If both system players do not fire $\#_1$ and $\#_2$, then the same or two different infinite solutions to the PCP are given which is not winning because, for the linear firing sequence on the indices or the letters, no good marking can be reached. If one process does not fire $\#_1$ or $\#_2$, then different solutions to the PCP are given which is not winning by Lemma 11. □

Lemma 13 (Same output of winning strategies)**.** *For the constructed Petri game $\mathcal{G}$, strategies outputting a different index or letter at the two system players at the same position are not winning.*

Proof. Assume that the strategy differs at position q on the indices i_1 and i_2. The case for letters works in an analog manner. For the turn-taking run of the system players after the decision of the environment player for *index* and *okay*, the bad marking $\{(p^1_{i_1}, x^1_{index}, b^1), (p^2_{i_2}, x^2_{index}, c^2), (index, okay)\}$ for the same value x of the MOD-3 index counter and arbitrary MOD-3 letter counters b^1 and c^2 is reached. □

Lemma 14 (Strategy translation)**.** *For every instance of the PCP, there exists a solution to the instance if and only if there exists a winning strategy in the Petri game from our construction.*

Proof. "⇒": Let $i_1, i_2, \ldots, i_l$ be the solution to the PCP. We build the corresponding strategy in the Petri game where both system players activate the transitions labeled by $i_1, w_{i_1}, \tau, i_2, w_{i_2}, \tau, \ldots, i_l, w_{i_l}, \tau, \#_k$, where all words are output letter-by-letter, and the environment player remains unrestricted. Initially, all three players have transitions enabled. When the transition of the environment player is not the first transition of the considered firing sequence, good markings from $\mathcal{M}_G^{first}$ are reached immediately and we do not need to consider these cases further. When the environment player suspects one system player to not terminate, good markings from $\mathcal{M}_G^{term}$ are reached eventually because the strategy terminates both system players and bad markings from $\mathcal{M}_B^{term_{index}}$ and $\mathcal{M}_B^{term_{letter}}$ are avoided. We therefore can neglect the second decision of the environment player and only need to consider the two cases further where the environment player either wants to check output indices or output letters. For both cases, firing sequences which are not turn-taking on the considered outputs reach good markings from $\mathcal{M}_G^{index}$ or $\mathcal{M}_G^{letter}$. For turn-taking firing sequences, bad markings from $\mathcal{M}_B^{index}$ or $\mathcal{M}_B^{letter}$ are avoided as the same indices and letters are output by the assumption that we translate a solution to the PCP. In the end, a good marking from $\mathcal{M}_G^{finish}$ is reached as the strategy is finite and of the same length at both system players.

"⇐": Winning strategies have to be finite at both system players and have to end with a transition to the final place due to Lemma 11 and Lemma 12. We claim that the allowed sequence of indices is the same at both system players and constitutes a solution to the instance of the PCP. If the sequence of indices is not the same, then the environment player could have tested for different indices and if the sequence of indices is no solution to the instance of the PCP, then the environment player could have tested for different letters. Therefore, each possible winning strategy constitutes a solution to the Post correspondence problem. □

From these lemmas, it follows that the only possibility for a winning strategy is to output the same finite solution to the instance of the PCP at both system players. Therefore, the realizability problem for Petri games with good and bad markings is undecidable as it would be able to decide the existence of such a strategy.

Theorem 8 (One environment player and good and bad markings)**.** *For Petri games with at least two system players, one environment player, and good and bad markings as global winning condition, it is undecidable whether the system players have a winning strategy.*

Proof. Follows from Lemma 10, Lemma 12, Lemma 11, Lemma 13, and Lemma 14. □

4.3.3. Second Reduction

We present the reduction from Petri games with good and bad markings to Petri games with only good markings. Environment places are added such that all players become environment players after each transition in the original Petri game. Transitions are added to reverse the change before the next transition in the original Petri game. For each bad marking, a transition is added from the additional environment places corresponding to the bad marking leading to a new sink place which is not part of any good marking. The environment players can fire the transition to the sink place when a bad marking is reached in the original Petri game. If no such transition is enabled, then the environment players have to return to system places and the next decision of the system players follows. Notice that this construction does not introduce additional synchronization between the players but enables firing orders between the players such that all players are environment players and a transition to the sink place is enabled if a bad marking would be reached in the original Petri game.

As proven in the first reduction, Petri games with good and bad markings and at least two system and one environment player are undecidable. Therefore, Petri games with good markings and at least three players, out of which one is always an environment player and two can change between being a system and environment player are undecidable. Without changing the type of players, this corresponds to Petri games with good markings and at least two system and three environment players where two pairs of system and environment player change between being active and inactive.

In the following, we use the symbols π and τ to introduce and reference new and unique places and transitions. Given a Petri game with good and bad markings $\mathcal{G}^A = (\mathcal{P}_S^A, \mathcal{P}_E^A, \mathcal{T}^A, \mathcal{F}^A, In^A, (\mathcal{M}_G^A, \mathcal{M}_B^A))$, we define the Petri game with good markings $\mathcal{G}^B = (\mathcal{P}_S^B, \mathcal{P}_E^B, \mathcal{T}^B, \mathcal{F}^B, In^B, \mathcal{M}_G^B)$ as

$$\mathcal{P}_S^B = \mathcal{P}_S^A \cup \{\pi'_p \mid p \in In^A \cap \mathcal{P}_S\},$$

$$\mathcal{P}_E^B = \mathcal{P}_E^A \cup \{\pi'_p \mid p \in In^A \cap \mathcal{P}_E\} \cup \{\pi_p \mid p \in \mathcal{P}_S^A \cup \mathcal{P}_E^A\} \cup \{\pi_{sink}\},$$

$$\mathcal{T}^B = \mathcal{T}^A \cup \{\tau_t \mid t \in \mathcal{T}^A\} \cup \{\tau_M \mid M \in \mathcal{M}_B^A\} \cup \{\tau_{In}\},$$

$$\begin{aligned}\mathcal{F}^B = {} & \{(p,t) \mid (p,t) \in \mathcal{F}^A\} \cup \{(t,\pi_p) \mid (t,p) \in \mathcal{F}^A\} \cup \\ & \{(\pi_p,\tau_t) \mid (p,t) \in \mathcal{F}^A\} \cup \{(\tau_t,p) \mid (t,p) \in \mathcal{F}^A\} \cup \\ & \{(\pi_p,\tau_M) \mid M \in \mathcal{M}_B^A \wedge p \in M\} \cup \{(\tau_M,\pi_{sink}) \mid M \in \mathcal{M}_B^A\} \cup \\ & \{(\pi'_p,\tau_{In}) \mid p \in In^A\} \cup \{(\tau_{In},p) \mid p \in In^A\}, In^B = \{\pi'_p \mid p \in In^A\}, \text{ and } \mathcal{M}_G^B = \mathcal{M}_G^A.\end{aligned}$$

Each original place p is preceded by an environment place π_p. They are connected by the transition π_t. For the original successor places p' of p, the places $\pi_{p'}$ are reached from p by transition t. To encode bad markings, we add a sink place π_{sink} which is not part of any good marking. For each bad marking M, the sink place is reachable from the places π_p for the places $p \in M$. The Petri game starts from places π'_p for places p from the initial marking. The original initial marking is reached after firing transition τ_{In}. Notice that this transition can fire at most once and is necessary in case the initial marking is a bad marking.

4.3.4. Correctness of the Second Reduction

Theorem 9 (Three environment players and good markings)**.** *For Petri games with good markings as global winning condition and at least three players, out of which one is an environment player and two change between being a system player and an environment player, it is undecidable whether the system players have a winning strategy.*

Proof. Each winning strategy for $\mathcal{G}^A$ can be translated into a winning strategy for $\mathcal{G}^B$, and vice versa.

Each winning strategy for $\mathcal{G}^A$ reaches no bad markings by the definition of being winning. Therefore, it can be translated into a winning strategy for $\mathcal{G}^B$ by adding places π_p and transitions τ_t as in the construction above. Reaching no bad markings implies that the sink place is never reached. Thus, the constructed strategy for $\mathcal{G}^B$ is a winning strategy because the maximal firing sequences of all maximal plays are the same when removing places π_p and transitions τ_t and because places π_p are not contained in good markings.

Each winning strategy for $\mathcal{G}^B$ never reaches the sink place because it is not contained in any good marking and otherwise the strategy would not be winning. Therefore, it can be translated into a winning strategy for $\mathcal{G}^A$ by removing places π_p and transitions τ_t to reverse the construction above. Never reaching the sink place implies that no bad marking is reached. Thus, the constructed strategy for $\mathcal{G}^A$ is a winning strategy because the maximal firing sequences of all maximal plays are the same when removing places π_p and transitions τ_t and because places π_p are not contained in good markings. □

4.4. Transfer of Undecidability Results to Control Games

Our results transfer to control games based on asynchronous automata, in case the winning condition is defined based on good markings in an equivalent way to our definition. As outlined in our section about related work (cf. Section 1.7), control games define actions (as the equivalent to transitions in Petri games) to be either controllable (as the equivalent to the system) or uncontrollable (as the equivalent to the environment) instead of defining locations of processes (as the equivalent to places of players in Petri games) to belong either to the system or to the environment. Therefore, environment players correspond to processes with only uncontrollable actions, whereas system players correspond to processes with only controllable actions.

The repeated change between system player and environment player to represent bad markings from the first undecidability result in the second one is not needed in control games. Instead, we can augment the locations of the process for the next decision in the form of controllable actions with additional uncontrollable actions in order to have uncontrollable actions from all processes encode when a bad marking for the current decision is reached. To identify this technique in the following, we say that such a process has almost only controllable actions. With these small changes, the same proofs as presented for Petri games apply to control games with good markings. Therefore, for control games with good markings and at least two processes with almost only controllable actions and at least one process with only uncontrollable actions, the question of whether there exists a winning strategy is undecidable.

Notice that good markings are different from final markings in control games. Final markings require that the players reach a final marking and terminate in it, whereas good markings only need to be reached. This implies that the winning condition of good markings does not require when or if players terminate after the players have reached a good marking. The core idea of our proofs is not applicable to final markings because the independence of the three players or processes and final markings cannot be used to restrict the asynchronous nature of control games or Petri games to a synchronous setting. For final markings, all schedulings between the three players would be tested because they only terminate after outputting their solutions to the PCP. In this case, unbounded memory would be needed to compare the output solutions. The independence of the three players or processes prevents them from ever knowing the position of other players or processes, which is the only option to terminate together in a final marking. Therefore, the transfer of our undecidability results to control games do not contradict the decidability results for control games with final markings [Gim17].

4.5. Summary

We showed that, after the global winning condition of bad markings, one quickly reaches undecidable cases for Petri games with global winning conditions. In particular, we proved that it is undecidable whether a winning strategy exists for the system players in Petri games with at least two system players, at least one environment player, and good and bad markings as global winning condition. We also proved that it is undecidable whether a winning strategy exists for the system players in Petri games with at least three players, out of which one player is always an environment player and two players can change between being a system and environment player and good markings as global winning condition. We also outlined how these undecidability proofs apply to control games based on asynchronous automata.

The underlying idea of the undecidability proofs is to simulate the synchronous setting of synthesis of distributed systems in Petri games which are asynchronous by definition. In the synchronous setting, two system players can be forced to simulate a Turing machine and an environment player checks that this is done truthfully by both system players. This is possible for the environment player because the system play-

ers have no information about the position of simulation of the other system players. Therefore, the only possibility for the system players to win is to truthfully simulate the Turing machine, and deciding the existence of a winning strategy for the system players in synthesis of distributed systems in the synchronous setting could decide the undecidable halting problem for Turing machines.

We showed that good markings in Petri games can identify firing sequences as winning that deviate too much from the synchronous setting. Therefore, only equivalent cases to the synchronous setting remain and the same argument as before applies to obtain undecidability. Bad markings or the increased number of environment players are used to detect wrong simulations of the Turing machine. In our proof, we used the Post correspondence problem (PCP) only for a simpler proof than when using Turing machines.

Part II.

Bounded Synthesis

Chapter 5

Bounded Synthesis for Bad Markings and for Good Markings

In this part of the thesis, we present bounded synthesis for Petri games. Bounded synthesis can alleviate the troubles of engineers when manually implementing distributed systems, which is an error-prone task due to the asynchronous interplay of components and the environment. Bounded synthesis is a semi-decision procedure to find winning strategies for the system players in Petri games. If a winning strategy for the system players exists, then bounded synthesis eventually returns such a winning strategy. Bounded synthesis cannot prove that *no* winning strategy for the system players in a Petri game exists. Meanwhile, it is applicable to Petri games with a bounded number of players and an arbitrary distribution between system and environment players. In particular, bounded synthesis can find winning strategies for the system players in Petri games with several system players *and* several environment players. Therefore, it allows to find winning strategies for the system players in Petri games where the restriction on the number of system players or on the number of environment players for the decidability results (cf. [FG17] and Chapter 3) cannot be met.

In a nutshell, bounded synthesis consists of two steps: First, a finite representation, which is called the *bounded unfolding*, of the possibly infinite unfolding of the Petri game is generated. Second, the existence of a winning strategy for the system players in the bounded unfolding is encoded as a *quantified Boolean formula* (QBF) and then solved by a corresponding QBF solver. These two steps are repeated with bounded unfoldings of increasing size until a winning strategy is found.

To generate bounded unfoldings, we utilize the *graph of reachable markings* of a Petri game. This graph has all reachable markings of the Petri game as vertices and edges between these markings corresponding to all possible firings of transitions. The case where bounded synthesis cannot find winning strategies for the system players can only

occur in Petri games with loops in the graph of reachable markings. The reason for this is that Petri games without loops in the graph of reachable markings only encode finite behavior. We will utilize this fact by having two distinctive algorithms for generating bounded unfoldings.

Bounded synthesis expands the cases where we can find winning strategies for the system players by decidability results: For bad markings, the existence of a winning strategy for the system players in Petri games with a bounded number of players is decidable if the number of system players can be bounded to one [FG17] or the number of environment players can be bounded to one (cf. Chapter 3).

Our presentation of bounded synthesis in this part of the thesis is based on *safe* Petri games, i.e., every place of the considered Petri games can contain at most one token in every reachable marking. This allows for a simpler notation and for the usage of QBF solvers. By using an SMT solver instead of a QBF solver, these results could be extended to k-bounded Petri games, i.e., to Petri games where every place can contain at most k tokens in every reachable marking.

In this chapter, we introduce the *sequential QBF encoding* of bounded synthesis for bad markings as winning condition and for good markings as winning condition. In Chapter 6, we show that we can optimize the sequential QBF encoding to the *true concurrent QBF encoding* when bad places as (less expressive) local winning condition are sufficient instead of bad markings as global winning condition. There, we also evaluate our implementation of the sequential QBF encoding and the true concurrent QBF encoding on a large set of benchmarks.

As mentioned before, bounded synthesis encodes the existence of a winning strategy for a bounded unfolding as QBF. A *bounded unfolding* is a finite representation of the (possibly infinite) unfolding where each place is repeated at most as often as a given bound. If the QBF solver returns satisfiable (SAT) as result for the given QBF encoding the existence of a winning strategy for the system players, then we obtain a winning strategy for the system players in the Petri game from the by the QBF solver returned valuation of the existential variables in the QBF. If the QBF solver returns unsatisfiable (UNSAT) as result for the given QBF encoding the existence of a winning strategy for the system players, then the bound for the bounded unfolding is increased and bounded synthesis recursively continues with the extended bounded unfolding.

The key contributions of this chapter are the following:

- We present efficient algorithms to obtain *bounded unfoldings* of Petri games. For this, we differentiate between Petri games with and without loops in their graphs of reachable markings. We make use of this differentiation because the bounded unfolding is equal to the unfolding in the case without loops. These algorithms constitute the first step of bounded synthesis for Petri games.

- We encode the existence of a winning strategy for the system players in a bounded unfolding of a Petri game with the global winning condition of *bad markings* as QBF. This QBF encodes the possible decisions of the system players as existential variables and the possible positions of the tokens as universal variables. This

allows for an efficient solving of the second and last step of bounded synthesis for Petri games. For a satisfiable QBF, the solver also returns an evaluation of the existential variables that directly encodes a winning strategy.

- We encode the existence of a winning strategy for the system players in a bounded unfolding of a Petri game with the global winning condition of *good markings* as QBF. This QBF can be solved as efficiently as in the previous case. As before, implementations are obtained from the output of the QBF solver.

This chapter is structured as follows: In Section 5.1, we present a motivating example for bounded synthesis of Petri games in the form of a Petri game with more than one system player and more than one environment player. We outline why all players in this example are necessary. In Section 5.2, we define bounded unfoldings and bounded winning strategies to lay down the theoretical foundations for bounded synthesis of Petri games. In Section 5.3, we present two algorithms to obtain bounded unfoldings both for Petri games with and without loops in the graph of reachable markings of their underlying Petri nets. In Section 5.4 and Section 5.5, we present the QBF encoding for bounded synthesis of Petri games with bad markings and with good markings as winning condition. In Section 5.6, we outline how to obtain winning bounded strategies from the output of the QBF solver for satisfiable instances of the QBF encoding.

5.1. Motivating Example

We motivate bounded synthesis with the example from Figure 5.1. We start with a short high-level overview of the example. In the remainder of this section, we present the example and its winning strategy more formally. We conclude this section with a discussion why all players in the motivating example are necessary.

The Petri game from Figure 5.1 has up to six system players and up to two environment players. In broad terms, this Petri game models two concurrent robots working on one product where each robot can fail uncontrollably. The first robot is depicted on the left-hand side of Figure 5.1 whereas the second robot is depicted on the right-hand side of Figure 5.1. In case of a failure of one robot and in case of failures of both robots, exactly one robot has to repair the product. The possible failures occur from the two environment places. The two robots can communicate with each other via two one-way communication channels. The two communication channels are depicted in the lower center of Figure 5.1. Two system players are necessary to model the two robots. Two environment players are necessary to let each robot fail uncontrollably without sharing the causal past of the other robot. The remaining four system players are used for the two one-way communication channels and to store whether each robot failed or not.

5.1.1. Description of the Two Robots

In the following, we give a more detailed description of the two robots. The first robot is identified by x and depicted on the left-hand side of Figure 5.1. The second robot

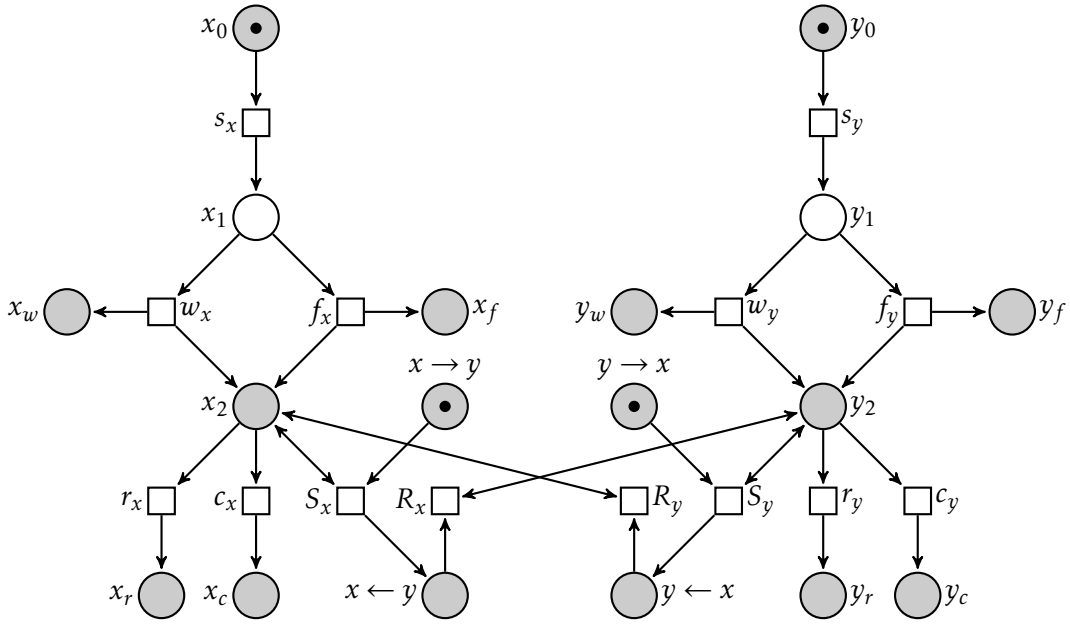


Figure 5.1.: A Petri game to synthesize the local controllers of two robots is depicted. Each robot can fail uncontrollably which requires repair by exactly one of the two robots. The robots can communicate with each other via one-way communication. The bad markings of the Petri game are defined such that the following three cases are avoided by the system players. First, it is defined as bad behavior when no repair of the product occurs despite a failure of one robot or failures of both robots damaging the product. Second, it is defined as bad behavior when the product is undamaged due to none of the two robots failing but a futile repair of the undamaged product occurs. Third, it is defined as bad behavior when the product is damaged due to a failure of one or both robots and both robots repair the damaged product.

is identified by y and depicted on the right-hand side of Figure 5.1. Both robots have the same structure and are concurrent except for the possible communication before making the decision to repair or to not repair the product. From the places x_0 and y_0 with a token each initially, the two robots start to work with the transitions s_x and s_y, respectively. These transitions lead to the environment places x_1 and y_1, respectively, which are used to realize uncontrollable behavior. From places x_1 and y_1, either the work is completed (w) or the robot fails (f). Transitions w_x and w_y encode the work being completed whereas transition f_x and f_y encode the respective robot failing. The robots continue in places x_2 and y_2, respectively. Places x_w, x_f, y_w, and y_f are used to store the decisions of the environment players regarding which robot(s) failed.

From places x_2 and y_2, the robots can either communicate with each other or decide to repair or to not repair the product. The communication between the two robots works via two one-way communication channels, from x to y and from y to x The communication is depicted in the lower center of Figure 5.1. Each communication

channel consists of two steps: At the start, a robot can initiate sending its causal past to the other robot via transition S_x or S_y. Afterward, a robot can receive the causal past of the other robot via transition R_x or R_y. The places $x \rightarrow y$, $x \leftarrow y$, $y \rightarrow x$, and $y \leftarrow x$ transfer the causal past and ensure that each robot can send its causal past to the other robot only once. Without a restriction on the number of sent messages, the robots could send infinitely many messages to each other to satisfy the global safety winning condition. The decision to repair the product (r) or to continue without repairing the product (c) happens via transitions r_x, c_x, r_y, and c_y. The made decisions by the two robots are stored in places x_r, x_c, y_r, and y_c.

We define bad markings to realize the intended behavior. Here, the position of the tokens for both one-way channels is arbitrary. Bad behavior occurs if one or both robots failed (observed by x_f or y_f) but no repair occurred (observed by x_c and y_c), i.e., $\mathcal{M}_B^{missingrepair} = \{\{x_f, y, x_c, y_c, c_1, c_2\} \mid y \in \{y_w, y_f\} \wedge c_1 \in \{x \rightarrow y, y \leftarrow x\} \wedge c_2 \in \{y \rightarrow x, y \leftarrow x\}\} \cup \{\{x, y_f, x_c, y_c, c_1, c_2\} \mid x \in \{x_w, x_f\} \wedge c_1 \in \{x \rightarrow y, y \leftarrow x\} \wedge c_2 \in \{y \rightarrow x, y \leftarrow x\}\}$. Furthermore, it is defined as bad behavior when the product is not damaged due to no robot failing (observed by x_w and y_w) but a repair occurred (observed by x_r or y_r), i.e., $\mathcal{M}_B^{futilerepair} = \{\{x_w, y_w, x_r, y, c_1, c_2\} \mid y \in \{y_r, y_c\} \wedge c_1 \in \{x \rightarrow y, y \leftarrow x\} \wedge c_2 \in \{y \rightarrow x, y \leftarrow x\}\} \cup \{\{x_w, y_w, x, y_r, c_1, c_2\} \mid x \in \{x_r, x_c\} \wedge c_1 \in \{x \rightarrow y, y \leftarrow x\} \wedge c_2 \in \{y \rightarrow x, y \leftarrow x\}\}$. Last but not least, it is defined as bad behavior when both robots repair the product (observed by x_r and y_r) and the product is damaged (observed by x_f or y_f), i.e., $\mathcal{M}_B^{bothrepair} = \{\{x_f, y, x_r, y_r, c_1, c_2\} \mid y \in \{y_w, y_f\} \wedge c_1 \in \{x \rightarrow y, y \leftarrow x\} \wedge c_2 \in \{y \rightarrow x, y \leftarrow x\}\} \cup \{\{x, y_f, x_r, y_r, c_1, c_2\} \mid x \in \{x_w, x_f\} \wedge c_1 \in \{x \rightarrow y, y \leftarrow x\} \wedge c_2 \in \{y \rightarrow x, y \leftarrow x\}\}$. The set of bad markings is the union of the three sets, i.e., $\mathcal{M}_B = \mathcal{M}_B^{missingrepair} \cup \mathcal{M}_B^{futilerepair} \cup \mathcal{M}_B^{bothrepair}$.

The position of the tokens for both one-way channels needs to be included in the bad markings because a bad marking is only reached when a reachable marking exactly matches a bad marking, i.e., being a subset does not suffice. If already a subset of a reachable marking should represent bad behavior, then all reachable markings that contain the subset can be defined as bad markings.

