Carl von Ossietzky Universität Oldenburg
Fakultät II – Informatik, Wirtschafts- und Rechtswissenschaften
Department für Informatik

Synthesis of Asynchronous Distributed Systems with Causal Memory via Petri Games

Von der Fakultät für Informatik, Wirtschafts- und Rechtswissenschaften der Carl von Ossietzky Universität Oldenburg zur Erlangung des Grades und Titels

Doktor der Naturwissenschaften (Dr. rer. nat.)

angenommene Dissertation

von Paul Jonathan Hannibal
geboren am 07.03.1995 in Oldenburg

Gutachter:	Prof. Dr. Ernst-Rüdiger Olderog Prof. Dr. Bernd Finkbeiner
Tag der Disputation:	05.12.2025

Abstract

The number of safety critical information systems in our daily lives continues to grow. Among others, these include systems for traffic and transport, medical systems, energy supply, and emergency communications. We heavily rely on such systems to function properly at any time. Those systems are thoroughly tested to detect possible flaws that might lead to dangerous situations. However, this does not guarantee the absence of flaws. Testing all cases requires enormous efforts or is even impossible. Here, applying formal verification techniques can greatly improve the reliability and safety guarantees.

Some formal verification techniques enhance testing efficiency, while others, like model checking, analyze the entire state space. The *synthesis approach* belongs to the latter and aims at automatically generating a system which is correct by construction. To synthesize a system, it is necessary to specify its desired behaviour beforehand. This is a challenging task. However, the error-prone task of implementing is automated.

Synthesis is especially hard for distributed systems which consist of several independent parts that act and interact concurrently. Each part has its local state and its local, incomplete view of the state of the system while the specification often concerns the joint local states, the global state of the system.

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

In order to win the Petri game, the system players have to satisfy a global safety condition against all possible behaviours of the adversarial environment players. The distributed winning strategy of the system players is the desired implementation of the system. A Petri game might be equipped with other winning conditions.

We show that the synthesis problem for 2-player Petri games under a global safety condition is NP-complete and can be solved within a non-deterministic exponential upper bound in the case of up to 4 players. Furthermore, we show the undecidability of the synthesis problem for Petri games with at least 6 players under a local safety condition. At last, we show how to find winning strategies that use only a memory of bounded size which opens up Petri games to more complex winning conditions while keeping the synthesis problem decidable and which enables refinements of the memory model.

Zusammenfassung

Die Zahl der sicherheitskritischen Informationssysteme im täglichen Leben nimmt weiter zu. Dazu gehören unter anderem Systeme für Verkehr und Transport, medizinische Systeme, Energieversorgung und Notfallkommunikation. Wir verlassen uns in hohem Maße darauf, dass solche Systeme jederzeit ordnungsgemäß funktionieren. Diese Systeme werden gründlich getestet, um mögliche Schwachstellen zu erkennen, die zu gefährlichen Situationen führen könnten. Dies ist jedoch keine Garantie für die Abwesenheit von Fehlern. Das Testen aller Fälle erfordert einen enormen Aufwand oder ist sogar unmöglich. Hier kann die Anwendung formaler Verifikationstechniken die Zuverlässigkeits- und Sicherheitsgarantien erheblich verbessern.

Einige formale Verifikationstechniken verbessern die Testeffizienz, während andere, wie das Model Checking, den gesamten Zustandsraum analysieren. Der Syntheseansatz gehört zu den letzteren und zielt darauf ab, automatisch ein korrektes System zu generieren. Um ein System zu synthetisieren, muss sein gewünschtes Verhalten im Voraus spezifiziert werden. Dies ist eine anspruchsvolle Aufgabe. Die fehleranfällige Aufgabe der Implementierung wird jedoch automatisiert.