5.1.2. Winning Strategies for the Two Robots

A winning strategy for the system players allows transitions s_x and s_y in the beginning and then imposes an order in which the one-way communication channels exchange information, e.g., by first only allowing transition S_x and afterward only allowing transition R_y for the system player in place x_2 and by first only allowing transition R_x and afterward only allowing S_y for the other system place in place y_2. After completed communication, the winning strategy for the system players lets the responsible robot repair the product if only one robot failed. This is achieved by the responsible robot only allowing transition r_x or r_y and the not responsible robot only allowing transition c_x or c_y. If both robots failed, then robot x repairs the product by only allowing transition r_x and robot y continues without repairing the product by only allowing transition c_y. Swapping the behavior of robot x and robot y in this case does also work.

If no robot failed, then both robots do not repair the product and continue instead by only allowing transition c_x and transition c_y, respectively.

Another winning strategy for the system players imposes an order between the two robots. Here, only robot x sends information whether it failed to robot y. Robot y is then informed about whether each robot failed and decides to repair or to not repair the product. Robot x never repairs the product and trusts robot y to repair the product when needed. Swapping the roles of robots x and y in this strategy does also work.

5.1.3. Discussion

We discuss why the number of system players and of environment players cannot be reduced to one for the Petri game in Figure 5.1 and the used winning condition. Such a reduction would allow us to use decidability results instead of bounded synthesis.

As there are two distributed robots, we need at least two system players to obtain two distributed controllers. There are only two options to reduce the number of environment players. First, the one and only environment player could with one transition decide for both robots whether they fail. In this case, both robots would be informed about whether they failed *and* whether the other robot failed. Then, without communication, the winning strategy for the system players could choose one robot that always repairs the product if needed. This could not be implemented in the real world because the two robots lack information about the other robot due to missing communication.

Second, one environment player could decide whether a robot fails for one robot after the other. In this case, the first robot could trust the second robot to always repair if necessary without any communication between the two robots. The reason for this is that the second robot is informed about the decision of the environment player for the first robot when the environment player decides whether the second robot fails. This could not be implemented in the real world because the second robot would lack information about the first robot to make the appropriate decision due to missing communication. Notice in both cases that a player cannot forget its causal past and always shares its entire causal past upon synchronization with another player, i.e., it cannot hide parts of its causal past.

The specification for the two robots is expressible with the local winning condition of bad places by having transitions from each bad marking leading to a bad place. This is only possible as the example has no infinite behavior. For Petri games with infinite behavior, the winning condition of bad markings allows us to specify losing behavior between players without requiring their synchronization which is impossible for the winning condition of bad places. We discuss how the motivating example can be extended to repeat itself in a loop. We need to add transitions from all combinations of behaviors of the two robots and the two one-way communication channels to the initial marking. The interested reader can check that 64 transitions would be needed. For simplicity reasons, we omitted these transitions. Each of the 64 transitions would inform all players about the behavior of the other players in the previous round. This information is not helpful in the then started round and our discussion about the necessary number of environment players and of system players from above still applies.

5.2. Bounded Unfoldings and Bounded Strategies

We recall the definitions of bounded unfoldings and bounded strategies [Fin15]. We include a homomorphism in the definition of a b-bounded strategy. In Petri games with bad markings or good markings as winning condition, a finite strategy suffices to win the game because the same decision can be repeated from some point onward to fulfill the winning condition. Bounded synthesis directly searches for bounded strategies which are by definition of finite size.

Definition 48 (Finite generator)
A strategy $\sigma = (\mathcal{N}^\sigma, \lambda^\sigma)$ is *finitely generated* if there exists a finite Petri net $\mathcal{N}^f$ such that $(\mathcal{N}^\sigma, \lambda^\sigma)$ is an unfolding of $\mathcal{N}^f$. We say that σ is *finitely generated by* $\mathcal{N}^f$ and that $\mathcal{N}^f$ is the *finite generator* of σ.

Notice that the homomorphism λ^σ is defined as σ being an unfolding of $\mathcal{N}^f$. Notice further that finitely generated strategies are of infinite size when the finite generator contains loops in the graph of reachable markings.

We search for the finite generator $\mathcal{N}^f$ by considering *b-bounded unfoldings* of the underlying Petri net of the Petri game, for which we are using bounded synthesis to find a winning strategy for the system players. In a b-bounded unfolding, $b : \mathcal{P} \to \mathbb{N}$ assigns a natural number to each place of the Petri net. This number limits the number of copies of the places in the b-bounded unfolding.

Definition 49 (b-bounded unfolding)
A *b-bounded unfolding* of a Petri net $\mathcal{N}$ is a pair $(\mathcal{N}^b, \lambda^b)$ consisting of a finite Petri net $\mathcal{N}^b$ and a homomorphism λ^b from $\mathcal{N}^b$ to the Petri net $\mathcal{N}$ if the following conditions hold:

- There exists a homomorphism λ from $\mathcal{N}^U$ of the unfolding $\iota_U = (\mathcal{N}^U, \lambda^U)$ of $\mathcal{N}$ to $\mathcal{N}^b$ such that $\lambda^U(x) = \lambda^b(\lambda(x))$ for all nodes x of $\mathcal{N}^U$.
- $|(\lambda^b)^{-1}(p)| \leq b(p)$ holds for every $p \in \mathcal{P}$, where $(\lambda^b)^{-1}(p)$ returns the set of places representing different causal pasts in the b-bounded unfolding of a place p of the Petri net $\mathcal{N}$.

The three homomorphisms have the following types: $\lambda^b : \mathcal{N}^b \to \mathcal{N}$, $\lambda : \mathcal{N}^U \to \mathcal{N}^b$, and $\lambda^U : \mathcal{N}^U \to \mathcal{N}$. The first requirement ensures that a b-bounded unfolding is an intermediate representation between the Petri net representing no causal history and the unfolding explicitly representing all causal history. The second requirement ensures that each place p of $\mathcal{N}$ occurs at most $b(p)$ times in $\mathcal{N}^b$.

We restrict the flow of a b-bounded unfolding to obtain *b-bounded strategies*.

Definition 50 (b-bounded strategy)
A *b-bounded strategy* for a Petri game $\mathcal{G}$ is a pair $(\mathcal{N}^f, \lambda^f)$ consisting of a finite Petri net $\mathcal{N}^f$ and a homomorphism λ^f from $\mathcal{N}^f$ to the underlying Petri net $\mathcal{N}$ of $\mathcal{G}$ such that the following conditions holds:

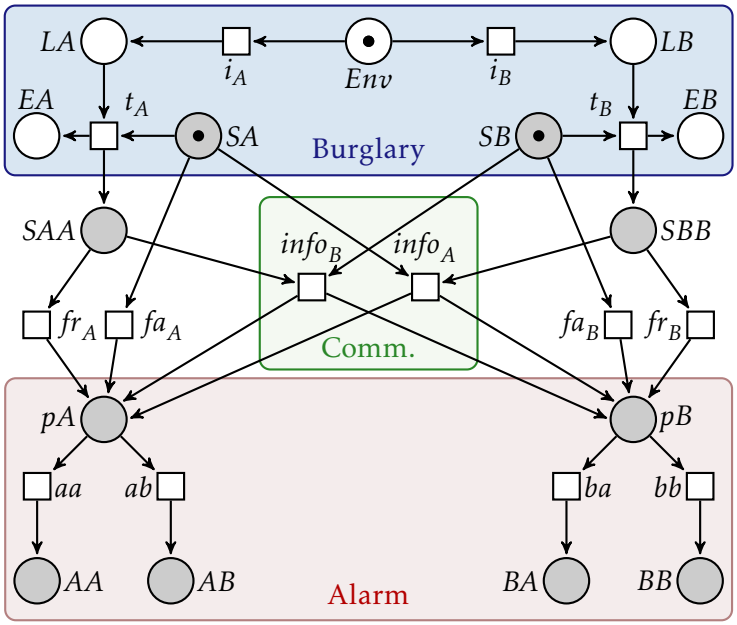


Figure 5.2.: A Petri game modeling the synthesis problem for a distributed alarm system is depicted. There are two parts of the distributed alarm system at the locations A and B modeled by two system players and one burglar modeled by one environment player. The burglar can intrude one of the two locations and both parts of the distributed alarm system should only in case of a burglary sound an alarm indicating the location of the burglary. The set of bad markings is defined in such a way that false alarms and false reports are avoided. A false alarm happens when an alarm is sounded before the burglary takes place. A false report happens when an alarm indicates a different location than the location where the burglary takes place.

- There is a b-bounded unfolding $(\mathcal{N}^b, \lambda^b)$ of the underlying Petri net $\mathcal{N}$ of $\mathcal{G}$ with $\mathcal{P}^f \subseteq \mathcal{P}^b$, $\mathcal{T}^f \subseteq \mathcal{T}^b$, $In^f = In^b$, $\mathcal{F}^f \subseteq \mathcal{F}^b$, and $\lambda^f = \lambda^b \restriction (\mathcal{P}^f \cup \mathcal{T}^f)$.
- There is a strategy σ that is finitely generated by $\mathcal{N}^f$.

We say that $\mathcal{G}^b$ *admits* the b-bounded strategy $\mathcal{G}^f$. The bounded strategy $\mathcal{G}^f$ is winning if σ is winning. We sometimes omit b in the b-bounded unfolding and the b-bounded strategy to aid readability.

Example 5.2.1. We present an example for the concepts of bounded unfoldings and of bounded strategies. Consider the exemplary Petri game in Figure 5.2 which is a simple example of a distributed alarm system from [FO17] presented as in [FGHO17]. It only has one environment player and therefore does not necessarily require bounded synthesis to find a winning strategy but allows for a more concise presentation of the concepts of bounded unfoldings and of bounded strategies.

In the Petri game for the distributed alarm system, a burglar (modeled by the environment player residing in place *Env*) decides to intrude one of two secured locations A (shown on the left-hand side of Figure 5.2) and B (shown on the right-hand side of Figure 5.2). The goal of the Petri game is to find a winning strategy for the system players initially residing in places SA and SB, modeling the distributed alarm system. A token in place XY (for $X, Y \in \{A, B\}$) represents that in location X an alarm is sounded 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 strategy for the system players should steer the token in SA to place AA and the token in SB to place BA to correctly indicate the intrusion. Analog behavior is needed for the burglar intruding location B.

A false alarm occurs when an intrusion is indicated by an alarm system at one location before the burglar actually intruded any location. A false report occurs when the alarm system at one location indicates an intrusion at a location where no intrusion occurred. Formally, the set of bad markings $\mathcal{M}_B$ is defined as the union of the set of bad markings for a *false alarm*

$$\begin{aligned}\mathcal{M}_B^{falseAlarm} = &\{\{Env, A, B\} \mid A \in \{AA, AB\} \wedge B \in \{SB, pB, BA, BB\}\} \cup \\ &\{\{Env, A, B\} \mid A \in \{SA, pA, AA, AB\} \wedge B \in \{BA, BB\}\}\end{aligned}$$

and of the set of bad markings for a *false report*

$$\begin{aligned}\mathcal{M}_B^{falseReport} = &\{\{E, AB, B\} \mid E \in \{LA, EA\} \wedge B \in \{SB, SBB, pB, BA, BB\}\} \cup \\ &\{\{E, A, BB\} \mid E \in \{LA, EA\} \wedge A \in \{SA, SAA, pA, AA, AB\}\} \cup \\ &\{\{E, AA, B\} \mid E \in \{LB, EB\} \wedge B \in \{SB, SBB, pB, BA, BB\}\} \cup \\ &\{\{E, A, BB\} \mid E \in \{LB, EB\} \wedge A \in \{SA, SAA, pA, AA, AB\}\}.\end{aligned}$$

In all bad markings from the set of bad markings for a false alarm $\mathcal{M}_B^{falseAlarm}$, the environment player has not left place *Env*. Meanwhile, the alarm system at location A has sounded an alarm in the first part of $\mathcal{M}_B^{falseAlarm}$ and the alarm system at location B has done so in the second part of $\mathcal{M}_B^{falseAlarm}$.

For the set of bad markings for a false report $\mathcal{M}_B^{falseReport}$, the environment player has intruded location A, i.e., is either in place LA or in place EA, in the first two parts of $\mathcal{M}_B^{falseReport}$ and has intruded location B, i.e., is either in place LB or in place EB, in the remaining two parts of $\mathcal{M}_B^{falseReport}$. Meanwhile, the alarm system at location A has sounded a wrong alarm in the first and third part of $\mathcal{M}_B^{falseReport}$ and the alarm system at location B has sounded a wrong alarm in the second and fourth part of $\mathcal{M}_B^{falseReport}$, while, in all four parts, the position of the alarm system at the other location is arbitrary.

Notice that we define the set of bad markings such that bad behavior is recognized as early as possible. This illustrates the expressive possibilities of bad markings. For example, every false alarm becomes a false report when the environment player makes a decision opposite to the already sounded alarm of one of the two alarm systems.

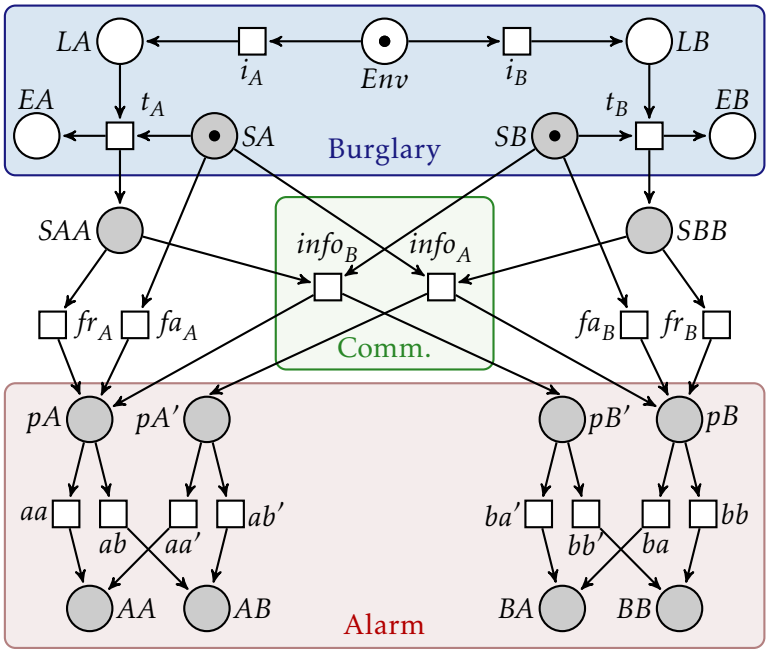


Figure 5.3.: A bounded unfolding for the Petri game from Figure 5.2 is depicted.

Depending on the interpretation of places LA and LB, one could add additional bad markings. Our interpretation is that the burglar has already intruded location A or location B when reaching place LA or place LB. Only when reaching place EA or place EB, the respective alarm system has picked up on the intrusion. Therefore, it is not bad behavior when an alarm system sounds the correct alarm while the environment player is still in place LA or in place LB, i.e., pairs of places from $LA \times \{AA, BA\}$ and pairs of places from $LB \times \{AB, BB\}$ are not contained in any bad marking.

An alternative interpretation would be that the environment player reaching place LA or place LB only declares its intention to burgle the respective location but no actual intrusion takes place. The actual intrusion only occurs when reaching place EA or place EB and is immediately detected by the respective alarm system. With this interpretation, markings containing pairs of places from $LA \times \{AA, BA\}$ or pairs of places from $LB \times \{AB, BB\}$ should be contained in the set of bad markings for false reports, because an alarm is sounded while no intrusion occurred. We will later see why both interpretations result in the same winning bounded strategy.

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 strategy for the system players in SA and SB needs 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 $info_B$.

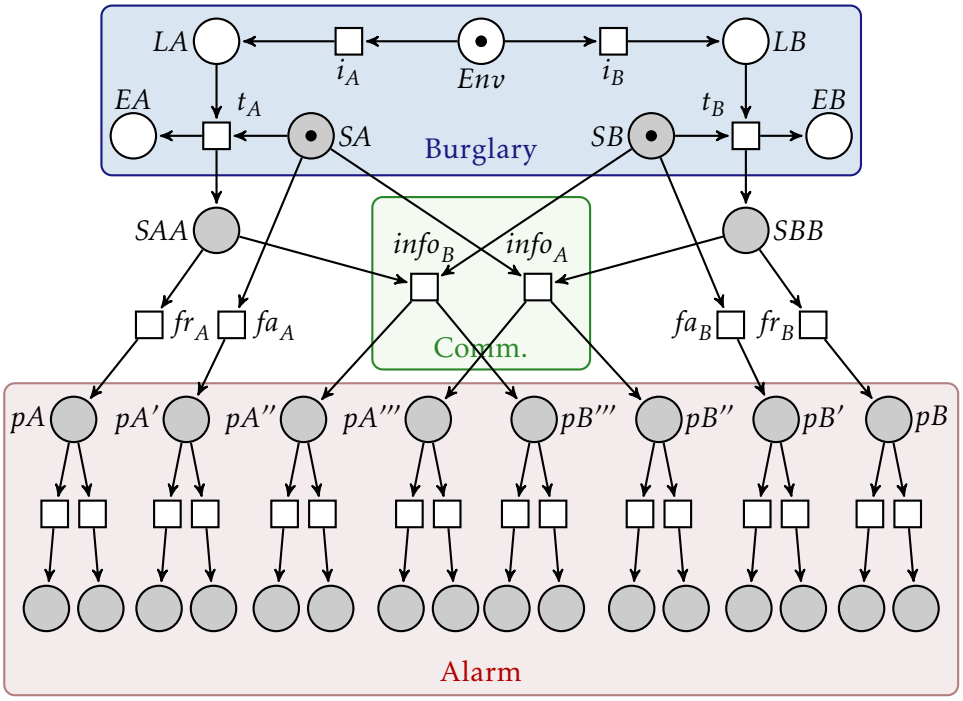


Figure 5.4.: The unfolding for the Petri game from Figure 5.2 is depicted. In the lower part of the figure, the names of transitions based on transitions *aa*, *ab*, *ba*, and *bb* and the names of places based on places *AA*, *AB*, *BA*, and *BB* are omitted to aid readability. Places pA, pA', pA'', and pA''' representing place pA in the original Petri game have one outgoing transition to a place representing transition *aa* to place *AA* and one outgoing transition to a place representing transition *ab* to place *AB*. Analogously, places pB, pB', pB'', and pB''' representing place pB in the original Petri game have one outgoing transition to a place representing transition *ba* to place *BA* and one outgoing transition to a place representing transition *bb* to place *BB*.

Figure 5.3 presents a bounded unfolding for the Petri game from Figure 5.2. The (general) unfolding for the Petri game from Figure 5.2 is depicted in Figure 5.4. The bounded unfolding only copies the places pA and pB once each in the form of places pA' and pB'. These places have unique outgoing transition aa', ab', ba', and bb'. In the (general) unfolding, there are four places each representing the places pA and pB in the original Petri game. These eight places have unique outgoing transitions leading to unique places. To find a winning bounded strategy for a bounded unfolding, it is essential that transitions $info_B$ and $info_A$ do not share any places in their postcondition. The depicted bounded unfolding in Figure 5.3 is one minimal (in terms of number of places and transitions in the bounded unfolding) way to achieve this. Different minimal bounded unfoldings can change the postcondition of transitions fr_A and fa_A to include place pA' instead of place pA and the postcondition of transitions fr_B and fa_B to include place pB' instead of place pB.

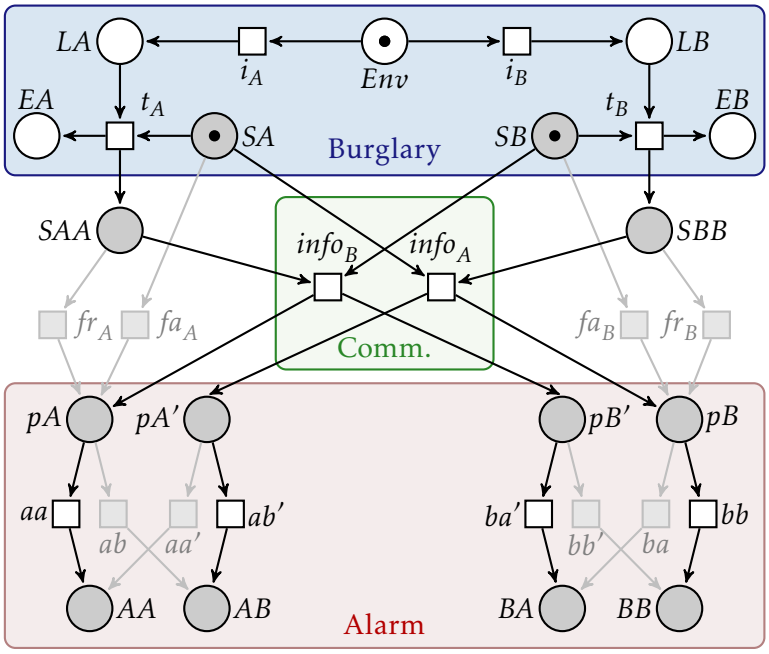


Figure 5.5.: A winning bounded strategy for the bounded unfolding from Figure 5.3 is depicted. The winning bounded strategy is obtained by leaving out the grayed-out parts from the bounded unfolding.

Figure 5.5 depicts a bounded winning strategy for the bounded unfolding shown in Figure 5.3 for the Petri game from Figure 5.2. The bounded 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 two interpretations whether transitions i_A and i_B model the intention of the burglar or the actual intrusion result in the same winning strategy, because disabling transitions t_A and t_B can never be winning regardless whether the additional bad markings are included or not. Disabling t_A or t_B prevents the environment from reaching place EA or place EB which makes it impossible to reach bad markings for a false report. The requirement of deadlock-avoidance then forces the system players to allow transitions fa_A or fa_B, because transitions $info_B$ or $info_A$ can only fire after t_A or t_B fired. After fa_A and fa_B, the system players have to sound an alarm which results in a false alarm, because the environment player has not necessarily left place Env. △

5.3. Generating Bounded Unfoldings

In this section, we describe how we can generate bounded unfoldings to use them for bounded synthesis of Petri games. We differentiate between Petri games with and without loops in the graph of reachable markings of their underlying Petri nets. Remember that markings correspond to sets of places (instead of multisets over places), because we only consider safe Petri nets in this part of the thesis, i.e., Petri nets where each place can contain at most one token for all reachable markings. For Petri games without loops in the graph of reachable markings of their underlying Petri nets, we generate the finite (general) unfolding. By contrast, for Petri games with loops in the graph of reachable markings of their underlying Petri nets, we generate a finite b-bounded unfolding.

Definition 51 (Graph of reachable markings)
The *graph of reachable markings* $G = (V, E)$ of a Petri net $\mathcal{N}$ is defined as follows:

- $V = \mathcal{R}(\mathcal{N})$
- $E = \{(M, M') \mid M \in \mathcal{R}(\mathcal{N}) \wedge \exists t \in \mathcal{T} : M[t\rangle M'\}$

The vertices of the *graph of reachable markings* of a Petri net are all reachable markings of the Petri net. The edges correspond to transitions being fired between markings. The graph of reachable markings can be calculated recursively: One starts with the initial marking as the first vertex. In a vertex, all enabled transitions are considered: The marking reached by an enabled transition firing is added to the set of vertices (if not already present) and an edge from the current vertex to the reached marking is added (if not already present). This process is repeated recursively for newly added vertices until no more elements are added.

5.3.1. Petri Games without Loops

For Petri games without loops in the graph of reachable markings of their underlying Petri nets, we use McMillian's unfolding algorithm [McM95]. We follow a more recent presentation [ERV02]. We define the possible extensions of a branching process as pairs of transitions and markings. With $\mathcal{G}$, we identify the Petri game for which we build the unfolding or a b-bounded unfolding depending on whether the underlying Petri net $\mathcal{N}$ of $\mathcal{G}$ has loops in its graph of reachable markings. With ι, we identify the branching process of $\mathcal{N}$ that will become the unfolding of $\mathcal{N}$ when the used algorithm terminates.

Definition 52 (Possible extensions [ERV02])
The *possible extensions* of a branching process $\iota = (\mathcal{N}^\iota, \lambda^\iota)$ are the pairs (t, M) where t is a transition of $\mathcal{N}$ and M is a subset of or equal to a reachable marking in $\mathcal{N}^\iota$ such that:

- $\lambda^\iota(M) = pre^{\mathcal{N}}(t)$ and
- $\mathcal{T}^\iota$ contains no transition t_0 with $\lambda^\iota(t_0) = t$ and $pre^\iota(t_0) = M$.

$PE(\iota)$ denotes the set of all possible extensions of ι.

Algorithm 1: McMillian's unfolding algorithm [McM95; ERV02]

input : Petri net $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, In)$ without loops in its graph of reachable markings
output: unfolding $\iota_U = (\mathcal{N}^U, \lambda^U)$

create empty Petri net $\mathcal{N}^U = (\mathcal{P}^U, \mathcal{T}^U, \mathcal{F}^U, In^U)$;
create empty homomorphism λ^U;
for each $p \in In$, add unique p to $\mathcal{P}^U$ and extend λ^U to $\lambda^U[\lambda^U(p) = p]$;
set $In^U = In$;
$pe = PE(\iota_U)$;
while $pe \neq \emptyset$ **do**
    pick (t, M) from pe;
    add unique t' for t to $\mathcal{T}^U$ and extend λ^U to $\lambda^U[\lambda^U(t') = t]$;
    for each $p \in post^{\mathcal{N}}(t)$, add unique p' to $\mathcal{P}^U$ and extend λ^U to $\lambda^U[\lambda^U(p') = p]$;
    add flows from M to t' and from t' to the newly created places to $\mathcal{F}^U$;
    $pe = PE(\iota_U)$;

McMillian's unfolding algorithm is given in Algorithm 1. It recursively adds possible extensions from the initial marking until no more possible extensions exist. For a homomorphism λ from Petri net $\mathcal{N}'$ to Petri net $\mathcal{N}$ and nodes $x' \in \mathcal{P}' \cup \mathcal{T}'$ and $x \in \mathcal{P} \cup \mathcal{T}$, we use the notation $\lambda[\lambda(x') = x]$ to extend λ in such a way that it fulfills $\lambda(x') = x$ and behaves as before for all nodes different from x'. Notice that the algorithm will terminate and produce a finite unfolding because we only apply it to Petri games without loops in the graph of reachable markings of their underlying Petri nets.

Algorithm 1 does not use the bound b for the following two reasons: First, the experience with our implementation shows that McMillian's algorithm is very quick for Petri games without loops in the graph of reachable markings of their underlying Petri nets and that solving the QBF encoding for the existence of a strategy for the system players also quickly terminates.

Second, from a finite unfolding, we can also determine the corresponding simulation length to decide the existence of a winning strategy for the system players more quickly than using the worst-case simulation length. The corresponding simulation length is the length of the longest sequence of fireable transitions plus two. As the encoding enumerates sequences of markings, we need to add one to get from sequences of fireable transition to sequences of markings. We need to add another one to find termination at the latest at the second to last marking.

Because of these two reasons, we decided to not use the bound b for Petri games without loops in the graph of reachable markings of their underlying Petri nets and thereby obtain a decision procedure (instead of a semi-decision procedure) for this class of Petri games at the cost of the possibility of a longer runtime. The source of the possibility for longer runtime is that not necessarily every branch of the (general) unfolding is needed but this cannot be known in advance of solving the Petri game.

5.3.2. Petri Games with Loops

For Petri games with loops in the graph of reachable markings of their underlying Petri nets, we utilize the bound b to obtain a b-bounded unfolding and additional system places. These additional system places are equipped with a corresponding flow relation to connect them to the b-bounded unfolding. The additional system places later allow the strategy for the system players to decide how to use the bounded causal memory, i.e., which causal pasts are grouped together upon reaching the bound b for a place.

We extend McMillian's unfolding algorithm [McM95; ERV02] to produce b-bounded unfoldings $\iota_U = (\mathcal{N}^U, \lambda^U)$ and additional system places $\mathcal{P}_S^U$ with a corresponding flow relation $\mathcal{F}_S^U : (\mathcal{P}_S^U \times \mathcal{T}^U) \cup (\mathcal{T}^U \times \mathcal{P}_S^U)$ for Petri games with loops in the graph of reachable markings of their underlying Petri nets. The bound b is used to produce at most $b(p)$ occurrences in the b-bounded unfolding of every place p in the Petri net. Each copy can represent a unique causal past in the b-bounded unfolding. Because there can be more unique causal pasts for a place p than the bound $b(p)$, we add *additional system places* with a corresponding flow relation and system players in them such that the strategy for the system players can decide which causal pasts should be grouped together. Causal pasts that are grouped together result in the same reaction by the strategy for the system players. Notice that there is a system player in each additional system. Thus, we do not explicitly include these system players in an initial marking. Instead, we later on use the set of additional system places $\mathcal{P}_S^U$ as initial marking for these places.

Using additional system places allows us to encode more possible strategies for the system players in the second step of bounded synthesis for Petri games. The reason for this is that the strategy for the system players can decide how to group causal pasts together instead of the unfolding algorithm fixing this. The decisions for the system players in the additional system places are included when obtaining a winning b-bounded strategy for a b-bounded unfolding, i.e., parts of the strategy are removed when the system players decide against firing the corresponding transition. The additional system places with a corresponding flow relation are *not* included in a winning b-bounded strategy because they are only used to make the b-bounded unfolding more expressive in terms of possible strategies for the system players.

For the unfolding algorithm, we assume that b allows each place at least once, i.e., $\forall p \in \mathcal{P} : b(p) \geq 1$. Furthermore, we introduce some notation to identify whether bound b is reached for a place. Each place p can occur at most $b(p)$ times and we enumerate the places by $(p)_1, (p)_2, \ldots, (p)_{b(p)}$. The number of occurrences of place p is defined as $|(p)|$ and can be used to compare against $b(p)$.

The set of all possible extensions is defined as before. The difference between Algorithm 1 and Algorithm 2 is how transitions and places are added to the unfolding. Here, the bound b comes into play and we distinguish two cases. First, when the bound is not reached for any place in the postcondition of t, then we add the same element as in McMillian's unfolding algorithm using explicit place names $(p)_{|(p)|+1}$ to identify reaching the bound (cf. Line 9 to Line 12).

Second, when the bound is reached for some place in the postcondition of t, then places from the postcondition of t are distributed into places where new copies can

Algorithm 2: unfolding algorithm to obtain b-bounded unfoldings

input : Petri net $\mathcal{N} = (\mathcal{P}, \mathcal{T}, \mathcal{F}, In)$ with loops in its graph of reachable markings and a bound $b : \mathcal{P} \to \mathbb{N}$
output: b-bounded unfolding $\iota_U = (\mathcal{N}^U, \lambda^U)$ with additional system places $\mathcal{P}_S^U$ and corresponding flow relation $\mathcal{F}_S^U$

1 create empty Petri net $\mathcal{N}^U = (\mathcal{P}^U, \mathcal{T}^U, \mathcal{F}^U, In^U)$;
2 create empty homomorphism λ^U ;
3 create empty set of system places $\mathcal{P}_S^U$ and empty flow relation $\mathcal{F}_S^U$;
4 for each $p \in In$, add place $(p)_1$ to $\mathcal{P}^U$ and extend λ^U to $\lambda^U[\lambda^U(p') = p]$;
5 set $In^U = \{(p)_1 \mid p \in In\}$;
6 $pe = PE(\iota_U)$;
7 **while** $pe \neq \emptyset$ **do**
8 pick (t, M) from pe;
9 **if** $\forall p \in post^{\mathcal{G}}(t) : |(p)| < b(p)$ **then**
10 add unique t' for t to $\mathcal{T}^U$ and extend λ^U to $\lambda^U[\lambda^U(t') = t]$;
11 for each $p \in post^{\mathcal{G}}(t)$, add unique $(p)_{|(p)|+1}$ to $\mathcal{P}^U$ and extend λ^U to $\lambda^U[\lambda^U((p)_{|(p)|+1}) = p]$;
12 add flows from M to t' and from t' to the newly created places to $\mathcal{F}^U$;
13 **else**
14 $uniquePost, missingPost = \emptyset$;
15 **for** $p \in post^{\mathcal{G}}(t)$ **do**
16 **if** $|(p)| < b(p)$ **then**
17 add unique $(p)_{|(p)|+1}$ to both $\mathcal{P}^U$ and $uniquePost$, and extend λ^U to $\lambda^U[\lambda^U((p)_{|(p)|+1}) = p]$;
18 **else**
19 add p to $missingPost$;
20 **for** $post \in \{uniquePost\} \times \times_{p \in missingPost} \{(p)_1, \ldots, (p)_{b(p)}\}$ **do**
21 add unique transition t' for t to $\mathcal{T}^U$ and extend λ^U to $\lambda^U[\lambda^U(t') = t]$;
22 add flows from M to t' and from t' to the places in $post$ to $\mathcal{F}^U$;
23 add unique p_{sys} to $\mathcal{P}_S^U$;
24 add flows from the transitions created in the last for-loop to p_{sys} to $\mathcal{F}_S^U$;
25 add flows from p_{sys} to the transitions created in the last for-loop to $\mathcal{F}_S^U$;
26 $pe = PE(\iota_U)$;

still be added (cf. Line 17 and set *uniquePost*) and places where the bound b is already reached and places have to be reused (cf. Line 19 and set *missingPost*). Continuing the second case, we add transitions for all possible postconditions where places in *uniquePost* are always part of the postcondition and for the remaining places in *missingPost* all combinations of copies of the place are enumerated (cf. Line 20 to

Line 22). To finish the second case, we add a unique additional system place p_{sys} that has all transitions created in the previous step in its precondition and in its postcondition (cf. Line 23 to Line 25). Following this algorithm, each transition has at most one additional system place in its precondition according to the corresponding flow relation $\mathcal{F}_S^U$ for the additional system places.

When each of the system players in the additional system places allows exactly one of its outgoing transitions, then the additional possibilities introduced by this approach are again removed. Then, it is fixed how different causal pasts are grouped together upon reaching the bound b for places. Choosing a different enabled outgoing transition for one of the system players in additional system places results in a different grouping together of causal pasts. Notice that the made decisions by the system players in additional system places are fixed for the entire game because the additional system places are not unfolded.

The QBF encoding as the second step of bounded synthesis for Petri games will ensure that for the system players in additional system places exactly one successor is chosen. This is achieved by adding a requirement such that each system player in an additional system place chooses at least one of its successors and the determinism requirement enforcing that each system player in an additional system place chooses at at most one of its successors. Instead of the unfolding algorithm fixing one way of grouping together causal pasts, the approach using system players in additional system places allows us to encode *all* possible ways of grouping together causal pasts. The strategy for the system players will choose one of these ways of grouping together causal pasts. By this approach, the b-bounded unfolding with additional system places admits more b-bounded winning strategies at the cost of increased number of system players. The experience with our implementation showed that the increase in system players is outweighed by the increase in expressiveness.

The idea behind this algorithm is inspired by using finite prefixes of unfoldings for model checking of Petri nets [KKV03]. Instead of defining a cutoff based on causal memory and obtaining an unfolding with McMillian's algorithm until the cutoff, we utilize an explicit bound b and give the strategy of the system players the possibility to utilize the additional system places to determine how to use the bounded number of places to represent causal memory.

5.4. Encoding for Bad Markings

The bounded synthesis algorithm [Fin15] takes a Petri game and increases the memory bound b until a winning strategy for the system players is found (or runs forever). In this section, we describe how to encode the existence of a winning strategy for the system players for a finite unfolding as the second step of bounded synthesis for Petri games. The first step of bounded synthesis obtains a finite (general) unfolding or a (finite) bounded unfolding as described in the previous section. The two algorithms to obtain a finite unfolding are not included in [Fin15] but derived newly for this thesis. The QBF encoding presented in the following is mostly from [Fin15]. We contribute the

usage of additional system places in this section, some optimizations in Section 5.4.1, and an extension to good markings as winning condition in Section 5.5.

A finite unfolding $\iota_U^b = (\mathcal{N}^b, \lambda^b)$ is obtained either by Algorithm 1 or by Algorithm 2 depending on whether the graph of reachable markings of the underlying Petri net $\mathcal{N}$ of the Petri game $\mathcal{G}$ with bad markings $\mathcal{M}_B$ does or does not have loops. In the case of loops in the graph of reachable markings, we obtain the additional system places $\mathcal{P}_S^U$ with a corresponding flow relation $\mathcal{F}_S^U$. With $pre^U(p)$ and $post^U(p)$, we identify the precondition and the postcondition according to $\mathcal{F}_S^U$ for places $p \in \mathcal{P}_S^U$. In the case of no loops in the graph of reachable markings, the additional system places $\mathcal{P}_S^U$ are an empty set with an empty corresponding flow relation $\mathcal{F}_S^U$.

The unfolding is used to encode the existence of a winning strategy for the system players (as variables $\mathcal{S}^b$) for all sequences of markings (as variables $\mathcal{M}_n$) up to the maximal simulation length $n \leq 2^{|\mathcal{P}^b|} + 1$ as QBF. As mentioned before, the maximal simulation length can be bounded more sharply for a finite unfolding obtained by Algorithm 1. In the now presented sequential QBF encoding, concurrent transitions are represented by all possible interleavings because between two markings only a single transition is fired.

In the following, we give extensions to the original QBF encoding [Fin15] in green and corrections of notation in blue. We explain both of them in Section 5.4.1. Corrections of notation refer to situations in the original paper where it is clear what is supposed to be expressed, but the used notation does not exist or is not entirely correct. The QBF has the form $\exists \mathcal{S}^b : \forall \mathcal{M}_n : \phi_n$, where

$$\mathcal{S}^b = \{(p, \lambda^b(t)) \mid p \in \mathcal{P}^b \wedge \lambda^b(p) \in \mathcal{P}_S \wedge t \in post^b(p)\} \cup \{(p,t) \mid p \in \mathcal{P}_S^U \wedge t \in post^U(p)\}$$

and $\mathcal{M}_n = \{(p,i) \mid p \in \mathcal{P}^b \cup \mathcal{P}_S^U \wedge 1 \leq i \leq n\}$. The strategy $\mathcal{S}^b$ for the system players 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 strategy for the system players satisfies *justified refusal* because neither pure environment transitions can be disabled nor can transitions be differentiated due to the bounded unfolding. For the system players in the additional system places, each instance of a transition can be differentiated. The marking sequence $\mathcal{M}_n$ 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, i.e., at which simulation step a token is in the place. For the marking sequence, we include the additional system places.

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

$$\phi_n \overset{def.}{=} oneormore \Rightarrow \bigwedge_{1 \leq i < n} \Big(sequence_i \Rightarrow win_i\Big) \wedge \Big(sequence_n \Rightarrow win_n \wedge loop_n\Big)$$

$$oneormore \overset{def.}{=} \bigwedge_{p \in \mathcal{P}_S^U} \left(\bigvee_{t \in post^U(p)} (p,t) \right)$$

$$sequence_i \overset{def.}{=} initial \wedge \bigwedge_{1 \leq j < i} seqflow_j$$

$$initial \overset{def.}{=} \bigwedge_{p \in In^b \cup \mathcal{P}_S^U} (p,1) \wedge \bigwedge_{p \in \mathcal{P}^b \setminus In^b} \neg(p,1)$$

$$seqflow_i \overset{def.}{=} \bigvee_{t \in \mathcal{T}^b} \Bigg(\bigwedge_{p \in pre^b(t)} (p,i) \wedge \bigwedge_{p \in pre^b(t) \wedge \lambda^b(p) \in \mathcal{P}_S} (p, \lambda^b(t)) \wedge \bigwedge_{p \in pre^U(t)} \big((p,t) \wedge (p,i) \wedge (p,i+1)\big) \wedge \bigwedge_{p \in post^b(t)} (p,i+1) \wedge \bigwedge_{p \in pre^b(t) \setminus post^b(t)} \neg(p,i+1) \wedge \bigwedge_{p \in \mathcal{P}^b \setminus (pre^b(t) \cup post^b(t))} \big((p,i) \Leftrightarrow (p,i+1)\big) \Bigg)$$

Initially, it is ensured that the system players in the additional system places allow *one or more* of their outgoing transitions. For each simulation point $1 \leq i \leq n$, it is tested whether the variables in $\mathscr{M}_n$ represent a correct $sequence_i$ 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 end of the simulation is reached, then it is additionally tested that a $loop_n$ occurred. A correct *sequence* of markings starts from the *initial* marking, which includes the system players in the additional system places. It is followed by the *sequential flow* of $i-1$ enabled and by the strategy for the system players allowed transitions. The *sequential flow* of a transition from time point i requires all places in its precondition to contain a token and the system places as well as the additional system places of its precondition to allow the transition according to the strategy for the system players. Then, at $i+1$, the places of its postcondition are set to true, places in its precondition but not its postcondition are set to false, the system players in the additional system places stay in place, and all other places retain their truth value.

$$win_i \overset{def.}{=} nobadmarking_i \wedge deterministic_i \wedge \big(deadlock_i \Rightarrow terminating_i\big)$$

$$nobadmarking_i \overset{def.}{=} \bigwedge_{M \in \mathcal{M}_B} \Bigg(\bigvee_{p \in \mathcal{P}^b \wedge \lambda^b(p) \in M} \neg(p,i) \vee \bigvee_{p \in \mathcal{P}^b \wedge \lambda^b(p) \notin M} (p,i) \Bigg)$$

$$deterministic_i \overset{def.}{=} \bigwedge_{\substack{t_1,t_2 \in \mathcal{T}^b \wedge t_1 \neq t_2 \wedge \\ \lambda^b[pre^b(t_1)] \cap \lambda^b[pre^b(t_2)] \cap \mathcal{P}_S \neq \emptyset}} \Bigg(\bigvee_{p \in pre^b(t_1) \cup pre^b(t_2)} \neg(p,i) \vee \bigvee_{p_1 \in pre^b(t_1) \wedge \lambda^b(p_1) \in \mathcal{P}_S} \neg(p_1, \lambda^b(t_1)) \vee \bigvee_{p_2 \in pre^b(t_2) \wedge \lambda^b(p_2) \in \mathcal{P}_S} \neg(p_2, \lambda^b(t_2)) \vee$$

$$\left. \bigvee_{p_1 \in pre^U(t_1)} \neg(p_1, t_1) \vee \bigvee_{p_2 \in pre^U(t_2)} \neg(p_2, t_2) \right)$$

$$deadlock_i \stackrel{def.}{=} \bigwedge_{t \in \mathcal{T}^b} \left(\bigvee_{p \in pre^b(t)} \neg(p, i) \vee \bigvee_{p \in pre^b(t) \wedge \lambda^b(p) \in \mathcal{P}_S} \neg(p, \lambda^b(t)) \vee \bigvee_{p \in pre^U(t)} \neg(p, t) \right)$$

$$terminating_i \stackrel{def.}{=} \bigwedge_{t \in \mathcal{T}^b} \left(\bigvee_{p \in pre^b(t)} \neg(p, i) \right)$$

$$loop_n \stackrel{def.}{=} \bigvee_{1 \leq i_1 < i_2 \leq n} \left(\bigwedge_{p \in \mathcal{P}^b} \big((p, i_1) \Leftrightarrow (p, i_2) \big) \right)$$

The formula win_i tests whether the marking at position i fulfills the requirements to be winning. For the limit n on the simulation length, it is additionally tested that a $loop_n$ occurred. The play is winning if *no bad marking* is reached, the system makes only *deterministic* decisions, and each *deadlock* is caused by *termination*. No bad marking is reached, when, for each bad marking, at least one player in a place from the bad marking is missing or at least one player is additionally in a place not contained in the bad marking. The system makes only deterministic decisions when, for each pair of transitions with a shared system place, at least one of the two transitions is not enabled. Here, a transition may not be enabled either because a player in a place from its precondition is missing or because the strategy for the system players (including the system players in the additional system places) decides against allowing the transition.

A deadlock occurs when no transition is enabled including the choices of the strategy for the system players. The decisions for the system players in the additional system places are included. Meanwhile, termination occurs when no transition is enabled independently of the strategy for the system players. Therefore, $deadlock_i \Rightarrow terminating_i$ ensures that the system does not prevent the reaching of bad markings by stopping to fire transitions, but deadlocks are only allowed when the Petri game terminates. A $loop_n$ occurs when the same marking is repeated at two different simulation points. As the strategy for the system players has to be deterministic, its behavior can be repeated infinitely often in the loop such that the strategy for the system players is also winning in an infinite play. Remember that a loop is only required when the maximal simulation length n is reached even though the repetition can occur at earlier simulation points. In the finite case, $sequence_i$ is only true until the simulated Petri game terminates.

5.4.1. Simplifying and Extending the QBF Encoding

We report on notational corrections, simplifications, and extensions for the original QBF encoding of bounded synthesis for Petri games [Fin15].

Most notational corrections are caused by the fact that the set of system places in the finite unfolding $\mathcal{P}_S^b$ is not defined. Instead, we have to use the homomorphism λ^b to

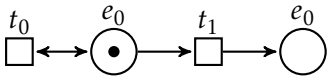


Figure 5.6.: A Petri game with only one environment player is depicted to illustrate why the QBF encoding has to use our definition of $loop_n$ instead of the original one. The winning condition of the Petri game is an empty set of bad markings. Therefore, the (empty set of) system players win this Petri game.

check whether a place is a system in the Petri game. Furthermore, in the definition of the formula to enforce $deterministic_i$ choices of the strategy for the system players, we need to differentiate between the places $p_1 \in pre^b(t_1) \wedge \lambda^b(p_1) \in \mathcal{P}_S$ and $p_2 \in pre^b(t_2) \wedge \lambda^b(p_2) \in \mathcal{P}_S$ (instead of having $p \in (pre^b(t_1) \cup pre^b(t_2)) \cap \mathcal{P}_S$ [Fin15]) because a transition can be deactivated by a system player in a system place that is only in the precondition of either t_1 or t_2 but not of both t_1 and t_2. In the last case, the original definition leads to either the undefined variable (p, t_1) or the undefined variable (p, t_2).

One can simplify the definition of $sequence_i = initial \wedge \neg deadlock_1 \wedge seqflow_1 \wedge \cdots \wedge \neg deadlock_{i-1} \wedge seqflow_{i-1}$ [Fin15][1] to $sequence_i = initial \wedge seqflow_1 \wedge \cdots \wedge seqflow_{i-1}$ because $seqflow_i \Rightarrow \neg deadlock_i$. When a transition fires from a marking, then this marking cannot deadlock. We implemented both versions and can show that the simplified definition is always beneficial for solving times, i.e., bounded synthesis finds a winning strategy for the system players quicker if such a strategy exists.

We present two extensions for the original QBF encoding: First, the formula to detect loops should be $loop_n = \bigvee_{1 \leq i_1 < i_2 \leq n} \Big(\bigwedge_{p \in \mathcal{P}^b} (p, i_1) \Leftrightarrow (p, i_2) \Big)$ (instead of $loop_n = \bigvee_{i \in \{1,\dots,n-1\}} \Big(\bigwedge_{p \in \mathcal{P}^b} (p, i) \Leftrightarrow (p, n) \Big)$ [Fin15]). For the Petri game from Figure 5.6, only the new definition of $loop_n$ determines that a winning strategy for the (empty set of) system players exists. With the old definition of $loop_n$, the sequence of markings corresponding to t_0 firing $n-2$ times followed by t_1 firing does not correspond to a loop because the marking at simulation length n is unique and a loop only occurred at the $n-1$ markings before. Because of this sequence of markings, the QBF encoding with the old definition of $loop_n$ (falsely) rejects the existence of a winning strategy. With the new correct definition of $loop_n$, this is no longer the case and a winning strategy for the system players is found.

Second, we introduce the system players in the additional system places from the bounded unfolding for the case that the underlying Petri net of the Petri game has loops in its graph of reachable markings. It is enforced by *oneormore* that these system players allow at least one of their outgoing transitions. Remember that the requirement for deterministic decisions by the system players ensure that these system players allow at most one of their outgoing transitions. Therefore, they allow exactly one of their outgoing transitions. The decisions of these system players are included when consid-

[1] We emphasize that transitions fire sequentially in the QBF encoding by writing $seqflow_i$ instead of $flow_i$ as we optimize the QBF encoding in Chapter 6 to fire transition true concurrently for bad places as winning condition.

ering the enabledness and firing of transitions. These system players do not need to be included when arguing about markings because they cannot leave their additional system places. With this encoding, we leave it in the hand of the set of the system players which b-bounded unfolding up to the given bound b to use instead of the unfolding algorithm from the previous section fixing this decision.