Die Synthese ist besonders schwierig bei verteilten Systemen, die aus mehreren unabhängigen Teilen bestehen, die gleichzeitig agieren und interagieren. Jeder Teil hat seinen lokalen Zustand und seine lokale, unvollständige Sicht auf den Zustand des Systems, während die Spezifikation oft die gemeinsamen lokalen Zustände, den globalen Zustand des Systems, betrifft.

Petri-Spiele werden verwendet, um verteilte Systeme zu spezifizieren. Es handelt sich dabei um ein Mehrspieler-Spielmodell für die Synthese von asynchronen, verteilten Systemen, bei dem die Spieler als Token auf einem Petri-Netz dargestellt und in in Umgebungsspieler und Systemspieler unterteilt werden. Solange sich die Spieler in unabhängigen Teilen des Netzes bewegen, wissen sie nicht voneinander; wenn sie sich bei einer gemeinsamen Transition synchronisieren, wird jeder Spieler über die gesamte kausale Geschichte der anderen Spieler informiert. Ein solches Speichermodell wird als *kausaler Speicher* bezeichnet.

Um das Petri-Spiel zu gewinnen, müssen die Systemspieler eine globale Sicherheitsbedingung gegen alle möglichen Verhaltensweisen der gegnerischen Umgebungsspieler erfüllen. Die verteilte Gewinnstrategie der Systemspieler ist die gewünschte Implementierung des Systems. Ein Petri-Spiel kann mit anderen Gewinnbedingungen ausgestattet sein.

Wir zeigen, dass das Syntheseproblem für Petri-Spiele mit 2 Spielern unter einer globalen Sicherheitsbedingung NP-vollständig ist und im Fall von bis zu 4 Spielern innerhalb einer nicht-deterministischen exponentiellen Zeitgrenze gelöst werden kann. Außerdem zeigen wir die Unentscheidbarkeit des Syntheseproblems für Petri-Spiele mit mindestens 6 Spielern unter einer lokalen Sicherheitsbedingung. Zuletzt zeigen wir, wie man Gewinnstrategien findet, die nur einen Speicher von begrenzter Größe verwenden, was Petri-Spiele für komplexere Gewinnbedingungen öffnet, während das Syntheseproblem entscheidbar bleibt, und was Verfeinerungen des Speichermodells ermöglicht.

Contents

Chapter 1

Introduction

The number of safety-critical information systems embedded in modern society continues to grow. Systems responsible for managing transportation networks, regulating medical equipment, supplying energy, or ensuring emergency communication availability are now integral to the infrastructure on which individuals and industries depend. Their failure can result in catastrophic consequences, ranging from financial losses to endangerment of human lives. Ensuring their correctness is thus not just desirable but imperative.

Traditionally, these systems undergo rigorous testing procedures designed to uncover and mitigate faults. However, exhaustive testing is infeasible due to the combinatorial explosion of system states, especially when dealing with concurrent and reactive behaviors. Testing can demonstrate the presence of faults, but not their absence. As a result, formal verification techniques have seen growing interest for their ability to mathematically prove properties of systems. Among these techniques, *model checking* and *reactive synthesis* are prominent. While *model checking* verifies whether an implementation satisfies a specification, *synthesis* extends this process: it automatically constructs an implementation that is correct by construction for a given specification. The synthesis problem was first described in [Chu62]. Both approaches check the whole state space of a system whether every possible state satisfies the specification. Characteristic for reactive synthesis is an *uncontrollable environment* to which the system needs to react, i.e. the environment determines the inputs of the reactive system. A synthesized system must satisfy the specification for all possible inputs.

In particular, the synthesis of *distributed* systems is both a highly relevant and notoriously difficult problem. Distributed systems are composed of multiple autonomous components that act independently and typically have only partial knowledge of the global system state. The components may communicate to gather information about the other components. These characteristics make the coordination and correctness guarantees of such systems especially challenging.