5.5. Encoding for Good Markings

We present the QBF encoding to find winning strategies for the system players in Petri games with good markings $\mathcal{M}_G$ as global winning condition. Notice that it is required that the Petri game does not have any transitions where the precondition of a transition equals the postcondition of this transition. The reason for this is that we have to exclude unfair firing sequences from the simulation of the Petri game. For this, we need to identify which transitions are fired, which is achieved by checking which precondition and postcondition of a transition match the change between two subsequent markings. If the precondition and the postcondition of a transition could be equal, then this would not be possible. We discuss later on how this small restriction can be circumvented. In the following, the QBF encoding follows the same structure as the QBF encoding for bad markings with a few changes. These changes are depicted in green and we explain the intuition behind the changes and why they are necessary.

$$
\begin{aligned}
\phi_n \stackrel{def.}{=}\ & \mathit{oneormore} \Rightarrow \\
& \bigwedge_{1\leq i<n} \Big(\mathit{sequence}_i \Rightarrow \big(\mathit{win}_i \wedge (\neg \mathit{sequence}_{i+1} \Rightarrow \mathit{goodmarking}_i) \big) \Big) \wedge \\
& \big((\mathit{sequence}_n \wedge \neg \mathit{unfair}_n) \Rightarrow (\mathit{win}_n \wedge \mathit{loop}_n \wedge \mathit{goodmarking}_n) \big)
\end{aligned}
$$

$$
\mathit{oneormore} \stackrel{def.}{=} \bigwedge_{p \in \mathcal{P}_S^U} \Big(\bigvee_{t \in post^U(p)} (p,t) \Big)
$$

$$
\mathit{sequence}_i \stackrel{def.}{=} \mathit{initial} \wedge \bigwedge_{1\leq j<i} \mathit{seqflow}_j
$$

$$
\mathit{initial} \stackrel{def.}{=} \bigwedge_{p \in In^b \cup \mathcal{P}_S^U} (p,1) \wedge \bigwedge_{p \in \mathcal{P}^b \setminus In^b} \neg(p,1)
$$

$$
\mathit{seqflow}_i \stackrel{def.}{=} \bigvee_{t \in \mathcal{T}^b} \mathit{fire}_i(t)
$$

$$
\mathit{fire}_i(t) \stackrel{def.}{=} \bigvee_{t \in \mathcal{T}^b} \Big(\bigwedge_{p \in pre^b(t)} (p,i) \wedge \bigwedge_{p \in pre^b(t) \wedge \lambda^b(p) \in \mathcal{P}_S} (p, \lambda^b(t)) \wedge
$$

$$\bigwedge_{p \in pre^U(t)} \Big((p,t) \wedge (p,i) \wedge (p,i+1)\Big) \wedge \bigwedge_{p \in post^b(t)} (p,i+1) \wedge$$

$$\bigwedge_{p \in pre^b(t) \setminus post^b(t)} \neg(p,i+1) \wedge \bigwedge_{p \in \mathcal{P}^b \setminus (pre^b(t) \cup post^b(t))} \Big((p,i) \Leftrightarrow (p,i+1)\Big)\Big)$$

$$unfair_n \stackrel{def.}{=} \bigvee_{1 \le i < j < n \wedge j - i \ge 2} \Big(\bigwedge_{p \in \mathcal{P}^b} (p,i) \Leftrightarrow (p,j)\Big) \wedge$$

$$\bigvee_{t \in \mathcal{T}^b} \Big(\bigwedge_{p \in pre^b(t) \wedge \lambda^b(p) \in \mathcal{P}_S} (p, \lambda^b(t)) \wedge \bigwedge_{p \in pre^U(t)} (p,t) \wedge$$

$$\bigwedge_{\substack{i \le k < j \wedge \\ p \in pre^b(t)}} \Big((p,k) \wedge \bigwedge_{t' \in post^b(p)} \neg fire_k(t')\Big)\Big)$$

$$goodmarking_i \stackrel{def.}{=} \bigvee_{1 \le j \le i \wedge M \in \mathcal{M}_G} \Big(\bigvee_{p \in \mathcal{P}^b \wedge \lambda^b(p) \in M} (p,j) \wedge \bigwedge_{p \in \mathcal{P}^b \wedge \lambda^b(p) \notin M} \neg(p,j)\Big)$$

There are two changes: We have to check that a good marking is reached instead of checking that no bad marking is reached and we need to identify unfair schedulings and to ignore these unfair schedulings. For the first change, we replace $nobadmarking_i$ by $goodmarking_n$. The check for bad markings occurs in win_i because, for all positions i, no bad marking should be reached. By contrast, the check for good markings ensures that at least once a good marking is reached at some position j. This check is positioned next to the check for a loop. Furthermore, if the Petri game terminates before reaching the maximal simulation length, i.e., $sequence_i \wedge \neg sequence_{i+1}$ holds, then a good marking already has to occur up to position i. A good marking is reached when, for each place in it, one of the copies for the place has a token and other places not in the good marking do not contain tokens. Notice that, by the definition of unfolding, at most one of the copies due to the unfolding of a place contains a token.

For the second change, we introduce $unfair_n$ to identify markings corresponding to sequences of transitions in which an enabled and by the strategy for the system players allowed transition never fires. Such transitions never firing can prevent the reaching of a good marking. We identify unfair loops by searching for loops that repeat a marking at points i and j and where a transition is allowed by the strategy for the system players and enabled at all positions from i to j but never fires. This is the reason for the restriction that no transition with the same precondition and postcondition can exist in the Petri game because a transition firing is identified by the change of tokens from the precondition to the postcondition. This restriction could be circumvented by adding additional variables to identify which transition fires at position i but a prototype implementation showed a considerable performance decrease for this approach. Furthermore, we rarely needed transitions with the same precondition and postcondi-

tion when modeling problems as Petri games with good markings.

$$win_i \stackrel{def.}{=} deterministic_i \wedge \Big(deadlock_i \Rightarrow terminating_i\Big)$$

$$deterministic_i \stackrel{def.}{=} \bigwedge_{\substack{t_1,t_2\in\mathcal{T}^b\wedge t_1\neq t_2\wedge\\ \lambda^b[pre^b(t_1)]\cap\lambda^b[pre^b(t_2)]\cap\mathcal{P}_S\neq\emptyset}} \Bigg(\bigvee_{p\in pre^b(t_1)\cup pre^b(t_2)} \neg(p,i) \vee \bigvee_{p_1\in pre^b(t_1)\wedge\lambda^b(p_1)\in\mathcal{P}_S} \neg(p_1,\lambda^b(t_1)) \vee \bigvee_{p_2\in pre^b(t_2)\wedge\lambda^b(p_2)\in\mathcal{P}_S} \neg(p_2,\lambda^b(t_2)) \vee \bigvee_{p_1\in pre^U(t_1)} \neg(p_1,t_1) \vee \bigvee_{p_2\in pre^U(t_2)} \neg(p_2,t_2)\Bigg)$$

$$deadlock_i \stackrel{def.}{=} \bigwedge_{t\in\mathcal{T}^b}\Bigg(\bigvee_{p\in pre^b(t)} \neg(p,i) \vee \bigvee_{p\in pre^b(t)\wedge\lambda^b(p)\in\mathcal{P}_S} \neg(p,\lambda^b(t)) \vee \bigvee_{p\in pre^U(t)} \neg(p,t)\Bigg)$$

$$terminating_i \stackrel{def.}{=} \bigwedge_{t\in\mathcal{T}^b}\Bigg(\bigvee_{p\in pre^b(t)} \neg(p,i)\Bigg)$$

$$loop_n \stackrel{def.}{=} \bigvee_{1\leq i_1<i_2\leq n}\Bigg(\bigwedge_{p\in\mathcal{P}^b} \Big((p,i_1)\Leftrightarrow(p,i_2)\Big)\Bigg)$$

The remaining parts of the encoding to identify winning markings and the repetition of markings to find loops stay the same.

We can extend the above encoding to good and bad markings $(\mathcal{M}_G,\mathcal{M}_B)$ as winning condition by including the avoidance of bad markings until a good marking is reached in the definition of $goodmarking_i$. We depict the simple extension in green.

$$goodandbadmarking_i \stackrel{def.}{=} \bigvee_{1\leq j\leq i\wedge M\in\mathcal{M}_G}\Bigg(\bigwedge_{1\leq k<j} nobadmarking_k\wedge \bigvee_{p\in\mathcal{P}^b\wedge\lambda^b(p)\in M}(p,j)\wedge \bigwedge_{p\in\mathcal{P}^b\wedge\lambda^b(p)\notin M}\neg(p,j)\Bigg)$$

$$nobadmarking_i \stackrel{def.}{=} \bigwedge_{M\in\mathcal{M}_B}\Bigg(\bigvee_{p\in\mathcal{P}^b\wedge\lambda^b(p)\in M}\neg(p,i)\vee \bigvee_{p\in\mathcal{P}^b\wedge\lambda^b(p)\notin M}(p,i)\Bigg)$$

The avoidance of bad markings is defined as in Section 5.4.

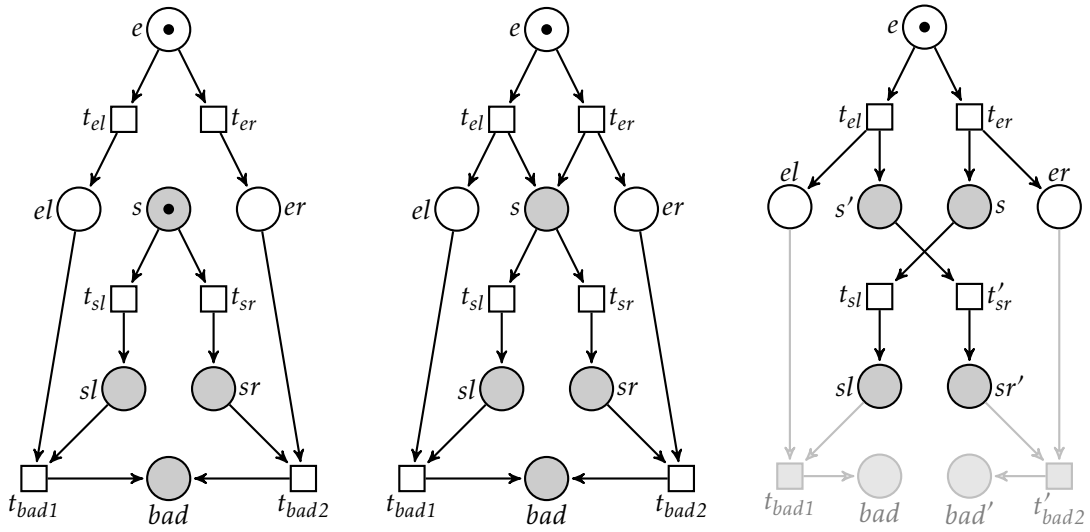
5.6. Obtaining Winning Bounded Strategies

Our implementation uses the QBF solver QuAbS [HT18; Ten19] because of its outstanding performance. We outline how the certification capabilities of QuAbS can be used for the analysis of unrealizable Petri games and for the construction of implementations from winning strategies. The QBF encodings presented before are particularly challenging for QBF solvers, that expect the QBF in conjunctive normal form (CNF). The QBF needs to be brought into CNF by an appropriate translation [Tse83; PG86] before such QBF solvers can be used, which results in a considerable blow-up in terms of the size of the QBF. Therefore, hardly any instance can be solved by such QBF solvers, even with enabled preprocessing and independent of the used solver. By contrast, non-CNF solvers scale much better with QuAbS performing best overall [HT18].

We consider the exemplary Petri game from Figure 5.7a where the system player and the environment player have both a left transition and a right transition. The bad marking $\{bad\}$ can be reached when either both players decide for their left transitions or both players decide for their right transitions. In Figure 5.7a, the choice of the system player and the environment player are *concurrent*, i.e., the system player has no strategy to avoid the bad marking $\{bad\}$: choosing either only the left transition t_{sl} or only the right transition t_{sr}, the environment player will do the same, leading the Petri game to the bad marking $\{bad\}$. All other strategies for the system player, especially strategies not allowing transition t_{bad1} or transition t_{bad2}, lead to deadlocks without termination or to nondeterministic decisions by the system players.

As the Petri game in Figure 5.7a has no winning strategy, the QBF encoding is unsatisfiable and QuAbS returns a certificate for the universal player representing the universal quantification in the QBF. This certificate represents a specific flow of tokens leading to a negative situation for *every* possible strategy for the system player represented by the existential quantification. When the system player only decides to enable transition t_{sl} and to not enable transition t_{sr}, then the counterexample moves the environment player from place e to place el, the system player from place s to place sl, and afterward fires the transition to reach the bad marking $\{bad\}$. An analog counterexample is returned when the system player enables transition t_{sr} and does not enable transition t_{sl}. When the system player enables neither transition, then the counterexample moves the environment player from place e to place el and then reaches a deadlock without termination. This situation is forbidden for strategies for the system players as otherwise the winning condition of avoiding bad markings would be a trivial. When the system player activates both transitions, then already the initial marking constitutes a counterexample as the decision of the system player is nondeterministic.

From these counterexamples, we can derive that we have to introduce communication between the system player and the environment player. The easiest way to do so is given in Figure 5.7b where the system player is created with the decision of the environment player and then can only afterward react to the decision of the environment player. The different causal memory of the system player in place s depending on whether transition t_{el} or transition t_{er} was fired results in the unfolding of place s.



(a) A Petri game where the system player should not mimic the behavior of the environment player but no communication takes place prior to the decision of the system player.

(b) The environment player forwards its decision to the system player and afterward the system player should not mimic this decision.

(c) A winning strategy where the system player does not mimic the environment player such that transitions to the bad marking $\{bad\}$ become unreachable.

Figure 5.7.: An example workflow of designing a Petri game is outlined. QuAbS produces counterexamples to any strategy in the left Petri game. From there, it becomes clear that there is no information exchange between the system player and the environment player. Therefore, the design of the Petri game is changed to the one in the middle where the environment player leaks its decision to the system player. For this game, we can extract the winning strategy on the right using QuAbS which avoids the bad marking $\{bad\}$ as the system player answers with opposite decisions to the decisions of the environment player.

Then, a winning strategy exists where the system player makes a different decision than the previous decision by the environment player. A corresponding winning strategy is depicted in Figure 5.7c. In the winning strategy, the duplication of places is indicated by the prime symbol $'$. The greyed-out parts are included in the winning strategy in order to highlight the transitions to the bad place, which become unreachable by the choices of the winning strategy. The winning strategy is based on a bounded unfolding, which includes, at each of the two places s and s', the two outgoing transitions with the corresponding following places and transitions.

For satisfiable QBFs, the QBF solver QuAbS returns an evaluation of the variables in $\mathcal{S}^b$. For each pair of a system place p in the bounded unfolding and a transition t in the postcondition of p in the original Petri game with a negative evaluation, we remove all transitions in the postcondition of p in the bounded unfolding that refer to t according to the homomorphism from the bounded unfolding to the original game and

their flows. Transitions, flows, and places that become thereby unreachable are also removed. For checking the reachability, we include the decisions by the system players in the additional system places. As last step, we also remove all additional system places and flows including the additional system places. Notice that this last step does not remove the transitions chosen by the system players in the additional system places to represent causal history. The satisfying assignment of QuAbS for the QBF encoding of bounded synthesis for Petri games allows to directly remove not activated transitions (t_{sr} and t'_{sl}) and their resulting unreachable parts of the game, making all remaining transitions to the bad marking $\{bad\}$ unreachable (indicated greyed-out in Figure 5.7c).

5.7. Summary

We presented bounded synthesis for the global winning conditions of bad markings and of good markings in safe Petri games. We outlined the concepts of bounded unfoldings and of bounded strategies. Furthermore, we described in detail the generation of bounded unfoldings and the QBF encoding for the existence of a winning strategy for a bounded unfolding. For bad markings, good markings, as well as good and bad markings, the results of this section allow us to find winning strategies for the system players in safe Petri games with a bounded number of players arbitrarily distributed between system and environment players.

Chapter 6

True Concurrent Encoding for Bounded Synthesis

We present the new true concurrent encoding for bounded synthesis of Petri games. As discussed in Chapter 5, bounded synthesis for Petri games automatically derives an implementation satisfying a specification of a distributed system if such an implementation exists [Fin15]. In Chapter 5, bounded synthesis for Petri games did not utilize the asynchronous nature of Petri games. Instead, concurrent behavior of components was encoded by all interleavings to be checked against the specification. This was necessary to encode the global winning conditions of bad markings and of good markings.

In this chapter, we identify true concurrency in the synthesis of asynchronous distributed systems represented as Petri games with the local winning condition of bad places. True concurrency 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. Preliminary work on the true concurrent encoding has been published in [Met17], which has been significantly improved and expanded in this chapter. This chapter has also been published in [HM19b].

Bounded synthesis [FS13] incrementally increases the memory of possible strategies for the system players until a winning strategy is found. Each iteration of the bounded synthesis algorithm for Petri games [Fin15] checks the existence of a winning strategy for the system players with bounded memory by simulating the resulting Petri game. This simulation is represented by all *interleavings* of fired transitions allowed by possible strategies for the system players. For two concurrent transitions, it makes no difference whether one transition or the other transition is scheduled first. It suffices to only check one scheduling where both transitions 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 *strategies for the environment players* for Petri games which explicitly represent the decisions of environment players. Strategies for the environment players restrict a given strategy for the system players and try to reach markings which prove the strategy for the system players to *not* be winning. We present how the explicit environment decisions of strategies for the environment players 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 players. For bounded synthesis, we encode the assumptions on strategies for the system players and strategies for the environment players 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 QBF encoding* on an extended set of benchmarks. Our implementations are part of the Adam toolkit [FGO15], open-source, and available online [Ada20]. Our experimental results show that the true concurrent QBF encoding outperforms the sequential encoding by a considerable margin.

The key contributions of this chapter 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 *strategies for the environment players* of Petri games.
- We prove that strategies for the environment players *preserve the existence of winning strategies for the system players* and encode them as QBFs for bounded synthesis for Petri games.
- We *implement* the true concurrent QBF encoding and show considerable improvements against the sequential encoding on an extended benchmark set.

This chapter is structured as follows: In Section 6.1, we illustrate the benefits of true concurrent scheduling for bounded synthesis for Petri games with an example. In Section 6.3, we introduce strategies for the environment players and prove that they preserve the existence of winning strategies. Section 6.4 formally presents the true concurrent QBF encoding. Section 6.5 surveys experimental results for the implementation of the true concurrent QBF encoding.

6.1. Motivating Example

Figure 6.1 illustrates how true concurrency simplifies bounded synthesis for Petri games. This figure has been used in Figure 2.3 of Chapter 2 to illustrate Petri games. Now, we focus on the concurrency between system players in the game. The Petri game from Figure 6.1 specifies a production line for repairing a product. The different possible requirements for repair are modeled as choices of the environment player. The product can either require repair by a single robot or by both robots concurrently.

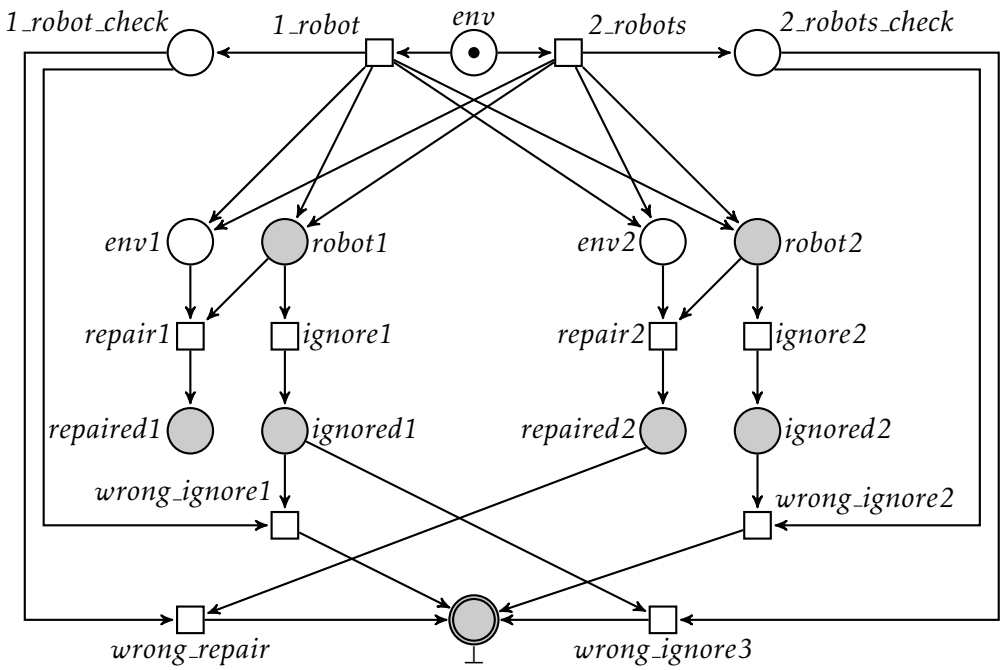


Figure 6.1.: This Petri game specifies a production line where two robots can repair a product. The product either requires repair by only one or by both robots.

These robots are represented by system players and have to collectively meet the requirement of the product. Places belonging to the environment player represent the product and its requirements for repair.

The winning objective of the game is represented by the bad place $\bot$ which is depicted as a double circle. The system players have to avoid reaching this place for all choices of the environment player. Based on its causal past, a system player can decide which outgoing 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 player has chosen *1_robot*, then no repair by the second robot is allowed whereas if the environment player has chosen *2_robots*, then repair by the second robot is required.

A bounded unfolding and the corresponding winning bounded strategy for the system players are presented in Figure 6.2 where primed places and transitions result from different causal pasts. This figure has been used in Figure 2.4 of Chapter 2 to illustrate unfoldings and strategies. Because the Petri game in Figure 6.1 does not have loops in the graph of reachable markings of its underlying Petri net, the bounded unfolding and the corresponding winning bounded strategy are equal to the general unfolding and a corresponding winning strategy. Including the grayed-out parts results in a bounded unfolding whereas leaving out the grayed-out parts results in a winning bounded strat-

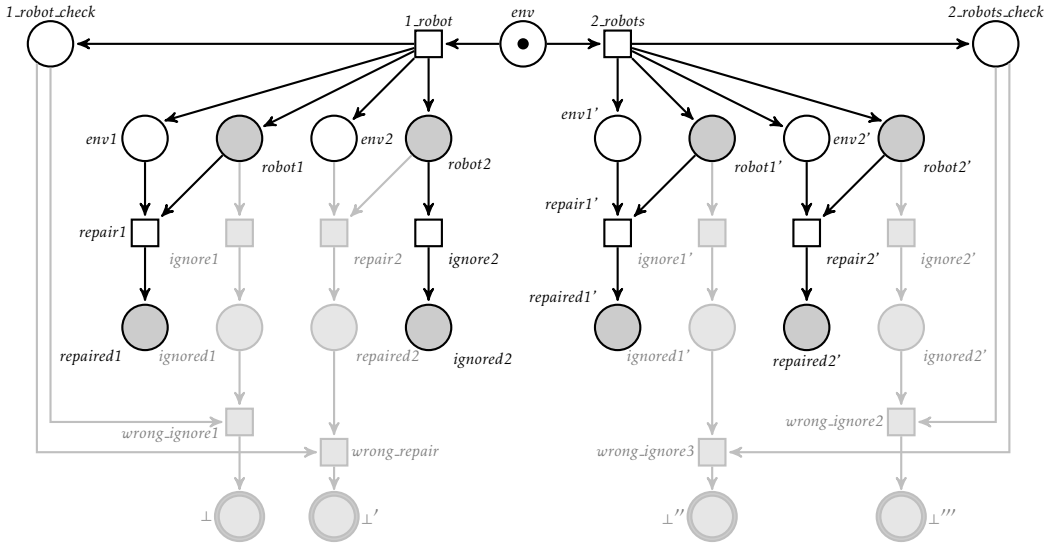


Figure 6.2.: When excluding the grayed-out parts, a winning (bounded) strategy for the system players is presented for the Petri game from Figure 6.1, which specifies a production line with two robots. The system players make different decisions depending on the choice of the environment player. Transitions which cannot be enabled and unreachable places are grayed-out. Including the grayed-out parts results in the (bounded) unfolding of the Petri game from Figure 6.1.

egy. In the winning strategy, 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 strategy for the system players.

In the QBF encoding the existence of a winning strategy for the system players for a given memory bound and a corresponding bounded unfolding, the sequential QBF encoding tests all possible interleavings of transitions This implies that, in the motivating example, first the environment player 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. The 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 strategy for the system players of Figure 6.2.

6.2. Strategies for the System Players

We recall the formal definition of a strategy for the system in player in a Petri game $\mathcal{G}$ because we use them intensively in the following when defining strategies for the environment player. As we will introduce strategies for the environment players later, we

rename *justified refusal* to *system refusal*. For readability, we introduce the abbreviations (S1), (S2), and (S3) for the three requirements *deterministic choice*, *system refusal*, and *deadlock-avoidance*. Because we focus on bad places as winning condition in this chapter, the requirement of deadlock-avoidance from the definition of bad places as winning condition becomes one of the three requirements of the strategy for the system players.

Definition 53 (Strategy for the system players)
A *strategy for the system players* in a Petri game $\mathcal{G}$ with bad places $\mathcal{P}_B$ as winning condition is a subprocess $\sigma = (\mathcal{N}^\sigma, \lambda^\sigma)$ of the unfolding $\iota_U = (\mathcal{N}^U, \lambda^U)$ of $\mathcal{G}$ where system places can remove outgoing transitions such that the following requirements hold:

(S1) *Deterministic choice*:
$\forall M \in \mathcal{R}(\mathcal{N}^\sigma) : \forall p \in M \cap \mathcal{P}_S^\sigma : \exists^{\leq 1} t \in post^\sigma(p) : pre^\sigma(t) \subseteq M$

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

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

Deterministic choice 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 strategy for the system players to enable at least one transition for each reachable marking as long as one transition is enabled in the unfolding. For a Petri game $\mathcal{G} = (\mathcal{P}_S, \mathcal{P}_E, \mathcal{T}, \mathcal{F}, In, \mathcal{W})$ with the local winning condition of bad places $\mathcal{W} = \mathcal{P}_B$, a strategy for the system players is winning if all reachable markings do not contain any bad places from $\mathcal{P}_B$, i.e., $\forall M \in \mathcal{R}(\mathcal{N}^\sigma) : \lambda^\sigma[M] \cap \mathcal{P}_B = \emptyset$.

6.3. True Concurrency in Petri Games

In this section, we define true concurrency in Petri games. Therefore, we first formalize *strategies for the environment players* to explicitly represent environment decisions in response to a given strategy for the system players. 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.

6.3.1. Strategy for the Environment Players

Strategies for the system players represent the system's restrictions of enabled transitions but purely environmental transitions remain uncontrollable. Therefore, a strategy for the system players can result in different fired transitions based on decisions

by the environment players. We introduce *strategies for the environment players* to explicitly represent decisions of environment players and to obtain a unique sequence of fired transitions up to reordering of concurrent transitions, i.e., transitions with a disjoint precondition.

Definition 54 (Strategy for the environment players)
A *strategy for the environment players* $\gamma = (\mathcal{N}^\gamma, \lambda^\gamma)$ is a subprocess of a strategy for the system players $\sigma = (\mathcal{N}^\sigma, \lambda^\sigma)$ (which is a subprocess of the unfolding $\iota_U = (\mathcal{N}^U, \lambda^U)$ of the given Petri game $\mathcal{G}$ with bad places $\mathcal{P}_B$ as winning condition) 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 strategies for the environment players to only remove transitions with at least one environment place in their precondition. *Progress* requires the strategy for the environment players to enable at least one transition for each reachable marking as long as a transition is enabled in the underlying strategy for the system players.

Notice that a strategy for the system players can be obtained again by taking the union of all possible strategies for the environment players. Next, we show that strategies for the environment players resolve the remaining conflicts of a strategy for the system players for a Petri game:

Lemma 15. *A strategy γ for the environment players leads to a unique sequence of fired transitions up to reordering of concurrent transitions, i.e., $\forall p \in \mathcal{P}^\gamma : |post^\gamma(p)| \leq 1$.*

Proof. A strategy σ for the system players satisfies for all system places $p \in \mathcal{P}_S^\sigma$ either the condition $|post^\sigma(p)| \leq 1$ or the nondeterminism in the choice of the successor transition is resolved by the strategy γ for the environment players. Since the strategy for the environment players 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 strategies for the environment players. 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 concurrent transitions. □

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

to (S2), again, to enable unique decision, and requirement (E3) is (S3) lifted directly to strategies for the environment players.

Notice that there is subtle difference between a strategy for the environment players and a maximal play of a strategy for the system players. We first recall the definition of a maximal play from Chapter 2 and then explain the similarities and differences in detail. A *play* $\pi = (\mathcal{N}^\pi, \lambda^\pi)$ is a subprocess of a strategy (for the system players) $\sigma = (\mathcal{N}^\sigma, \lambda^\sigma)$ such that $\forall p \in \mathcal{P}^\pi : |post(p)| \leq 1$. A play is *maximal* when, for each set of pairwise concurrent places C in $\mathcal{N}^\pi$ such that $C = pre^\sigma(t)$ for some transition $t \in \mathcal{T}^\sigma$, a place $p \in C$ and a transition $t' \in \mathcal{T}^\pi$ exist such that $t' \in post^\pi(p)$.

Both strategies for the environment players and maximal plays of strategies for the system players resolve the remaining conflicts in the strategy for the system players (cf. definition of plays and Lemma 15). The definitions of strategies for the environment players and maximal plays of a strategies for the system players differ in how they define maximality. According to the definition of a maximal play, transitions have to be added to each set of pairwise concurrent places, i.e., the progress of the play is defined locally for each possible set of players. Meanwhile, in a strategy for the environment players, the progress is defined globally for reachable markings (cf. requirement (E3)). Together with requirement (E2), it is ensured that the strategy for the environment players can only remove transitions in which an environment player is participating.

On the one hand, this difference between strategies for the environment players and maximal plays of strategies for the system players is negligible for safety winning conditions such as bad places, because the environment players have the goal to reach the negative situations. On the other hand, using strategies for the environment player instead of maximal plays makes it easier to utilize true concurrent firing in the QBF encoding later on.

We introduce some notation to identify a strategy for the environment players based on a strategy for the system players and to identify a strategy for the system players based on an unfolding. By $\gamma \sqsubseteq_E \sigma$, we denote a strategy γ for the environment players as subprocess of a strategy σ for the system players subject to requirements (E1) to (E3). By $\sigma \sqsubseteq_S \iota_U$, we denote a strategy σ for the system players as a subprocess of the unfolding ι_U subject to requirements (S1) to (S3).

A strategy γ for the environment players is *winning* for the environment players (and a *counterexample* to the strategy σ for the system players being winning) if it reaches a bad place. We define a strategy for the system players to be *winning* against all strategies for the environment players: A strategy σ for the system players is winning if no bad places are reached for all strategies for the environment players, i.e., if the formula $\forall \gamma \sqsubseteq_E \sigma : \forall M \in \mathcal{R}(\mathcal{N}^\gamma) : \lambda^\gamma[M] \cap \mathcal{P}_B = \emptyset$ holds. Notice that a strategy for the system players fulfills requirements (S1) to (S3) by definition.

Example 6.3.1. Figure 6.3 shows a winning strategy for the environment players for a strategy for the system players of our running example with the bad place $\bot$. By the initial decision for transition *1_robot* by the strategy for the environment players, the right side of the strategy for the system players becomes unreachable. The system chooses the transitions *repair1* and *repair2* in response to transition transition *1_robot* being cho-

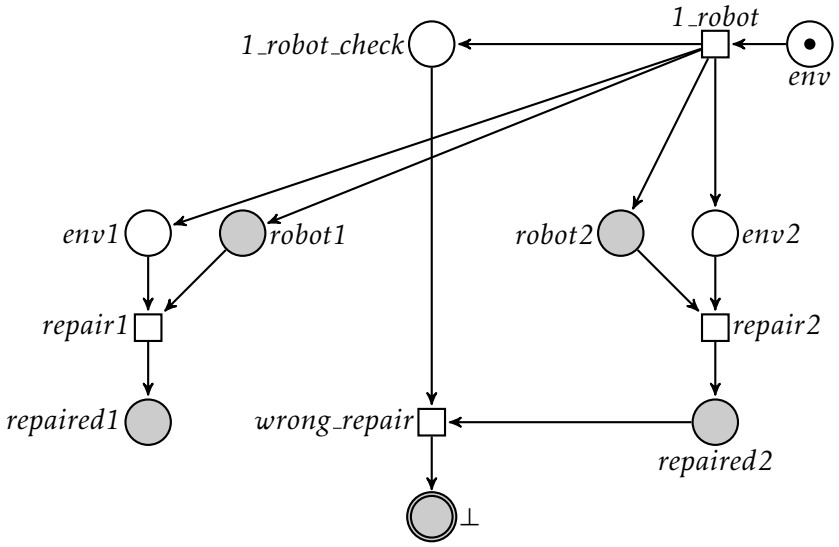


Figure 6.3.: A winning strategy for the environment players for a strategy for the system players for the Petri game specifying a production line from Figure 6.1 is depicted.

sen by the strategy for the environment players. By choosing transition *1_robot*, the second robot should have ignored the product. The strategy for the system players has to enable transition *wrong_repair* to avoid a deadlock and the strategy for the environment players agrees on firing it to reach the bad place. △

6.3.2. 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. By using $\uplus$ instead of $\cup$, we highlight that the union is disjoint by definition, i.e., we only write $A \uplus B$ when A and B are disjoint and then define it as the usual union. For the marking M and the set of enabled, conflict-free transitions $T = \{t_1, \ldots, t_n\}$, the successor marking M' is defined as $M[T\rangle M'$, where all transitions in T are enabled in M, i.e., $pre^{\mathcal{N}}(t_1) \uplus \ldots \uplus pre^{\mathcal{N}}(t_n) \subseteq M$, and the successor marking obtained by firing all transitions at the same time, i.e.,

$$M' = (M \setminus (pre^{\mathcal{N}}(t_1) \uplus \ldots \uplus pre^{\mathcal{N}}(t_n))) \uplus post^{\mathcal{N}}(t_1) \uplus \ldots \uplus post^{\mathcal{N}}(t_n).$$

The union of the preconditions of the transitions from T is disjoint because they are conflict-free. The union of the postconditions of the transitions from T is disjoint because the underlying Petri net of the given Petri game is safe, i.e., allows at most one token in every place of every reachable marking. The set of reachable markings according to the true concurrent flow semantics is defined as

$$\mathcal{R}^{tc}(\mathcal{N}) \quad = \quad \{M \subseteq \mathcal{P} \mid \exists \text{ maximal } T_1, \ldots, T_n \subseteq \mathcal{T} : \exists M_1, \ldots, M_n \subseteq \mathcal{P} :$$

$$In[T_1\rangle M_1[T_2\rangle \dots [T_n\rangle M_n = M\} \qquad \text{where } |T_1|,\dots,|T_n| > 0.$$

A set is *maximal* when adding each possible new element violates the conditions imposed on the set.

We denote the set of reachable markings in the sequential flow semantics by the set of reachable markings, i.e., $\mathcal{R}^{seq}(\mathcal{N}) = \mathcal{R}(\mathcal{N})$. 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 16. *There exists a winning strategy for the system players of a Petri game under the sequential flow semantics if and only if there exists a winning strategy for the system players of a Petri game under the true concurrent flow semantics.*

Proof. We show that

(1) using or not using strategies for the environment players agrees on reaching bad places for strategies for the system places in the sequential flow semantics, i.e.,
$\exists \sigma \sqsubseteq_S \iota_U : \forall M \in \mathcal{R}^{seq}(\mathcal{N}^\sigma) : \lambda^\sigma[M] \cap \mathcal{P}_B = \emptyset \Leftrightarrow$
$\exists \sigma \sqsubseteq_S \iota_U : \forall \gamma \sqsubseteq_E \sigma : \forall M \in \mathcal{R}^{seq}(\mathcal{N}^\gamma) : \lambda^\gamma[M] \cap \mathcal{P}_B = \emptyset$, and that

(2) the sequential flow semantics and the true concurrent flow semantics agree on reaching bad places when using strategies for the environment players, i.e.,
$\exists \sigma \sqsubseteq_S \iota_U : \forall \gamma \sqsubseteq_E \sigma : \forall M \in \mathcal{R}^{seq}(\mathcal{N}^\gamma) : \lambda^\gamma[M] \cap \mathcal{P}_B = \emptyset \Leftrightarrow$
$\exists \sigma \sqsubseteq_S \iota_U : \forall \gamma \sqsubseteq_E \sigma : \forall M \in \mathcal{R}^{tc}(\mathcal{N}^\gamma) : \lambda^\gamma[M] \cap \mathcal{P}_B = \emptyset$.

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

6.4. True Concurrent Encoding of Bounded Synthesis

We show how the requirements (E1) to (E3) on strategies for the environment players and the true concurrent flow semantics can be encoded as QBF. We introduce *stalling* of transitions to let environment players find nondeterministic decisions in a strategy for the system players. 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.

6.4.1. Stalling of Transitions to Find Nondeterministic Decisions

To use the true concurrent flow semantics in bounded synthesis for Petri games, we ensure that all possible strategies for the system players fulfill the requirements (S1) to (S3) and do not reach any bad place. In the formal definition, a strategy for the

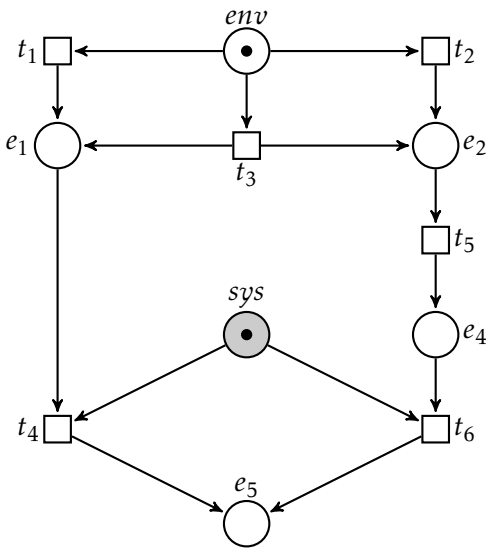


Figure 6.4.: A Petri game is shown that has no strategy for the system players as the four possible decisions at system place *sys* result in a deadlock when no or one transition is allowed or in nondeterminism when both transitions are allowed.

system players fulfills requirements (S1) to (S3) by definition. In the QBF encoding to search for strategies for the system players, we need to ensure that a potential strategy for the system players fulfills requirements (S1) to (S3). The requirement of deterministic choice 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 6.4 shows a new Petri game without bad places. It cannot be won by the system players because transitions t_4 and t_6 have to be enabled to be deadlock-avoiding and there is a marking where both transitions are enabled resulting in a nondeterministic decision. This contradicts requirement (S1) but in the true concurrent flow semantics, transition t_4 will always be fired before transition t_6 such that the marking with a nondeterministic decision of the system player is never reached. To check the requirements for strategies for the system players in the true concurrent QBF encoding when doing bounded synthesis, environment players can *stall* transitions with at least one system place in their precondition globally to catch up with the system.

The stalling of a transition means that the transition can never be fired in this simulation of the Petri game. This stalling makes it easier for the system players to win the Petri game because bad places constitute a safety winning condition. The requirement *deterministic choice* (S1) can only be violated at system places. In Figure 6.4, the firing of transition t_4 needs to be stalled such that transition t_5 is fired to prove that a potential strategy for the system players enabling both transitions is nondeterministic. Notice that stalling only concerns the firing of transitions but not their enabledness when checking for deterministic choices.

Notice further that the stalling of transitions does *not* occur in Theorem 16 because a strategy for the system players fulfills requirement (S1) for deterministic choice by definition. The stalling of transitions is only necessary when searching for strategies for the system players in the QBF encoding in order to ensure that the system players make deterministic choices.

6.4.2. Encoding True Concurrency as QBF

We extend the sequential QBF encoding of bounded synthesis for Petri games [Fin15] to strategies for the environment players with stalling and the true concurrent flow semantics. In the following, we highlight extensions of the QBF encoding in green. We utilize bad places $\mathcal{P}_B$ as winning condition of the Petri game $\mathcal{G}$ for which we search for a winning strategy for the system players. As in Chapter 5, a finite unfolding $\iota_U^b = (\mathcal{N}^b, \lambda^b)$ is obtained either by Algorithm 1 or by Algorithm 2 depending on whether the graph of reachable markings of the underlying Petri net $\mathcal{N}$ of $\mathcal{G}$ does or does not have loops. In the case of loops in the graph of reachable markings, we obtain the additional system places $\mathcal{P}_S^U$ with a corresponding flow relation $\mathcal{F}_S^U$. With $pre^U(p)$ and $post^U(p)$, we identify the precondition and the postcondition according to $\mathcal{F}_S^U$ for places $p \in \mathcal{P}_S^U$. In the case of no loops in the graph of reachable markings, the additional system places $\mathcal{P}_S^U$ are an empty set with an empty corresponding flow relation $\mathcal{F}_S^U$.

The strategy of the environment players is translated into additional universally quantified variables. The QBF-formula is $\exists \mathscr{S}^b : \forall \mathscr{M}_n : \forall \mathscr{E}^b : \phi_n$. The variables $\mathscr{S}^b$ for the strategy for the system players and for the sequences of markings $\mathscr{M}_n$ are defined as in Chapter 5 by

$$\mathscr{S}^b = \{(p, \lambda^b(t)) \mid p \in \mathcal{P}^b \wedge \lambda^b(p) \in \mathcal{P}_S \wedge t \in post^b(p)\} \cup \{(p,t) \mid p \in \mathcal{P}_S^U \wedge t \in post^U(p)\}$$

and

$$\mathscr{M}_n = \{(p,i) \mid p \in \mathcal{P}^b \cup \mathcal{P}_S^U \wedge 1 \leq i \leq n\}.$$

The variables $\mathscr{E}^b$ are defined 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 precondition to stall their progress, i.e.,

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

This encoding preserves the requirement of *environment refusal* (E2).

Bounded unfoldings may contain loops. The variables for the environment players 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. We apply the requirement *explicit choice* (E1) of the strategy for the environment players to ϕ_n and encode it in *choice*:

$$\phi_n \overset{def.}{=} oneormore \wedge choice \Rightarrow \bigwedge_{1 \leq i < n} \Big(sequence_i \Rightarrow win_i \Big) \wedge \Big(sequence_n \Rightarrow win_n \wedge loop_n \Big)$$

$$oneormore \overset{def.}{=} \bigwedge_{p \in \mathcal{P}_S^U} \Big(\bigvee_{t \in post^U(p)} (p,t) \Big)$$

$$choice \overset{def.}{=} \bigwedge_{p \in \mathcal{P}^b \wedge \lambda^b(p) \in \mathcal{P}_E \wedge 1 \leq i < n} \Big(\bigvee_{t \in post^b(p)} \Big((p,t,i) \wedge \bigwedge_{t' \in post^b(p) \setminus \{t\}} \neg(p,t',i) \Big) \Big)$$

$$sequence_i \overset{def.}{=} initial \wedge \bigwedge_{1 \leq j < i} tcflow_j$$

$$initial \overset{def.}{=} \bigwedge_{p \in In^b \cup \mathcal{P}_S^U} (p,1) \wedge \bigwedge_{p \in \mathcal{P}^b \setminus In^b} \neg(p,1)$$

Each environment place has to choose exactly one outgoing transition which results in the firing of at most one outgoing transition per environment place, because the other places in the precondition of the transition also have to decide for the transition. This encoding furthermore ensures *progress* (E3). Notice that we include the system players in the additional system places $\mathcal{P}_S^U$ as in Chapter 5. We substitute the sequential flow $seqflow_i$ by the true concurrent flow $tcflow_i$, which enforces the firing of all enabled and not stalled transitions and maintains all other tokens.

$$tcflow_i \overset{def.}{=} fireenabled_i \wedge updateplaces_i$$

$$fireenabled_i \overset{def.}{=} \bigwedge_{t \in \mathcal{T}^b} \Big(enabled_{i,t} \Rightarrow \bigwedge_{p \in pre^b(t) \setminus post^b(t)} \neg(p, i+1) \wedge \bigwedge_{p \in post^b(t)} (p, i+1) \Big)$$

$$updateplaces_i \overset{def.}{=} \bigwedge_{p \in \mathcal{P}^b} \Big(\bigwedge_{t \in pre^b(p) \cup post^b(p)} \neg enabled_{i,t} \Rightarrow \big((p,i) \Leftrightarrow (p,i+1) \big) \Big)$$

$$enabled_{i,t} \overset{def.}{=} \bigwedge_{p \in pre^b(t)} (p,i) \wedge \bigwedge_{p \in pre^b(t) \wedge \lambda^b(p) \in \mathcal{P}_S} (p, \lambda^b(t)) \wedge \bigwedge_{p \in pre^b(t) \wedge \lambda^b(p) \in \mathcal{P}_E} (p,t,i) \wedge (t) \wedge \bigwedge_{p \in pre^U(t)} \big((p,t) \wedge (p,i) \wedge (p,i+1) \big)$$

The formula $enabled_{i,t}$ requires tokens in all places in the precondition of the transition, both the strategy for the system players and the strategy for the environment players to allow the transition for corresponding places in the precondition of the transition,

that stalling allows the transition, and that the system players in the additional system places allow the transition and do not move. In $fireenabled_i$, it is ensured that all enabled transitions are fired by moving the corresponding tokens. Formula $updateplaces_i$ requires that tokens stay in places where all transitions from the place's precondition and from the place's postcondition are not enabled.

$$win_i \stackrel{def.}{=} nobadplace_i \wedge deterministic_i \wedge \Big(deadlock_i \Rightarrow terminating_i\Big)$$

$$nobadplace_i \stackrel{def.}{=} \bigwedge_{p \in \mathcal{P}^b \wedge \lambda^b(p) \in \mathcal{P}_B} \neg(p,i)$$

$$\begin{aligned} deterministic_i \stackrel{def.}{=} \bigwedge_{\substack{t_1,t_2 \in \mathcal{T}^b \wedge t_1 \neq t_2 \wedge \\ \lambda^b[pre^b(t_1)] \cap \lambda^b[pre^b(t_2)] \cap \mathcal{P}_S \neq \emptyset}} \Bigg(& \bigvee_{p \in pre^b(t_1) \cup pre^b(t_2)} \neg(p,i) \vee \\ & \bigvee_{p_1 \in pre^b(t_1) \wedge \lambda^b(p_1) \in \mathcal{P}_S} \neg(p_1, \lambda^b(t_1)) \vee \bigvee_{p_2 \in pre^b(t_2) \wedge \lambda^b(p_2) \in \mathcal{P}_S} \neg(p_2, \lambda^b(t_2)) \vee \\ & \bigvee_{p_1 \in pre^U(t_1)} \neg(p_1,t_1) \vee \bigvee_{p_2 \in pre^U(t_2)} \neg(p_2,t_2) \Bigg) \end{aligned}$$

$$deadlock_i \stackrel{def.}{=} \bigwedge_{t \in \mathcal{T}^b} \Bigg(\bigvee_{p \in pre^b(t)} \neg(p,i) \vee \bigvee_{p \in pre^b(t) \wedge \lambda^b(p) \in \mathcal{P}_S} \neg(p, \lambda^b(t)) \vee \bigvee_{p \in pre^U(t)} \neg(p,t) \Bigg)$$

$$terminating_i \stackrel{def.}{=} \bigwedge_{t \in \mathcal{T}^b} \Bigg(\bigvee_{p \in pre^b(t)} \neg(p,i) \Bigg)$$

The formula $nobadplace_i$ prevents that a bad places can be reached by enumerating the bad places and excluding that they are reached. Otherwise, the formula win_i remains unchanged, i.e., the formulas $deterministic_i$, $deadlock_i$, and $terminating_i$ are as in the sequential QBF encoding including the addition of system players in additional system places. Therefore, strategies for the environment players and stalling only affect the flow of tokens but not the check that reached markings are winning.

Strategies for the environment players allow us to define the true concurrent flow semantics. In turn, this allows us to detect loops in the form of repetitions of markings earlier by searching for them in strongly connected components (SCCs) [Jen13]. We define SCCs on the graph of reachable markings of the underlying Petri net. A subgraph is *strongly connected* when every vertex can be reached from every other vertex. Strongly connected components partition a graph into subgraphs in such a way that all subgraphs are themselves strongly connected. We define the places in the markings of a SCC to form the SCC. Places that are not part of any SCC form an additional set included in the SCC.

With SCCs, we find loops in independent parts of the Petri game as early as possible, i.e., we deploy a local view on loops instead of the global view of the encoding in Chapter 5. This constitutes an optimization. We encode that a loop no longer only occurs at the repetition of a global marking but also when all $SCCs \subseteq \mathbb{P}\left(\mathcal{P}^b\right)$ repeat their marking, respectively:

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

6.5. Experimental Results

We compare the sequential QBF encoding [Fin15] with the new true concurrent QBF encoding from Section 6.4 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. Afterward, we outline the technical details of our comparison framework and state our observations and explanations concerning the observed times required in order to find winning strategies for the system players.

6.5.1. Benchmark Families

Table 6.5 refers to the following scalable benchmark families where Collision Avoidance, Disjoint Routing, and Production Line are new benchmark families:

- **AS**: *Alarm System* [FGHO17]. 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, i.e., no false alarms and no false reports. For $m = 2$, the corresponding Petri game is depicted in Figure 5.2 of Chapter 5. It is used to illustrate bounded unfoldings and bounded strategies.

- **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 switch and an egress switch where the network allows m disjoint paths between the two switches.

- **PL:** *Production Line*. Parameters: m robots. The m concurrent 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* [FGO15]. Parameters: m workers. A document circulates between m workers with the environment player choosing the first worker to see the document. It is required that all workers unanimously endorse or reject the document.

6.5.2. Comparison Framework

Because both the sequential and the true concurrent QBF encoding result in a QBF not in conjunctive normal form, we use the QBF solver QuAbS [Ten16; HT18] as discussed in Section 5.6. The results from Table 6.5 were obtained on a 3.6 GHz quad-core Intel Xeon processor and 32 GB RAM. They are the average over five runs. There were only negligible differences between individual runs on the same problem instance. For each benchmark family (column *Ben.*), we report on the attempted parameters of the benchmark (column *Par.*), the necessary model checking iterations (column *Iter.*) of bounded synthesis, and on the runtime for finding a winning strategy for the system players (column *Runtime in sec.*). A timeout of 30 minutes is used. For reference, 30 minutes are equal to 1800 seconds. We prepared and published an artifact to replicate our experimental results [HM19a].

6.5.3. Observation

For each row in Table 6.5, the lower iteration count and the lower runtime between the sequential encoding and the true concurrent encoding are indicated in bold, respectively. The true concurrent QBF encoding shows considerable improvements over the sequential QBF encoding on the presented benchmark families: It solves more instances and has mostly faster solving times as shown in Table 6.5. The improvements are based on fewer model checking iterations of the bounded synthesis algorithm witnessed by the column *Iter.*.

We can make the following observations concerning the specific benchmark families: The complex communication structure of the benchmark family 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, the benchmark family Collision Avoidance has a complex pairwise communication structure which can be better synthesized by the true concurrent QBF encoding. The simpler communication structure of the benchmark family Production Line allows constant bounds for the true concurrent QBF encoding compared to linearly increasing bounds for the sequential QBF encoding. The communication structure of the benchmark family Disjoint Routing lays between complex and simple such that the true concurrent QBF encoding enables a smaller linear increase in the bound. The true concurrent QBF 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 Disjoint Routing and Production Line whereas they are completely open in the benchmark families Alarm System and Collision Avoidance.

Table 6.5.: Benchmarking results for the five *benchmark families* (*Ben.*) for increasing *parameters* (*Par.*) are presented. The five benchmark families are alarm system (AS), collision avoidance (CA), disjoint routing (DR), production line (PL), and document workflow (DW). For an increasing parameter, they produce larger Petri games, respectively. For the *sequential* and the *true concurrent* encoding, the needed model checking *iterations* (*Iter.*) with accumulated *runtime in seconds* (*sec.*) in order to find a winning strategy for the system players are reported. The used timeout is 1800 seconds.

		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
	6	18	**26.71**	**17**	31.79
	7	20	**49.69**	**19**	56.38
	8	22	**90.41**	**21**	132.73
	9	24	**239.47**	**23**	327.68
	10	26	**716.61**	**25**	823.94
	11	**28**	**1304.14**	-	timeout
	12	-	timeout	-	timeout

In the benchmark family 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.

6.6. Summary

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 QBF encoding. The previous sequential QBF encoding enumerated all interleavings. For the true concurrent QBF encoding, we represent the decisions of the environment players explicitly as strategies for the environment players and showed that this enables us to fire all enabled transitions as early as possible while maintaining the existence of winning strategies for the system players. The experimental results show the following: Only in the rare case of benchmark families without true concurrent transitions, our tool implementation of the true concurrent QBF encoding is similar to the sequential QBF encoding despite resulting in larger QBFs. In all other cases, our tool implementation of the true concurrent QBF encoding outperforms the sequential QBF encoding by a considerable margin.

Chapter 7

Conclusions

This thesis made two contributions to the foundations of Petri games. Petri games are a formal model that defines the synthesis problem of obtaining asynchronous distributed systems with causal memory. In the first part of this thesis, we presented decidability and undecidability results for Petri games. We outlined for which classes of Petri games is the question decidable or undecidable, whether a winning strategy for the system players in the Petri game exists. In the second part of this thesis, we presented bounded synthesis for Petri games as a semi-decision procedure to find smallest winning strategies for the system players in all Petri games, i.e., in Petri games with known as well as with unknown decidability and undecidability.

7.1. Decidability and Undecidability

As a *decidability* result, we proved that the existence of a winning strategy for the system players in Petri games with a bounded number of system players, at most one environment player, and bad markings as global winning condition is decidable. This result follows from solving the Büchi game created by our reduction in Chapter 3. It extends the existing decidability result for Petri games with a bounded number of system players, one environment player, and bad places as local winning condition [FO17] from local to global winning conditions while handling the determinism requirement for the strategy for the system players correctly in all situations. The decidability result complements the existing decidability result for Petri games with a bounded number of environment players, one system player, and bad markings as global winning condition [FG17]. The existence of a winning strategy for the system players is also decidable in Petri games with an acyclic communication architecture [BFH19a].

Our first *undecidability* result was the proof that the existence of a winning strategy for the system players in Petri games with good and bad markings as global winning condition, at least one environment player, and at least two system players is undecidable. This result follows from a reduction from the Post correspondence prob-

lem (PCP). Second, we proved that the existence of a winning strategy for the system players in Petri games with good markings as global winning condition, with at least one environment player, and with at least two players that each can change between being a system player and an environment player is undecidable. To obtain this result, we encode bad markings from the first undecidability result by each player repeatedly changing to an environment player.

The general decidability (or undecidability) of Petri games with a bounded number of system players *and* a bounded number of environment players remains an intriguing open problem. Further steps towards general decidability (or undecidability) could be to extend the decidability result for bad markings as global safety winning condition with local liveness specifications per player as in Flow-LTL [FGHO19; FGHO20a] and in Flow-CTL* [FGHO20b]. Another step could be to tackle Petri games with more than one environment (system) player and stronger restrictions, e.g., fixing, on the number of system (environment) players than just bounding their number. Here, the case of two system players and two environment players could be a fascinating starting point with some preliminary work to build upon [Han19].

Another interesting direction for future work is to investigate how action-based control games [MWZ09] relate to Petri games and to study unified models that combine features from control games and Petri games. Furthermore, the translations between control games and Petri games could be tailored more towards specific winning conditions or towards specific decidability results. This could create more opportunities to translate more decidability results from control games to Petri games, and vice versa. This relates to another difference between Petri games and control games: Petri games allow for the creation of new players and the termination of old players, i.e., the size of the precondition of a transition and the postcondition of the transition are not necessarily the same, whereas, in control games, the number of players is fixed. Therefore, it is an interesting question whether Petri games with a bounded number of players can be extended by adding places, tokens, flow, and maybe transitions such that all transitions are concurrency-preserving (i.e., the size of the precondition and the size of the postcondition of all transitions match) without leakage of any information between the players. The prohibition against leaking information between the players makes this problem both challenging and interesting because classical approaches from Petri nets do not apply. There exists some preliminary work that solves this problem for some Petri games [Sch19], but the general problem is open.

Although the synthesis of asynchronous distributed systems with causal memory is already a hard problem, an even harder problem emerges when causal memory and partial observation [AVW03; CD10] are combined in order to obtain a more advanced memory model. Partial observation could allow to inform only certain players during synchronization. An intermediate step could be a so-called hiding operator or so-called forgetful places. A hiding operator could allow to abstract certain parts of the behavior of processes [OH86]. Forgetful places are created by enforcing players to forget their causal past [Buh19]. Both approaches could help with modeling the avoidance of accidental leakage of information.

7.2. Bounded Synthesis

For bounded synthesis, we presented the sequential encoding in which the asynchronous nature of the players in Petri games is encoded in terms of so-called sequential firing sequences. In a sequential firing sequence, only one transition is fired between two subsequent markings. We also presented the true concurrent encoding, where the asynchronous nature of the players is encoded by grouping together concurrent transitions in the firing sequences. Both encodings are implemented in the tool AdamSYNT [Ada20], which has an accompanying web interface [GHY21]. Our evaluation showed that the true concurrent encoding outperforms the sequential one on a broad set of benchmarks from robotic control, workflow management, and other distributed applications.

Bounded synthesis could be extended with more expressive winning conditions, as in the case of general decidability results for Petri games. In bounded synthesis, it is easier to encode more expressive winning conditions than when searching for decidability results. For instance, LTL, CTL, Büchi winning conditions, parity winning conditions, or generalizations thereof [CHP07] seem achievable. Going from solving QBFs to SMT solving [MB08; Dut14] could allow for extending bounded synthesis from safe Petri games to bounded Petri games. That is, a bounded number of players per place could be possible instead of only one.

An orthogonal extension for bounded synthesis of Petri games could be partial observation or forgetful places, as mentioned before. Here, DQBF solving [FT14b; FKBV14; Git+15; TR19] could present a starting point. DQBF is an extension of QBF and allows for specific dependencies between quantifiers in contrast to the linear dependencies of quantifiers in QBF. These dependencies could be used to restrict the knowledge of parts of the strategy.

An exciting application scenario for bounded synthesis of Petri games could be multi-lane traffic maneuvers of autonomous cars [BHLO17]. A promising direction to apply bounded synthesis of Petri games to real-world problems is to design access-control policies for physical spaces [TDB16]. Here, the rely-guarantee paradigm [Sta85; AL89] could be used to synthesize policies for separate floors or wings of the building assuming certain assumptions and then to compose the individual floors or wings such that all relied-on assumptions are guaranteed. In general, we could identify disconnected parts of Petri games, solve the parts in isolation, and compose them back together.

Bibliography

[ABP21] Federica Adobbati, Luca Bernardinello, and Lucia Pomello. "A two-player asynchronous game on fully observable Petri nets". In: *Trans. Petri Nets Other Model. Concurr.* 15 (2021), pp. 126–149. DOI: 10.1007/978-3-662-63079-2_6.

[AD94] Rajeev Alur and David L. Dill. "A theory of timed automata". In: *Theor. Comput. Sci.* 126.2 (1994), pp. 183–235. DOI: 10.1016/0304-3975(94)90010-8.

[Ada20] *Adam*. https://github.com/adamtool/. 2020.

[AETP01] Rajeev Alur, Kousha Etessami, Salvatore La Torre, and Doron A. Peled. "Parametric temporal logic for "model measuring"". In: *ACM Trans. Comput. Log.* 2.3 (2001), pp. 388–407. DOI: 10.1145/377978.377990.

[AFSS17] Rajeev Alur, Dana Fisman, Rishabh Singh, and Armando Solar-Lezama. "SyGuS-Comp 2017: results and analysis". In: *Proceedings of SYNT@CAV 2017*. EPTCS. Vol. 260. 2017, pp. 97–115. DOI: 10.4204/EPTCS.260.9.

[AHK02] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. "Alternating-time temporal logic". In: *J. ACM* 49.5 (2002), pp. 672–713. DOI: 10.1145/585265.585270.

[AL89] Martín Abadi and Leslie Lamport. "Composing specifications". In: *Proceedings of REX*. Lecture Notes in Computer Science. Vol. 430. Springer, 1989, pp. 1–41. DOI: 10.1007/3-540-52559-9_59.

[Alu+15] Rajeev Alur, Rastislav Bodík, Eric Dallal, Dana Fisman, Pranav Garg, Garvit Juniwal, Hadas Kress-Gazit, P. Madhusudan, Milo M. K. Martin, Mukund Raghothaman, Shambwaditya Saha, Sanjit A. Seshia, Rishabh Singh, Armando Solar-Lezama, Emina Torlak, and Abhishek Udupa. "Syntax-guided synthesis". In: *Dependable Software Systems Engineering*. NATO Science for Peace and Security Series, D: Information and Communication Security. Vol. 40. IOS Press, 2015, pp. 1–25. DOI: 10.3233/978-1-61499-495-4-1.

[Alu+19] Rajeev Alur, Dana Fisman, Saswat Padhi, Rishabh Singh, and Abhishek Udupa. "SyGuS-Comp 2018: results and analysis". In: *CoRR* abs/1904.07146 (2019). arXiv: 1904.07146.

[Alu99] Rajeev Alur. "Timed automata". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 1633. Springer, 1999, pp. 8–22. DOI: 10.1007/3-540-48683-6_3.

[ALW89] Martín Abadi, Leslie Lamport, and Pierre Wolper. "Realizable and unrealizable specifications of reactive systems". In: *Proceedings of ICALP*. Lecture Notes in Computer Science. Vol. 372. Springer, 1989, pp. 1–17. DOI: 10.1007/BFb0035748.

[AMPS98] Eugene Asarin, Oded Maler, Amir Pnueli, and Joseph Sifakis. "Controller synthesis for timed automata". In: *IFAC Proceedings Volumes* 31.18 (1998), pp. 447–452. DOI: 10.1016/S1474-6670(17)42032-5.

[AMT15] Rajeev Alur, Salar Moarref, and Ufuk Topcu. "Pattern-based refinement of assume-guarantee specifications in reactive synthesis". In: *Proceedings of TACAS*. Lecture Notes in Computer Science. Vol. 9035. Springer, 2015, pp. 501–516. DOI: 10.1007/978-3-662-46681-0_49.

[AT17] Rajeev Alur and Stavros Tripakis. "Automatic synthesis of distributed protocols". In: *SIGACT News* 48.1 (2017), pp. 55–90. DOI: 10.1145/3061640.3061652.

[AVW03] André Arnold, Aymeric Vincent, and Igor Walukiewicz. "Games for synthesis of controllers with partial observation". In: *Theor. Comput. Sci.* 303.1 (2003), pp. 7–34. DOI: 10.1016/S0304-3975(02)00442-5.

[Bar+11] Clark W. Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. "CVC4". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 6806. Springer, 2011, pp. 171–177. DOI: 10.1007/978-3-642-22110-1_14.

[BBD15] Éric Badouel, Luca Bernardinello, and Philippe Darondeau. *Petri net synthesis*. Texts in Theoretical Computer Science. An EATCS Series. Springer, 2015. DOI: 10.1007/978-3-662-47967-4.

[BBJ16] Roderick Bloem, Nicolas Braud-Santoni, and Swen Jacobs. "Synthesis of self-stabilising and byzantine-resilient distributed systems". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 9779. Springer, 2016, pp. 157–176. DOI: 10.1007/978-3-319-41528-4_9.

[BBLS20] Béatrice Bérard, Benedikt Bollig, Mathieu Lehaut, and Nathalie Sznajder. "Parameterized synthesis for fragments of first-order logic over data words". In: *Proceedings of FoSSaCS*. Lecture Notes in Computer Science. Vol. 12077. Springer, 2020, pp. 97–118. DOI: 10.1007/978-3-030-45231-5_6.

[BCHJ09] Roderick Bloem, Krishnendu Chatterjee, Thomas A. Henzinger, and Barbara Jobstmann. "Better quality in synthesis through quantitative objectives". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 5643. Springer, 2009, pp. 140–156. DOI: 10.1007/978-3-642-02658-4_14.

[BCJ18] Roderick Bloem, Krishnendu Chatterjee, and Barbara Jobstmann. "Graph games and reactive synthesis". In: *Handbook of Model Checking*. Springer, 2018, pp. 921–962. DOI: 10.1007/978-3-319-10575-8_27.

[BCJK15] Roderick Bloem, Krishnendu Chatterjee, Swen Jacobs, and Robert Könighofer. "Assume-guarantee synthesis for concurrent reactive programs with partial information". In: *Proceedings of TACAS*. Lecture Notes in Computer Science. Vol. 9035. Springer, 2015, pp. 517–532. DOI: 10.1007/978-3-662-46681-0_50.

[BDLV05] Ugo Buy, Houshang Darabi, Mihai Lehene, and Vikram Venepally. "Supervisory control of time Petri nets using net unfolding". In: *Proceedings of COMPSAC*. IEEE Computer Society, 2005, pp. 97–100. DOI: 10.1109/COMPSAC.2005.148.

[BDMP03] Patricia Bouyer, Deepak D'Souza, P. Madhusudan, and Antoine Petit. "Timed control with partial observability". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 2725. Springer, 2003, pp. 180–192. DOI: 10.1007/978-3-540-45069-6_18.

[Beh+07] Gerd Behrmann, Agnès Cougnard, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. "UPPAAL-TIGA: time for playing games!" In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 4590. Springer, 2007, pp. 121–125. DOI: 10.1007/978-3-540-73368-3_14.

[BELM12] Bernd Becker, Rüdiger Ehlers, Matthew Lewis, and Paolo Marin. "ALLQBF solving by computational learning". In: *Proceedings of ATVA*. Lecture Notes in Computer Science. Vol. 7561. Springer, 2012, pp. 370–384. DOI: 10.1007/978-3-642-33386-6_29.

[Beu19] Raven Beutner. "Translating asynchronous games for distributed synthesis". Bachelor's Thesis. Saarland University, Saarbrücken, Germany, 2019.

[BF88] Eike Best and César Fernández. *Nonsequential processes - a Petri net view*. EATCS Monographs on Theoretical Computer Science. Vol. 13. Springer, 1988. DOI: 10.1007/978-3-642-73483-0.

[BFH19a] Raven Beutner, Bernd Finkbeiner, and Jesko Hecking-Harbusch. "Translating asynchronous games for distributed synthesis". In: *Proceedings of CONCUR*. LIPIcs. Vol. 140. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 26:1–26:16. DOI: 10.4230/LIPIcs.CONCUR.2019.26.

[BFH19b] Raven Beutner, Bernd Finkbeiner, and Jesko Hecking-Harbusch. "Translating asynchronous games for distributed synthesis (Full Version)". In: *CoRR* abs/1907.00829 (2019). arXiv: 1907.00829.

[BHLO17] Gregor von Bochmann, Martin Hilscher, Sven Linker, and Ernst-Rüdiger Olderog. "Synthesizing and verifying controllers for multi-lane traffic maneuvers". In: *Formal Aspects Comput.* 29.4 (2017), pp. 583–600. DOI: 10.1007/s00165-017-0424-4.

[Bie+03] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. "Bounded model checking". In: *Adv. Comput.* 58 (2003), pp. 117–148. DOI: 10.1016/S0065-2458(03)58003-2.

[BL69] J. Richard Büchi and Lawrence H. Landweber. "Solving sequential conditions by finite state strategies". In: *Trans. Am. Math. Soc.* 138 (1969), pp. 295–311.

[Blo+07] Roderick Bloem, Stefan J. Galler, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Martin Weiglhofer. "Automatic hardware synthesis from specifications: a case study". In: *Proceedings of DATE*. EDA Consortium, 2007, pp. 1188–1193.

[Blo+12] Roderick Bloem, Barbara Jobstmann, Nir Piterman, Amir Pnueli, and Yaniv Sa'ar. "Synthesis of reactive(1) designs". In: *J. Comput. Syst. Sci.* 78.3 (2012), pp. 911–938. DOI: 10.1016/j.jcss.2011.08.007.

[BMMR01] Thomas Ball, Rupak Majumdar, Todd D. Millstein, and Sriram K. Rajamani. "Automatic predicate abstraction of C programs". In: *Proceedings of PLDI*. ACM, 2001, pp. 203–213. DOI: 10.1145/378795.378846.

[BMS05] Clark W. Barrett, Leonardo Mendonça de Moura, and Aaron Stump. "SMT-COMP: satisfiability modulo theories competition". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 3576. Springer, 2005, pp. 20–23. DOI: 10.1007/11513988_4.

[BNS18] Suguman Bansal, Kedar S. Namjoshi, and Yaniv Sa'ar. "Synthesis of asynchronous reactive programs from temporal specifications". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 10981. Springer, 2018, pp. 367–385. DOI: 10.1007/978-3-319-96145-3_20.

[Boh+12] Aaron Bohy, Véronique Bruyère, Emmanuel Filiot, Naiyong Jin, and Jean-François Raskin. "Acacia+, a tool for LTL synthesis". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 7358. Springer, 2012, pp. 652–657. DOI: 10.1007/978-3-642-31424-7_45.

[Bon+14] Blai Bonet, Patrik Haslum, Victor Khomenko, Sylvie Thiébaux, and Walter Vogler. "Recent advances in unfolding technique". In: *Theor. Comput. Sci.* 551 (2014), pp. 84–101. DOI: 10.1016/j.tcs.2014.07.003.

[Bra11] Aaron R. Bradley. "SAT-based model checking without unrolling". In: *Proceedings of VMCAI*. Lecture Notes in Computer Science. Vol. 6538. Springer, 2011, pp. 70–87. DOI: 10.1007/978-3-642-18275-4_7.

[BRS17] Romain Brenguier, Jean-François Raskin, and Ocan Sankur. "Assume-admissible synthesis". In: *Acta Informatica* 54.1 (2017), pp. 41–83. DOI: 10.1007/s00236-016-0273-2.

[Büc60] J. Richard Büchi. *On a decision method in restricted second-order arithmetic*. 1960.

[Buh19] Moritz Buhr. "Forgetful Petri games – synthesizing distributed systems with partially observable causal memory". Bachelor's Thesis. University of Oldenburg, Oldenburg, Germany, 2019.

[Cal+17] Cristian S. Calude, Sanjay Jain, Bakhadyr Khoussainov, Wei Li, and Frank Stephan. "Deciding parity games in quasipolynomial time". In: *Proceedings of STOC*. ACM, 2017, pp. 252–263. DOI: 10.1145/3055399.3055409.

[Car+17] Luca Cardelli, Milan Ceska, Martin Fränzle, Marta Z. Kwiatkowska, Luca Laurenti, Nicola Paoletti, and Max Whitby. "Syntax-guided optimal synthesis for chemical reaction networks". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 10427. Springer, 2017, pp. 375–395. DOI: 10.1007/978-3-319-63390-9_20.

[Cas+05] Franck Cassez, Alexandre David, Emmanuel Fleury, Kim Guldstrand Larsen, and Didier Lime. "Efficient on-the-fly algorithms for the analysis of timed games". In: *Proceedings of CONCUR*. Lecture Notes in Computer Science. Vol. 3653. Springer, 2005, pp. 66–80. DOI: 10.1007/11539452_9.

[CD10] Krishnendu Chatterjee and Laurent Doyen. "The complexity of partial-observation parity games". In: *Proceedings of LPAR*. Lecture Notes in Computer Science. Vol. 6397. Springer, 2010, pp. 1–14. DOI: 10.1007/978-3-642-16242-8_1.

[CE81] Edmund M. Clarke and E. Allen Emerson. "Design and synthesis of synchronization skeletons using branching-time temporal logic". In: *Proceedings of LOP*. Lecture Notes in Computer Science. Vol. 131. Springer, 1981, pp. 52–71. DOI: 10.1007/BFb0025774.

[CH07] Krishnendu Chatterjee and Thomas A. Henzinger. "Assume-guarantee synthesis". In: *Proceedings of TACAS*. Lecture Notes in Computer Science. Vol. 4424. Springer, 2007, pp. 261–275. DOI: 10.1007/978-3-540-71209-1_21.

[CH12] Krishnendu Chatterjee and Monika Henzinger. "An $O(n^2)$ time algorithm for alternating Büchi games". In: *Proceedings of SODA*. SIAM, 2012, pp. 1386–1399. DOI: 10.1137/1.9781611973099.109.

[CHP07] Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. "Generalized parity games". In: *Proceedings of FoSSaCS*. Lecture Notes in Computer Science. Vol. 4423. Springer, 2007, pp. 153–167. DOI: 10.1007/978-3-540-71389-0_12.

[CHP10] Krishnendu Chatterjee, Thomas A. Henzinger, and Nir Piterman. "Strategy logic". In: *Inf. Comput.* 208.6 (2010), pp. 677–693. DOI: 10.1016/j.ic.2009.07.004.

[Chu57] Alonzo Church. *Application of recursive arithmetic to the problem of circuit synthesis*. 1957.

[Chu64] Alonzo Church. "Logic, arithmetic, and automata". In: *J. Symb. Log.* 29.4 (1964). DOI: 10.2307/2270398.

[CIP09] Chris Calabro, Russell Impagliazzo, and Ramamohan Paturi. "The complexity of satisfiability of small depth circuits". In: *Proceedings of IWPEC*. Lecture Notes in Computer Science. Vol. 5917. Springer, 2009, pp. 75–85. DOI: 10.1007/978-3-642-11269-0_6.

[CKL04] Edmund M. Clarke, Daniel Kroening, and Flavio Lerda. "A tool for checking ANSI-C programs". In: *Proceedings of TACAS*. Lecture Notes in Computer Science. Vol. 2988. Springer, 2004, pp. 168–176. DOI: 10.1007/978-3-540-24730-2_15.

[CKP15] Byron Cook, Heidy Khlaaf, and Nir Piterman. "Fairness for infinite-state systems". In: *Proceedings of TACAS*. Lecture Notes in Computer Science. Vol. 9035. Springer, 2015, pp. 384–398. DOI: 10.1007/978-3-662-46681-0_30.

[CMT99] Ilaria Castellani, Madhavan Mukund, and P. S. Thiagarajan. "Synthesizing distributed transition systems from global specification". In: *Proceedings of FSTTCS*. Lecture Notes in Computer Science. Vol. 1738. Springer, 1999, pp. 219–231. DOI: 10.1007/3-540-46691-6_17.

[Cop+01] Fady Copty, Limor Fix, Ranan Fraer, Enrico Giunchiglia, Gila Kamhi, Armando Tacchella, and Moshe Y. Vardi. "Benefits of bounded model checking at an industrial setting". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 2102. Springer, 2001, pp. 436–453. DOI: 10.1007/3-540-44585-4_43.

[DF11] Werner Damm and Bernd Finkbeiner. "Does it pay to extend the perimeter of a world model?" In: *Proceedings of FM*. Lecture Notes in Computer Science. Vol. 6664. Springer, 2011, pp. 12–26. DOI: 10.1007/978-3-642-21437-0_4.

[DF12] Rayna Dimitrova and Bernd Finkbeiner. "Counterexample-guided synthesis of observation predicates". In: *Proceedings of FORMATS*. Lecture Notes in Computer Science. Vol. 7595. Springer, 2012, pp. 107–122. DOI: 10.1007/978-3-642-33365-1_9.

[DF14] Werner Damm and Bernd Finkbeiner. "Automatic compositional synthesis of distributed systems". In: *Proceedings of FM*. Lecture Notes in Computer Science. Vol. 8442. Springer, 2014, pp. 179–193. DOI: 10.1007/978-3-319-06410-9_13.

[DFT19] Rayna Dimitrova, Bernd Finkbeiner, and Hazem Torfah. "Synthesizing approximate implementations for unrealizable specifications". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 11561. Springer, 2019, pp. 241–258. DOI: 10.1007/978-3-030-25540-4_13.

[Did+21] Martin Didriksen, Peter Gjøl Jensen, Jonathan F. Jønler, Andrei-Ioan Katona, Sangey D. L. Lama, Frederik B. Lottrup, Shahab Shajarat, and Jirí Srba. "Automatic synthesis of transiently correct network updates via Petri games". In: *Proceedings of PETRI NETS*. Lecture Notes in Computer Science. Vol. 12734. Springer, 2021, pp. 118–137. DOI: 10.1007/978-3-030-76983-3_7.

[DM02] Deepak D'Souza and P. Madhusudan. "Timed control synthesis for external specifications". In: *Proceedings of STACS*. Lecture Notes in Computer Science. Vol. 2285. Springer, 2002, pp. 571–582. DOI: 10.1007/3-540-45841-7_47.

[DR96] Jörg Desel and Wolfgang Reisig. "The synthesis problem of Petri nets". In: *Acta Informatica* 33.4 (1996), pp. 297–315. DOI: 10.1007/s002360050046.

[Dut14] Bruno Dutertre. "Yices 2.2". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 8559. Springer, 2014, pp. 737–744. DOI: 10.1007/978-3-319-08867-9_49.

[EF17] Rüdiger Ehlers and Bernd Finkbeiner. "Symmetric synthesis". In: *Proceedings of FSTTCS*. LIPIcs. Vol. 93. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 26:1–26:13. DOI: 10.4230/LIPIcs.FSTTCS.2017.26.

[EH08] Javier Esparza and Keijo Heljanko. *Unfoldings - a partial-order approach to model checking*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2008. DOI: 10.1007/978-3-540-77426-6.

[EH85] E. Allen Emerson and Joseph Y. Halpern. "Decision procedures and expressiveness in the temporal logic of branching time". In: *J. Comput. Syst. Sci.* 30.1 (1985), pp. 1–24. DOI: 10.1016/0022-0000(85)90001-7.

[Ehl11] Rüdiger Ehlers. "Unbeast: symbolic bounded synthesis". In: *Proceedings of TACAS*. Lecture Notes in Computer Science. Vol. 6605. Springer, 2011, pp. 272–275. DOI: 10.1007/978-3-642-19835-9_25.

[EJ88] E. Allen Emerson and Charanjit S. Jutla. "The complexity of tree automata and logics of programs (Extended Abstract)". In: *Proceedings of FOCS*. IEEE Computer Society, 1988, pp. 328–337. DOI: 10.1109/SFCS.1988.21949.

[EJ91] E. Allen Emerson and Charanjit S. Jutla. "Tree automata, μ-calculus and determinacy (Extended Abstract)". In: *Proceedings of FOCS*. IEEE Computer Society, 1991, pp. 368–377. DOI: 10.1109/SFCS.1991.185392.

[EJS93] E. Allen Emerson, Charanjit S. Jutla, and A. Prasad Sistla. "On model-checking for fragments of μ-calculus". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 697. Springer, 1993, pp. 385–396. DOI: 10.1007/3-540-56922-7_32.

[EL87] E. Allen Emerson and Chin-Laung Lei. "Modalities for model checking: branching time logic strikes back". In: *Sci. Comput. Program.* 8.3 (1987), pp. 275–306. DOI: 10.1016/0167-6423(87)90036-0.

[Eme85] E. Allen Emerson. "Automata, tableaux and temporal logics (Extended Abstract)". In: *Proceedings of LOP*. Lecture Notes in Computer Science. Vol. 193. Springer, 1985, pp. 79–88. DOI: 10.1007/3-540-15648-8_7.

[Eme90] E. Allen Emerson. "Temporal and modal logic". In: *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*. Elsevier and MIT Press, 1990, pp. 995–1072. DOI: 10.1016/b978-0-444-88074-1.50021-4.

[Eng91] Joost Engelfriet. "Branching processes of Petri nets". In: *Acta Informatica* 28.6 (1991), pp. 575–591. DOI: 10.1007/BF01463946.

[ER90] Andrzej Ehrenfeucht and Grzegorz Rozenberg. "Partial (set) 2-structures. Part II: state spaces of concurrent systems". In: *Acta Informatica* 27.4 (1990), pp. 343–368. DOI: 10.1007/BF00264612.

[ERV02] Javier Esparza, Stefan Römer, and Walter Vogler. "An improvement of McMillan's unfolding algorithm". In: *Formal Methods Syst. Des.* 20.3 (2002), pp. 285–310. DOI: 10.1023/A:1014746130920.

[Esp94] Javier Esparza. "Model checking using net unfoldings". In: *Sci. Comput. Program.* 23.2-3 (1994), pp. 151–195. DOI: 10.1016/0167-6423(94)00019-0.

[FB18] Grigory Fedyukovich and Rastislav Bodík. "Accelerating syntax-guided invariant synthesis". In: *Proceedings of TACAS*. Lecture Notes in Computer Science. Vol. 10805. Springer, 2018, pp. 251–269. DOI: 10.1007/978-3-319-89960-2_14.

[FFRT17] Peter Faymonville, Bernd Finkbeiner, Markus N. Rabe, and Leander Tentrup. "Encodings of bounded synthesis". In: *Proceedings of TACAS*. Lecture Notes in Computer Science. Vol. 10205. 2017, pp. 354–370. DOI: 10.1007/978-3-662-54577-5_20.

[FFT17] Peter Faymonville, Bernd Finkbeiner, and Leander Tentrup. "BoSy: an experimentation framework for bounded synthesis". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 10427. Springer, 2017, pp. 325–332. DOI: 10.1007/978-3-319-63390-9_17.

[FG05] Cormac Flanagan and Patrice Godefroid. "Dynamic partial-order reduction for model checking software". In: *Proceedings of POPL*. ACM, 2005, pp. 110–121. DOI: 10.1145/1040305.1040315.

[FG17] Bernd Finkbeiner and Paul Gölz. "Synthesis in distributed environments". In: *Proceedings of FSTTCS*. LIPIcs. Vol. 93. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 28:1–28:14. DOI: 10.4230/LIPIcs.FSTTCS.2017.28.

[FGHO17] Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. "Symbolic vs. bounded synthesis for Petri games". In: *Proceedings of SYNT@CAV*. EPTCS. Vol. 260. 2017, pp. 23–43. DOI: 10.4204/EPTCS.260.5.

[FGHO19] Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. "Model checking data flows in concurrent network updates". In: *Proceedings of ATVA*. Lecture Notes in Computer Science. Vol. 11781. Springer, 2019, pp. 515–533. DOI: 10.1007/978-3-030-31784-3_30.

[FGHO20a] Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. "AdamMC: a model checker for Petri nets with transits against Flow-LTL". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 12225. Springer, 2020, pp. 64–76. DOI: 10.1007/978-3-030-53291-8_5.

[FGHO20b] Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. "Model checking branching properties on Petri nets with transits". In: *Proceedings of ATVA*. Lecture Notes in Computer Science. Vol. 12302. Springer, 2020, pp. 394–410. DOI: 10.1007/978-3-030-59152-6_22.

[FGHO21] Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. "Global winning conditions in synthesis of distributed systems with causal memory (Full Version)". In: *CoRR* abs/2107.09280 (2021). arXiv: 2107.09280.

[FGHO22] Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. "Global winning conditions in synthesis of distributed systems with causal memory". In: *Proceedings of CSL*. LIPIcs. Vol. 216. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 36:1–36:19. DOI: 10.4230/LIPIcs.CSL.2022.36.

[FGO15] Bernd Finkbeiner, Manuel Gieseking, and Ernst-Rüdiger Olderog. "Adam: causality-based synthesis of distributed systems". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 9206. Springer, 2015, pp. 433–439. DOI: 10.1007/978-3-319-21690-4_25.

[Fin15] Bernd Finkbeiner. "Bounded synthesis for Petri games". In: *Proceedings of Correct System Design - Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday*. Lecture Notes in Computer Science. Vol. 9360. Springer, 2015, pp. 223–237. DOI: 10.1007/978-3-319-23506-6_15.

[Fin16] Bernd Finkbeiner. "Synthesis of reactive systems". In: *Dependable Software Systems Engineering*. NATO Science for Peace and Security Series - D: Information and Communication Security. Vol. 45. IOS Press, 2016, pp. 72–98. DOI: 10.3233/978-1-61499-627-9-72.

[FJ12] Bernd Finkbeiner and Swen Jacobs. "Lazy synthesis". In: *Proceedings of VMCAI*. Lecture Notes in Computer Science. Vol. 7148. Springer, 2012, pp. 219–234. DOI: 10.1007/978-3-642-27940-9_15.

[FK16] Bernd Finkbeiner and Felix Klein. "Bounded cycle synthesis". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 9779. Springer, 2016, pp. 118–135. DOI: 10.1007/978-3-319-41528-4_7.

[FK17] Bernd Finkbeiner and Felix Klein. "Reactive synthesis: towards output-sensitive algorithms". In: *Dependable Software Systems Engineering*. NATO Science for Peace and Security Series - D: Information and Communication Security. Vol. 50. IOS Press, 2017, pp. 25–43. DOI: 10.3233/978-1-61499-810-5-25.

[FKBV14] Andreas Fröhlich, Gergely Kovásznai, Armin Biere, and Helmut Veith. "iDQ: instantiation-based DQBF solving". In: *Proceedings of SAT*. EPiC Series in Computing. Vol. 27. EasyChair, 2014, pp. 103–116.

[FKPS19a] Bernd Finkbeiner, Felix Klein, Ruzica Piskac, and Mark Santolucito. "Synthesizing functional reactive programs". In: *Proceedings of Haskell@ICFP*. ACM, 2019, pp. 162–175. DOI: 10.1145/3331545.3342601.

[FKPS19b] Bernd Finkbeiner, Felix Klein, Ruzica Piskac, and Mark Santolucito. "Temporal stream logic: synthesis beyond the bools". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 11561. Springer, 2019, pp. 609–629. DOI: 10.1007/978-3-030-25540-4_35.

[FL79] Michael J. Fischer and Richard E. Ladner. "Propositional dynamic logic of regular programs". In: *J. Comput. Syst. Sci.* 18.2 (1979), pp. 194–211. DOI: 10.1016/0022-0000(79)90046-1.

[FMMV20] Nathanaël Fijalkow, Bastien Maubert, Aniello Murano, and Moshe Y. Vardi. "Assume-guarantee synthesis for prompt linear temporal logic". In: *Proceedings of IJCAI*. ijcai.org, 2020, pp. 117–123. DOI: 10.24963/ijcai.2020/17.

[FO17] Bernd Finkbeiner and Ernst-Rüdiger Olderog. "Petri games: synthesis of distributed systems with causal memory". In: *Inf. Comput.* 253 (2017), pp. 181–203. DOI: 10.1016/j.ic.2016.07.006.

[FP12] Bernd Finkbeiner and Hans-Jörg Peter. "Template-based controller synthesis for timed systems". In: *Proceedings of TACAS*. Lecture Notes in Computer Science. Vol. 7214. Springer, 2012, pp. 392–406. DOI: 10.1007/978-3-642-28756-5_27.

[FP20] Bernd Finkbeiner and Noemi Passing. "Dependency-based compositional synthesis". In: *Proceedings of ATVA*. Lecture Notes in Computer Science. Vol. 12302. Springer, 2020, pp. 447–463. DOI: 10.1007/978-3-030-59152-6_25.

[FP21] Bernd Finkbeiner and Noemi Passing. "Compositional synthesis of modular systems". In: *Proceedings of ATVA*. Lecture Notes in Computer Science. Vol. 12971. Springer, 2021, pp. 303–319. DOI: 10.1007/978-3-030-88885-5_20.

[FPMG19] Grigory Fedyukovich, Sumanth Prabhu, Kumar Madhukar, and Aarti Gupta. "Quantified invariants via syntax-guided synthesis". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 11561. Springer, 2019, pp. 259–277. DOI: 10.1007/978-3-030-25540-4_14.

[FS05] Bernd Finkbeiner and Sven Schewe. "Uniform distributed synthesis". In: *Proceedings of LICS*. IEEE Computer Society, 2005, pp. 321–330. DOI: 10.1109/LICS.2005.53.

[FS10] Bernd Finkbeiner and Sven Schewe. "Coordination logic". In: *Proceedings of CSL*. Lecture Notes in Computer Science. Vol. 6247. Springer, 2010, pp. 305–319. DOI: 10.1007/978-3-642-15205-4_25.

[FS13] Bernd Finkbeiner and Sven Schewe. "Bounded synthesis". In: *STTT* 15.5-6 (2013), pp. 519–539. DOI: 10.1007/s10009-012-0228-z.

[FT14a] Bernd Finkbeiner and Leander Tentrup. "Detecting unrealizable specifications of distributed systems". In: *Proceedings of TACAS*. Lecture Notes in Computer Science. Vol. 8413. Springer, 2014, pp. 78–92. DOI: 10.1007/978-3-642-54862-8_6.

[FT14b] Bernd Finkbeiner and Leander Tentrup. "Fast DQBF refutation". In: *Proceedings of SAT*. Lecture Notes in Computer Science. Vol. 8561. Springer, 2014, pp. 243–251. DOI: 10.1007/978-3-319-09284-3_19.

[GCH13] Yashdeep Godhal, Krishnendu Chatterjee, and Thomas A. Henzinger. "Synthesis of AMBA AHB from formal specification: a case study". In: *Int. J. Softw. Tools Technol. Transf.* 15.5-6 (2013), pp. 585–601. DOI: 10.1007/s10009-011-0207-9.

[GGMW13] Blaise Genest, Hugo Gimbert, Anca Muscholl, and Igor Walukiewicz. "Asynchronous games over tree architectures". In: *Proceedings of ICALP*. Lecture Notes in Computer Science. Vol. 7966. Springer, 2013, pp. 275–286. DOI: 10.1007/978-3-642-39212-2_26.

[GH82] Yuri Gurevich and Leo Harrington. "Trees, automata, and games". In: *Proceedings of STOC*. ACM, 1982, pp. 60–65. DOI: 10.1145/800070.802177.

[GHKF19] Gideon Geier, Philippe Heim, Felix Klein, and Bernd Finkbeiner. "Syntroids: synthesizing a game for FPGAs using temporal logic specifications". In: *Proceedings of FMCAD*. IEEE, 2019, pp. 138–146. DOI: 10.23919/FMCAD.2019.8894261.

[GHY21] Manuel Gieseking, Jesko Hecking-Harbusch, and Ann Yanich. "A web interface for Petri nets with transits and Petri games". In: *Proceedings of TACAS*. Lecture Notes in Computer Science. Vol. 12652. Springer, 2021, pp. 381–388. DOI: 10.1007/978-3-030-72013-1_22.

[Gim17] Hugo Gimbert. "On the control of asynchronous automata". In: *Proceedings of FSTTCS*. LIPIcs. Vol. 93. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 30:1–30:15. DOI: 10.4230/LIPIcs.FSTTCS.2017.30.

[Git+15] Karina Gitina, Ralf Wimmer, Sven Reimer, Matthias Sauer, Christoph Scholl, and Bernd Becker. "Solving DQBF through quantifier elimination". In: *Proceedings of DATE*. ACM, 2015, pp. 1617–1622.

[GJTV11] Sumit Gulwani, Susmit Jha, Ashish Tiwari, and Ramarathnam Venkatesan. "Synthesis of loop-free programs". In: *Proceedings of PLDI*. ACM, 2011, pp. 62–73. DOI: 10.1145/1993498.1993506.

[GKF18] Carsten Gerstacker, Felix Klein, and Bernd Finkbeiner. "Bounded synthesis of reactive programs". In: *Proceedings of ATVA*. Lecture Notes in Computer Science. Vol. 11138. Springer, 2018, pp. 441–457. DOI: 10.1007/978-3-030-01090-4_26.

[GL81] Hartmann J. Genrich and Kurt Lautenbach. "System modelling with high-level Petri nets". In: *Theor. Comput. Sci.* 13 (1981), pp. 109–136. DOI: 10.1016/0304-3975(81)90113-4.

[GLZ04a] Paul Gastin, Benjamin Lerman, and Marc Zeitoun. "Distributed games and distributed control for asynchronous systems". In: *Proceedings of LATIN*. Lecture Notes in Computer Science. Vol. 2976. Springer, 2004, pp. 455–465. DOI: 10.1007/978-3-540-24698-5_49.

[GLZ04b] Paul Gastin, Benjamin Lerman, and Marc Zeitoun. "Distributed games with causal memory are decidable for series-parallel systems". In: *Proceedings of FSTTCS*. Lecture Notes in Computer Science. Vol. 3328. Springer, 2004, pp. 275–286. DOI: 10.1007/978-3-540-30538-5_23.

[GO01] Paul Gastin and Denis Oddoux. "Fast LTL to Büchi automata translation". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 2102. Springer, 2001, pp. 53–65. DOI: 10.1007/3-540-44585-4_6.

[GO19] Manuel Gieseking and Ernst-Rüdiger Olderog. "High-level representation of benchmark families for Petri games". In: *CoRR* abs/1904.05621 (2019). arXiv: 1904.05621.

[GOW20] Manuel Gieseking, Ernst-Rüdiger Olderog, and Nick Würdemann. "Solving high-level Petri games". In: *Acta Informatica* 57.3-5 (2020), pp. 591–626. DOI: 10.1007/s00236-020-00368-5.

[GPVW95] Rob Gerth, Doron A. Peled, Moshe Y. Vardi, and Pierre Wolper. "Simple on-the-fly automatic verification of linear temporal logic". In: *Proceedings of PSTV*. IFIP Conference Proceedings. Vol. 38. Chapman & Hall, 1995, pp. 3–18.

[GR83] Ursula Goltz and Wolfgang Reisig. "The non-sequential behavior of Petri nets". In: *Inf. Control.* 57.2/3 (1983), pp. 125–147. DOI: 10.1016/S0019-9958(83)80040-0.

[Gre69] C. Cordell Green. "Application of theorem proving to problem solving". In: *Proceedings of IJCAI*. William Kaufmann, 1969, pp. 219–240.

[GS13] Paul Gastin and Nathalie Sznajder. "Fair synthesis for asynchronous distributed systems". In: *ACM Trans. Comput. Log.* 14.2 (2013), 9:1–9:31. DOI: 10.1145/2480759.2480761.

[Han19] Paul Hannibal. "Entscheidbarkeit von Petri-Spielen mit 2 Umgebungs- und 2 System-Spielern". German. Master's Thesis. University of Oldenburg, Oldenburg, Germany, 2019.

[HBS06] Marc Herbstritt, Bernd Becker, and Christoph Scholl. "Advanced SAT-techniques for bounded model checking of blackbox designs". In: *Proceedings of MTV*. IEEE Computer Society, 2006, pp. 37–44. DOI: 10.1109/MTV.2006.3.

[HCDR20] Qinheping Hu, John Cyphert, Loris D'Antoni, and Thomas W. Reps. "Exact and approximate methods for proving unrealizability of syntax-guided synthesis problems". In: *Proceedings of PLDI*. ACM, 2020, pp. 1128–1142. DOI: 10.1145/3385412.3385979.

[HD05] Paul Hunter and Anuj Dawar. "Complexity bounds for regular games". In: *Proceedings of MFCS*. Lecture Notes in Computer Science. Vol. 3618. Springer, 2005, pp. 495–506. DOI: 10.1007/11549345_43.

[HD18] Qinheping Hu and Loris D'Antoni. "Syntax-guided synthesis with quantitative syntactic objectives". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 10981. Springer, 2018, pp. 386–403. DOI: 10.1007/978-3-319-96145-3_21.

[Hel02] Keijo Heljanko. "Combining symbolic and partial order methods for model checking 1-safe Petri nets". PhD thesis. Aalto University, Helsinki, Finland, 2002.

[Hel99] Keijo Heljanko. "Using logic programs with stable model semantics to solve deadlock and reachability problems for 1-safe Petri nets". In: *Fundam. Inform.* 37.3 (1999), pp. 247–268. DOI: 10.3233/FI-1999-37304.

[HJS19] Marijn J. H. Heule, Matti Järvisalo, and Martin Suda. "SAT competition 2018". In: *J. Satisf. Boolean Model. Comput.* 11.1 (2019), pp. 133–154. DOI: 10.3233/SAT190120.

[HM19a] Jesko Hecking-Harbusch and Niklas Metzger. "BoundedAdam: Efficient Trace Encodings for Bounded Synthesis of Petri Games". In: *Figshare* (2019). DOI: 10.6084/m9.figshare.8313215.

[HM19b] Jesko Hecking-Harbusch and Niklas Metzger. "Efficient trace encodings of bounded synthesis for asynchronous distributed systems". In: *Proceedings of ATVA*. Lecture Notes in Computer Science. Vol. 11781. Springer, 2019, pp. 369–386. DOI: 10.1007/978-3-030-31784-3_22.

[Hor08] Florian Horn. "Explicit Muller games are PTIME". In: *Proceedings of FSTTCS*. LIPIcs. Vol. 2. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2008, pp. 235–243. DOI: 10.4230/LIPIcs.FSTTCS.2008.1756.

[HP84] David Harel and Amir Pnueli. "On the development of reactive systems". In: *Proceedings of Logics and Models of Concurrent Systems*. NATO ASI Series. Vol. 13. Springer, 1984, pp. 477–498. DOI: 10.1007/978-3-642-82453-1_17.

[HR04] Michael Huth and Mark Dermot Ryan. *Logic in computer science - modelling and reasoning about systems (2. ed.)* Cambridge University Press, 2004.

[HR72] R. Hossley and Charles Rackoff. "The emptiness problem for automata on infinite trees". In: *Proceedings of SWAT (FOCS)*. IEEE Computer Society, 1972, pp. 121–124. DOI: 10.1109/SWAT.1972.28.

[HT18] Jesko Hecking-Harbusch and Leander Tentrup. "Solving QBF by abstraction". In: *Proceedings of GandALF*. EPTCS. Vol. 277. 2018, pp. 88–102. DOI: 10.4204/EPTCS.277.7.

[Hu+19] Qinheping Hu, Jason Breck, John Cyphert, Loris D'Antoni, and Thomas W. Reps. "Proving unrealizability for syntax-guided synthesis". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 11561. Springer, 2019, pp. 335–352. DOI: 10.1007/978-3-030-25540-4_18.

[IP01] Russell Impagliazzo and Ramamohan Paturi. "On the complexity of k-SAT". In: *J. Comput. Syst. Sci.* 62.2 (2001), pp. 367–375. DOI: 10.1006/jcss.2000.1727.

[Jac+15] Swen Jacobs, Roderick Bloem, Romain Brenguier, Robert Könighofer, Guillermo A. Pérez, Jean-François Raskin, Leonid Ryzhyk, Ocan Sankur, Martina Seidl, Leander Tentrup, and Adam Walker. "The second reactive synthesis competition (SYNTCOMP 2015)". In: *Proceedings of SYNT@CAV*. EPTCS. Vol. 202. 2015, pp. 27–57. DOI: 10.4204/EPTCS.202.4.

[Jac+16] Swen Jacobs, Roderick Bloem, Romain Brenguier, Ayrat Khalimov, Felix Klein, Robert Könighofer, Jens Kreber, Alexander Legg, Nina Narodytska, Guillermo A. Pérez, Jean-François Raskin, Leonid Ryzhyk, Ocan Sankur, Martina Seidl, Leander Tentrup, and Adam Walker. "The 3rd reactive synthesis competition (SYNTCOMP 2016): benchmarks, participants & results". In: *Proceedings of SYNT@CAV*. EPTCS. Vol. 229. 2016, pp. 149–177. DOI: 10.4204/EPTCS.229.12.

[Jac+17a] Swen Jacobs, Nicolas Basset, Roderick Bloem, Romain Brenguier, Maximilien Colange, Peter Faymonville, Bernd Finkbeiner, Ayrat Khalimov, Felix Klein, Thibaud Michaud, Guillermo A. Pérez, Jean-François Raskin, Ocan Sankur, and Leander Tentrup. "The 4th reactive synthesis competition (SYNTCOMP 2017): benchmarks, participants & results". In: *Proceedings of SYNT@CAV*. EPTCS. Vol. 260. 2017, pp. 116–143. DOI: 10.4204/EPTCS.260.10.

[Jac+17b] Swen Jacobs, Roderick Bloem, Romain Brenguier, Rüdiger Ehlers, Timotheus Hell, Robert Könighofer, Guillermo A. Pérez, Jean-François Raskin, Leonid Ryzhyk, Ocan Sankur, Martina Seidl, Leander Tentrup, and Adam Walker. "The first reactive synthesis competition (SYNTCOMP 2014)". In: *Int. J. Softw. Tools Technol. Transf.* 19.3 (2017), pp. 367–390. DOI: 10.1007/s10009-016-0416-3.

[Jac+19] Swen Jacobs, Roderick Bloem, Maximilien Colange, Peter Faymonville, Bernd Finkbeiner, Ayrat Khalimov, Felix Klein, Michael Luttenberger, Philipp J. Meyer, Thibaud Michaud, Mouhammad Sakr, Salomon Sickert, Leander Tentrup, and Adam Walker. "The 5th reactive synthesis competition (SYNTCOMP 2018): benchmarks, participants & results". In: *CoRR* abs/1904.07736 (2019). arXiv: 1904.07736.

[JB16] Swen Jacobs and Roderick Bloem. "The reactive synthesis competition: SYNTCOMP 2016 and beyond". In: *Proceedings of SYNT@CAV*. EPTCS. Vol. 229. 2016, pp. 133–148. DOI: 10.4204/EPTCS.229.11.

[Jen13] Kurt Jensen. *Coloured Petri nets: basic concepts, analysis methods and practical use*. Vol. 2. Springer, 2013.

[Jen92] Kurt Jensen. *Coloured Petri nets - basic concepts, analysis methods and practical use - Volume 1*. EATCS Monographs on Theoretical Computer Science. Springer, 1992. DOI: 10.1007/978-3-662-06289-0.

[JGST10] Susmit Jha, Sumit Gulwani, Sanjit A. Seshia, and Ashish Tiwari. "Oracle-guided component-based program synthesis". In: *Proceedings of ICSE*. ACM, 2010, pp. 215–224. DOI: 10.1145/1806799.1806833.

[JGWB07] Barbara Jobstmann, Stefan J. Galler, Martin Weiglhofer, and Roderick Bloem. "Anzu: a tool for property synthesis". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 4590. Springer, 2007, pp. 258–262. DOI: 10.1007/978-3-540-73368-3_29.

[JLS16] Peter Gjøl Jensen, Kim Guldstrand Larsen, and Jirí Srba. "Real-time strategy synthesis for timed-arc Petri net games via discretization". In: *Proceedings of SPIN*. Lecture Notes in Computer Science. Vol. 9641. Springer, 2016, pp. 129–146. DOI: 10.1007/978-3-319-32582-8_9.

[JLS18] Peter Gjøl Jensen, Kim Guldstrand Larsen, and Jirí Srba. "Discrete and continuous strategies for timed-arc Petri net games". In: *Int. J. Softw. Tools Technol. Transf.* 20.5 (2018), pp. 529–546. DOI: 10.1007/s10009-017-0473-2.

[Job07] Barbara Jobstmann. "Applications and optimizations for LTL synthesis". PhD thesis. Graz University of Technology, Graz, Austria, 2007.

[JRLD07] Jan Jakob Jessen, Jacob Illum Rasmussen, Kim Guldstrand Larsen, and Alexandre David. "Guided controller synthesis for climate controller using UPPAAL TIGA". In: *Proceedings of FORMATS*. Lecture Notes in Computer Science. Vol. 4763. Springer, 2007, pp. 227–240. DOI: 10.1007/978-3-540-75454-1_17.

[JS20] Swen Jacobs and Mouhammad Sakr. "A symbolic algorithm for lazy synthesis of eager strategies". In: *Acta Informatica* 57.1-2 (2020), pp. 81–106. DOI: 10.1007/s00236-019-00344-8.

[JS97] Roberto J. Bayardo Jr. and Robert Schrag. "Using CSP look-back techniques to solve real-world SAT instances". In: *Proceedings of AAAI*. AAAI Press / The MIT Press, 1997, pp. 203–208.

[JTZ18] Swen Jacobs, Leander Tentrup, and Martin Zimmermann. "Distributed synthesis for parameterized temporal logics". In: *Inf. Comput.* 262 (2018), pp. 311–328. DOI: 10.1016/j.ic.2018.09.009.

[Jur98] Marcin Jurdzinski. "Deciding the winner in parity games is in UP ∩ co-UP". In: *Inf. Process. Lett.* 68.3 (1998), pp. 119–124. DOI: 10.1016/S0020-0190(98)00150-1.

[KB17] Ayrat Khalimov and Roderick Bloem. "Bounded synthesis for Streett, Rabin, and CTL*". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 10427. Springer, 2017, pp. 333–352. DOI: 10.1007/978-3-319-63390-9_18.

[KK19] Ayrat Khalimov and Orna Kupferman. "Register-bounded synthesis". In: *Proceedings of CONCUR*. LIPIcs. Vol. 140. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 25:1–25:16. DOI: 10.4230/LIPIcs.CONCUR.2019.25.

[KKV03] Victor Khomenko, Maciej Koutny, and Walter Vogler. "Canonical prefixes of Petri net unfoldings". In: *Acta Inf.* 40.2 (2003), pp. 95–118. DOI: 10.1007/s00236-003-0122-y.

[KLVY11] Orna Kupferman, Yoad Lustig, Moshe Y. Vardi, and Mihalis Yannakakis. "Temporal synthesis for bounded systems and environments". In: *Proceedings of STACS*. LIPIcs. Vol. 9. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011, pp. 615–626. DOI: 10.4230/LIPIcs.STACS.2011.615.

[KMB18] Ayrat Khalimov, Benedikt Maderbacher, and Roderick Bloem. "Bounded synthesis of register transducers". In: *Proceedings of ATVA*. Lecture Notes in Computer Science. Vol. 11138. Springer, 2018, pp. 494–510. DOI: 10.1007/978-3-030-01090-4_29.

[KMPS12] Viktor Kuncak, Mikaël Mayer, Ruzica Piskac, and Philippe Suter. "Software synthesis procedures". In: *Commun. ACM* 55.2 (2012), pp. 103–111. DOI: 10.1145/2076450.2076472.

[KPP12] Uri Klein, Nir Piterman, and Amir Pnueli. "Effective synthesis of asynchronous systems from GR(1) specifications". In: *Proceedings of VMCAI*. Lecture Notes in Computer Science. Vol. 7148. Springer, 2012, pp. 283–298. DOI: 10.1007/978-3-642-27940-9_19.

[KPS11] Gal Katz, Doron A. Peled, and Sven Schewe. "Synthesis of distributed control through knowledge accumulation". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 6806. Springer, 2011, pp. 510–525. DOI: 10.1007/978-3-642-22110-1_41.

[KPV09] Orna Kupferman, Nir Piterman, and Moshe Y. Vardi. "From liveness to promptness". In: *Formal Methods Syst. Des.* 34.2 (2009), pp. 83–103. DOI: 10.1007/s10703-009-0067-z.

[Kup12] Orna Kupferman. "Recent challenges and ideas in temporal synthesis". In: *Proceedings of SOFSEM*. Lecture Notes in Computer Science. Vol. 7147. Springer, 2012, pp. 88–98. DOI: 10.1007/978-3-642-27660-6_8.

[KV00] Orna Kupferman and Moshe Y. Vardi. "Synthesis with incomplete informatio". In: *Advances in temporal logic*. Springer, 2000, pp. 109–127. DOI: 10.1007/978-94-015-9586-5_6.

[KV01] Orna Kupferman and Moshe Y. Vardi. "Synthesizing distributed systems". In: *Proceedings of LICS*. IEEE Computer Society, 2001, pp. 389–398. DOI: 10.1109/LICS.2001.932514.

[KV05] Orna Kupferman and Moshe Y. Vardi. "Safraless decision procedures". In: *Proceedings of FOCS*. IEEE Computer Society, 2005, pp. 531–542. DOI: 10.1109/SFCS.2005.66.

[LCMT18] Kim Guldstrand Larsen, Adrien Le Coënt, Marius Mikucionis, and Jakob Haahr Taankvist. "Guaranteed control synthesis for continuous systems in UPPAAL TIGA". In: *Proceedings of CyPhy*. Lecture Notes in Computer Science. Vol. 11615. Springer, 2018, pp. 113–133. DOI: 10.1007/978-3-030-23703-5_6.

[Les95] Helmut Lescow. "On polynomial-size programs winning finite-state games". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 939. Springer, 1995, pp. 239–252. DOI: 10.1007/3-540-60045-0_54.

[LMS20] Michael Luttenberger, Philipp J. Meyer, and Salomon Sickert. "Practical synthesis of reactive systems from LTL specifications via parity games". In: *Acta Informatica* 57.1-2 (2020), pp. 3–36. DOI: 10.1007/s00236-019-00349-3.

[Mad11] P. Madhusudan. "Synthesizing reactive programs". In: *Proceedings of CSL*. LIPIcs. Vol. 12. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2011, pp. 428–442. DOI: 10.4230/LIPIcs.CSL.2011.428.

[MB08] Leonardo Mendonça de Moura and Nikolaj Bjørner. "Z3: an efficient SMT solver". In: *Proceedings of TACAS*. Lecture Notes in Computer Science. Vol. 4963. Springer, 2008, pp. 337–340. DOI: 10.1007/978-3-540-78800-3_24.

[MC18] Thibaud Michaud and Maximilien Colange. "Reactive synthesis from LTL specification with Spot". In: *Proceedings of SYNT@CAV*. 2018.

[McM03] Kenneth L. McMillan. "Craig interpolation and reachability analysis". In: *Proceedings of SAS*. Lecture Notes in Computer Science. Vol. 2694. Springer, 2003, p. 336. DOI: 10.1007/3-540-44898-5_18.

[McM95] Kenneth L. McMillan. "A technique of state space search based on unfolding". In: *Formal Methods Syst. Des.* 6.1 (1995), pp. 45–65. DOI: 10.1007/BF01384314.

[McN66] Robert McNaughton. "Testing and generating infinite sequences by a finite automaton". In: *Inf. Control.* 9.5 (1966), pp. 521–530. DOI: 10.1016/S0019-9958(66)80013-X.

[Met17] Niklas Metzger. "Bounded synthesis of Petri games with true concurrency semantics". Bachelor's Thesis. Saarland University, Saarbrücken, Germany, 2017.

[MMS96] José Meseguer, Ugo Montanari, and Vladimiro Sassone. "Process versus unfolding semantics for place/transition Petri nets". In: *Theor. Comput. Sci.* 153.1&2 (1996), pp. 171–210. DOI: 10.1016/0304-3975(95)00121-2.

[MMSZ20] Rupak Majumdar, Kaushik Mallik, Anne-Kathrin Schmuck, and Damien Zufferey. "Assume-guarantee distributed synthesis". In: *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.* 39.11 (2020), pp. 3215–3226. DOI: 10.1109/TCAD.2020.3012641.

[Mor+20] Kairo Morton, William T. Hallahan, Elven Shum, Ruzica Piskac, and Mark Santolucito. "Grammar filtering for syntax-guided synthesis". In: *Proceedings of AAAI*. AAAI Press, 2020, pp. 1611–1618.

[Mos84] Andrzej Wlodzimierz Mostowski. "Regular expressions for infinite trees and a standard form of automata". In: *Proceedings of SCT*. Lecture Notes in Computer Science. Vol. 208. Springer, 1984, pp. 157–168. DOI: 10.1007/3-540-16066-3_15.

[MP09] José Vander Meulen and Charles Pecheur. "Combining partial order reduction with bounded model checking". In: *Proceedings of CPA*. Concurrent Systems Engineering Series. Vol. 67. IOS Press, 2009, pp. 29–48. DOI: 10.3233/978-1-60750-065-0-29.

[MPS95] Oded Maler, Amir Pnueli, and Joseph Sifakis. "On the synthesis of discrete controllers for timed systems (An Extended Abstract)". In: *Proceedings of STACS*. Lecture Notes in Computer Science. Vol. 900. Springer, 1995, pp. 229–242. DOI: 10.1007/3-540-59042-0_76.

[MT01] P. Madhusudan and P. S. Thiagarajan. "Distributed controller synthesis for local specifications". In: *Proceedings of ICALP*. Vol. 2076. Lecture Notes in Computer Science. Vol. 2076. Springer, 2001, pp. 396–407. DOI: 10.1007/3-540-48224-5_33.

[MT02] P. Madhusudan and P. S. Thiagarajan. "A decidable class of asynchronous distributed controllers". In: *Proceedings of CONCUR*. Lecture Notes in Computer Science. Vol. 2421. Springer, 2002, pp. 145–160. DOI: 10.1007/3-540-45694-5_11.

[MTY05] P. Madhusudan, P. S. Thiagarajan, and Shaofa Yang. "The MSO theory of connectedly communicating processes". In: *Proceedings of FSTTCS*. Lecture Notes in Computer Science. Vol. 3821. Springer, 2005, pp. 201–212. DOI: 10.1007/11590156_16.

[Mus15] Anca Muscholl. "Automated synthesis of distributed controllers". In: *Proceedings of ICALP*. Lecture Notes in Computer Science. Vol. 9135. Springer, 2015, pp. 11–27. DOI: 10.1007/978-3-662-47666-6_2.

[MW03] Swarup Mohalik and Igor Walukiewicz. "Distributed games". In: *Proceedings of FSTTCS*. Lecture Notes in Computer Science. Vol. 2914. Springer, 2003, pp. 338–351. DOI: 10.1007/978-3-540-24597-1_29.

[MW14] Anca Muscholl and Igor Walukiewicz. "Distributed synthesis for acyclic architectures". In: *Proceedings of FSTTCS*. LIPIcs. Vol. 29. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2014, pp. 639–651. DOI: 10.4230/LIPIcs.FSTTCS.2014.639.

[MW80] Zohar Manna and Richard J. Waldinger. "A deductive approach to program synthesis". In: *ACM Trans. Program. Lang. Syst.* 2.1 (1980), pp. 90–121. DOI: 10.1145/357084.357090.

[MWZ09] Anca Muscholl, Igor Walukiewicz, and Marc Zeitoun. "A look at the control of asynchronous automata". In: *Perspect. Concurr. Theory* (2009), pp. 356–371.

[Nea15] Turlough Neary. "Undecidability in binary tag systems and the post correspondence problem for five pairs of words". In: *Proceedings of STACS*. LIPIcs. Vol. 30. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015, pp. 649–661. DOI: 10.4230/LIPIcs.STACS.2015.649.

[NPW81] Mogens Nielsen, Gordon D. Plotkin, and Glynn Winskel. "Petri nets, event structures and domains, Part I". In: *Theor. Comput. Sci.* 13 (1981), pp. 85–108. DOI: 10.1016/0304-3975(81)90112-2.

[OH86] Ernst-Rüdiger Olderog and C. A. R. Hoare. "Specification-oriented semantics for communicating processes". In: *Acta Informatica* 23.1 (1986), pp. 9–66. DOI: 10.1007/BF00268075.

[Old91] Ernst-Rüdiger Olderog. *Nets, terms and formulas: three views of concurrent processes and their relationship*. Cambridge University Press, 1991, p. 267. DOI: 10.1017/CBO9780511526589.

[PF12] Hans-Jörg Peter and Bernd Finkbeiner. "The complexity of bounded synthesis for timed control with partial observability". In: *Proceedings of FORMATS*. Lecture Notes in Computer Science. Vol. 7595. Springer, 2012, pp. 204–219. DOI: 10.1007/978-3-642-33365-1_15.

[PG86] David A. Plaisted and Steven Greenbaum. "A structure-preserving clause form translation". In: *J. Symb. Comput.* 2.3 (1986), pp. 293–304. DOI: 10.1016/S0747-7171(86)80028-1.

[Pin07] Sophie Pinchinat. "A generic constructive solution for concurrent games with expressive constraints on strategies". In: *Proceedings of ATVA*. Lecture Notes in Computer Science. Vol. 4762. Springer, 2007, pp. 253–267. DOI: 10.1007/978-3-540-75596-8_19.

[PMNS19] Saswat Padhi, Todd D. Millstein, Aditya V. Nori, and Rahul Sharma. "Overfitting in synthesis: theory and practice". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 11561. Springer, 2019, pp. 315–334. DOI: 10.1007/978-3-030-25540-4_17.

[Pnu77] Amir Pnueli. "The temporal logic of programs". In: *Proceedings of FOCS*. IEEE Computer Society, 1977, pp. 46–57. DOI: 10.1109/SFCS.1977.32.

[Pos46] Emil L. Post. "A variant of a recursively unsolvable problem". In: *Bull. Am. Math. Soc.* 52.4 (1946), pp. 264–268. DOI: 10.1090/S0002-9904-1946-08555-9.

[PR11] Andreas Podelski and Andrey Rybalchenko. "Transition invariants and transition predicate abstraction for program termination". In: *Proceedings of TACAS*. Lecture Notes in Computer Science. Vol. 6605. Springer, 2011, pp. 3–10. DOI: 10.1007/978-3-642-19835-9_2.

[PR89a] Amir Pnueli and Roni Rosner. "On the synthesis of a reactive module". In: *Proceedings of POPL*. ACM Press, 1989, pp. 179–190. DOI: 10.1145/75277.75293.

[PR89b] Amir Pnueli and Roni Rosner. "On the synthesis of an asynchronous reactive module". In: *Proceedings of ICALP*. Lecture Notes in Computer Science. Vol. 372. Springer, 1989, pp. 652–671. DOI: 10.1007/BFb0035790.

[PR90] Amir Pnueli and Roni Rosner. "Distributed reactive systems are hard to synthesize". In: *Proceedings of FOCS*. IEEE Computer Society, 1990, pp. 746–757. DOI: 10.1109/FSCS.1990.89597.

[PSM16] Saswat Padhi, Rahul Sharma, and Todd D. Millstein. "Data-driven precondition inference with learned features". In: *Proceedings of PLDI*. ACM, 2016, pp. 42–56. DOI: 10.1145/2908080.2908099.

[PSM17] Saswat Padhi, Rahul Sharma, and Todd D. Millstein. "LoopInvGen: a loop invariant generator based on precondition inference". In: *CoRR* abs/1707.02029 (2017). arXiv: 1707.02029.

[QS82] Jean-Pierre Queille and Joseph Sifakis. "Specification and verification of concurrent systems in CESAR". In: *Proceedings of International Symposium on Programming*. Lecture Notes in Computer Science. Vol. 137. Springer, 1982, pp. 337–351. DOI: 10.1007/3-540-11494-7_22.

[Rab69] Michael O. Rabin. "Decidability of second-order theories and automata on infinite trees". In: *Trans. Am. Math. Soc.* 141 (1969), pp. 1–35.

[Rab72] Michael O. Rabin. *Automata on infinite objects and Church's problem*. 1972. DOI: 10.1090/cbms/013.

[RBLT20] Andrew Reynolds, Haniel Barbosa, Daniel Larraz, and Cesare Tinelli. "Scalable algorithms for abduction via enumerative syntax-guided synthesis". In: *Proceedings of IJCAR*. Lecture Notes in Computer Science. Vol. 12166. Springer, 2020, pp. 141–160. DOI: 10.1007/978-3-030-51074-9_9.

[Rei85] Wolfgang Reisig. *Petri nets: an introduction*. EATCS Monographs on Theoretical Computer Science. Vol. 4. Springer, 1985. DOI: 10.1007/978-3-642-69968-9.

[Rey+19] Andrew Reynolds, Haniel Barbosa, Andres Nötzli, Clark W. Barrett, and Cesare Tinelli. "CVC4SY: smart and fast term enumeration for syntax-guided synthesis". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 11562. Springer, 2019, pp. 74–83. DOI: 10.1007/978-3-030-25543-5_5.

[Roz16] Kristin Yvonne Rozier. "Specification: the biggest bottleneck in formal methods and autonomy". In: *Proceedings of VSTTE*. Lecture Notes in Computer Science. Vol. 9971. 2016, pp. 8–26. DOI: 10.1007/978-3-319-48869-1_2.

[RSVB03] Jean-François Raskin, Mathias Samuelides, and Laurent Van Begin. *Petri games are monotonic but difficult to decide*. Tech. rep. 2003.

[RT17] Andrew Reynolds and Cesare Tinelli. "SyGuS techniques in the core of an SMT solver". In: *Proceedings of SYNT@CAV*. EPTCS. Vol. 260. 2017, pp. 81–96. DOI: 10.4204/EPTCS.260.8.

[Saf88] Shmuel Safra. "On the complexity of ω-automata". In: *Proceedings of FOCS*. IEEE Computer Society, 1988, pp. 319–327. DOI: 10.1109/SFCS.1988.21948.

[SB00] Fabio Somenzi and Roderick Bloem. "Efficient Büchi automata from LTL formulae". In: *Proceedings of CAV*. Lecture Notes in Computer Science. Vol. 1855. Springer, 2000, pp. 248–263. DOI: 10.1007/10722167_21.

[Sch14] Sven Schewe. "Distributed synthesis is simply undecidable". In: *Inf. Process. Lett.* 114.4 (2014), pp. 203–207. DOI: 10.1016/j.ipl.2013.11.012.

[Sch19] Sanny Schmitt. "Generating concurrency-preserving Petri games". Bachelor's Thesis. Saarland University, Saarbrücken, Germany, 2019.

[SF06] Sven Schewe and Bernd Finkbeiner. "Synthesis of asynchronous systems". In: *Proceedings of LOPSTR*. Lecture Notes in Computer Science. Vol. 4407. Springer, 2006, pp. 127–142. DOI: 10.1007/978-3-540-71410-1_10.

[SF07] Sven Schewe and Bernd Finkbeiner. "Bounded synthesis". In: *Proceedings of ATVA*. Lecture Notes in Computer Science. Vol. 4762. Springer, 2007, pp. 474–488. DOI: 10.1007/978-3-540-75596-8_33.

[Si+18] Xujie Si, Woosuk Lee, Richard Zhang, Aws Albarghouthi, Paraschos Koutris, and Mayur Naik. "Syntax-guided synthesis of Datalog programs". In: *Proceedings of FSE*. ACM, 2018, pp. 515–527. DOI: 10.1145/3236024.3236034.

[SN07] Alberto L. Sangiovanni-Vincentelli and Marco Di Natale. "Embedded system design for automotive applications". In: *Computer* 40.10 (2007), pp. 42–51. DOI: 10.1109/MC.2007.344.

[Sol+06] Armando Solar-Lezama, Liviu Tancau, Rastislav Bodík, Sanjit A. Seshia, and Vijay A. Saraswat. "Combinatorial sketching for finite programs". In: *Proceedings of ASPLOS*. ACM, 2006, pp. 404–415. DOI: 10.1145/1168857.1168907.

[Sol13] Armando Solar-Lezama. "Program sketching". In: *Int. J. Softw. Tools Technol. Transf.* 15.5-6 (2013), pp. 475–495. DOI: 10.1007/s10009-012-0249-7.

[Spr15] Valentin Spreckels. "Petri-Spiele: Erweiterung der Sicherheitsgewinnbedingung auf Markierungen". German. Master's Thesis. University of Oldenburg, Oldenburg, Germany, 2015.

[SRBE05] Armando Solar-Lezama, Rodric M. Rabbah, Rastislav Bodík, and Kemal Ebcioglu. "Programming by sketching for bit-streaming programs". In: *Proceedings of PLDI*. ACM, 2005, pp. 281–294. DOI: 10.1145/1065010.1065045.

[SS99] João P. Marques Silva and Karem A. Sakallah. "GRASP: a search algorithm for propositional satisfiability". In: *IEEE Trans. Computers* 48.5 (1999), pp. 506–521. DOI: 10.1109/12.769433.

[SSA13] Eric Schkufza, Rahul Sharma, and Alex Aiken. "Stochastic superoptimization". In: *Proceedings of ASPLOS*. ACM, 2013, pp. 305–316. DOI: 10.1145/2451116.2451150.

[Sta85] Eugene W. Stark. "A proof technique for rely/guarantee properties". In: *Proceedings of FSTTCS*. Lecture Notes in Computer Science. Vol. 206. Springer, 1985, pp. 369–391. DOI: 10.1007/3-540-16042-6_21.

[TDB16] Petar Tsankov, Mohammad Torabi Dashti, and David A. Basin. "Access control synthesis for physical spaces". In: *Proceedings of CSF*. IEEE Computer Society, 2016, pp. 443–457. DOI: 10.1109/CSF.2016.38.

[Ten16] Leander Tentrup. "Non-prenex QBF solving using abstraction". In: *Proceedings of SAT*. Lecture Notes in Computer Science. Vol. 9710. Springer, 2016, pp. 393–401. DOI: 10.1007/978-3-319-40970-2_24.

[Ten19] Leander Tentrup. "CAQE and QuAbS: abstraction based QBF solvers". In: *J. Satisf. Boolean Model. Comput.* 11.1 (2019), pp. 155–210. DOI: 10.3233/SAT190121.

[Tho09] Wolfgang Thomas. "Facets of synthesis: revisiting Church's problem". In: *Proceedings of FoSSaCS*. Lecture Notes in Computer Science. Vol. 5504. Springer, 2009, pp. 1–14. DOI: 10.1007/978-3-642-00596-1_1.

[TR19] Leander Tentrup and Markus N. Rabe. "Clausal abstraction for DQBF". In: *Proceedings of SAT*. Lecture Notes in Computer Science. Vol. 11628. Springer, 2019, pp. 388–405. DOI: 10.1007/978-3-030-24258-9_27.

[TS07] Andrew S. Tanenbaum and Maarten van Steen. *Distributed systems - principles and paradigms, 2nd edition*. Pearson Education, 2007. ISBN: 978-0-13-239227-3.

[Tse83] Grigori S. Tseitin. "On the Complexity of Derivation in Propositional Calculus". In: *Automation of Reasoning: 2: Classical Papers on Computational Logic 1967–1970*. Springer, 1983, pp. 466–483. DOI: 10.1007/978-3-642-81955-1_28.

[Tur37] Alan Mathison Turing. "On computable numbers, with an application to the Entscheidungsproblem". In: *Proc. Lond. Math. Soc.* 2.1 (1937), pp. 230–265.

[Udu+13] Abhishek Udupa, Arun Raghavan, Jyotirmoy V. Deshmukh, Sela Mador-Haim, Milo M. K. Martin, and Rajeev Alur. "TRANSIT: specifying protocols with concolic snippets". In: *Proceedings of PLDI*. ACM, 2013, pp. 287–296. DOI: 10.1145/2491956.2462174.

[VW21] Nikhil Vyas and R. Ryan Williams. "On super strong ETH". In: *J. Artif. Intell. Res.* 70 (2021), pp. 473–495. DOI: 10.1613/jair.1.11859.

[VW86] Moshe Y. Vardi and Pierre Wolper. "Automata-theoretic techniques for modal logics of programs". In: *J. Comput. Syst. Sci.* 32.2 (1986), pp. 183–221. DOI: 10.1016/0022-0000(86)90026-7.

[VW94] Moshe Y. Vardi and Pierre Wolper. "Reasoning about infinite computations". In: *Inf. Comput.* 115.1 (1994), pp. 1–37. DOI: 10.1006/inco.1994.1092.

[Web+19] Tjark Weber, Sylvain Conchon, David Déharbe, Matthias Heizmann, Aina Niemetz, and Giles Reger. "The SMT competition 2015-2018". In: *J. Satisf. Boolean Model. Comput.* 11.1 (2019), pp. 221–259. DOI: 10.3233/SAT190123.

[Zie87] Wieslaw Zielonka. "Notes on finite asynchronous automata". In: *RAIRO Theor. Informatics Appl.* 21.2 (1987), pp. 99–135. DOI: 10.1051/ita/1987210200991.