For *distributed* systems, we further distinguish between *synchronous* and *asynchronous* systems. Informally speaking, all components of a *synchronous distributed* system share a clock and proceed synchronously step by step. However, they still have only *partial knowledge* of the global system state. In [PR90], the authors establish the undecidability

of the synthesis problem for even simple synchronous concurrent architectures with temporal specifications and [FS05] identifies so-called *information forks* as a structural cause of undecidability in distributed architectures. Petri games are a model for the synthesis of *asynchronous distributed* systems [PR89, SF06]. Here, each component proceeds in its own rate independently from the rates of the other components. An asynchronous distributed system models a system more accurately if the components do not have a shared clock. For synthesis of hardware circuits a shared clock is usual. However, if the components are physically separated or for the synthesis of software an asynchronous distributed system appears to be the more suitable choice.

In contrast to synthesis of distributed systems, the synthesis for systems, which consist of a single component with *complete knowledge*, has seen significant positive decidability results [BL69].

The model of Petri games was introduced to address the difficulties of distributed synthesis [FO17]. An overview of the results achieved for Petri games can be found in [FO24]. Petri games provide a natural and powerful framework to represent concurrent behaviour and *partial knowledge*. In a Petri net, tokens move via transitions between places. Building on Petri nets, which are well-established for modelling distributed and concurrent systems, Petri games add a game-theoretic perspective by interpreting tokens as players. In a Petri game, the set of places is divided into a set of system places and a set of environment places. A token on a system place is a system player and a token on an environment place is an environment player. Characteristic for Petri games is the causal memory model: The players do not know of each other as long as they move in independent parts of the net. When they take a joint transition they exchange information about their complete causal history.

The aim in a Petri game is for the system players to act and cooperate to satisfy a given specification, typically expressed as a safety condition. A system player controls which transitions are allowed for its next step. A safety condition can be either local (avoidance of certain places by individual players) or global (avoidance of certain combinations of token positions). A strategy is composed of the local strategies of every system player. A local strategy describes for a system player which transitions are allowed depending on its whole causal history, i.e. all transitions the token has taken so far and all transitions it has got knowledge of by communicating with other tokens. A winning strategy for the system players is one that guarantees safety for all possible behaviours of the adversarial, uncontrollable environment players. Solving the synthesis problem thus involves checking the existence of such a strategy and constructing it when it exists.

The complexity and decidability of the synthesis problem in Petri games depend significantly on the number and types of players. Previous work has demonstrated that the problem is decidable and EXPTIME-complete for Petri games with a single environment player and a bounded number of system players under a local safety condition [FO17]. This has been extended to global safety conditions taking double exponential time (2-EXP) [FGHO22]. In [FG18], the inverted case with a single system player and a bounded number of environment players under a global safety condition is shown to be EXPTIME-complete.

In this work, we explore the synthesis problem for Petri games under new and extended settings. We present three main results:

- A non-deterministic exponential upper time bound for Petri games with up to four players which are arbitrarily distributed among system and environment, especially the previously uncovered case of two players each. Player conversions in both direction are allowed.
- Undecidability for 6-player Petri games equipped with a local safety condition.
- A finite state space that is consistent to a bounded part of the causal memory. This state space is used to find winning strategies with a bounded memory. This approach remains decidable even for more complex winning conditions such as linear temporal logic (LTL).

Furthermore, NP-completeness for two-player Petri games with a global safety condition is shown.

The undecidability result here is closely related to the undecidability result in [Gim22]. There, undecidability in the setting of asynchronous games played on Zielonka automata [Zie87] is shown for 6-player games equipped with a local termination condition. Roughly speaking the model of asynchronous games and Petri games are the same model. The main difference to Petri games is the distinction of controllable and uncontrollable transitions, called actions in an asynchronous game, instead of the distinction of controllable and uncontrollable places, called states in an asynchronous game. In [BFH19], Petri games with a local winning condition, i.e. safety or termination, are converted into asynchronous games and vice versa with an exponential blow up in the number of places and transitions, and in the number of states and actions, respectively. As a result, theoretical findings in one model can be transferred to the other, albeit with exponential overhead.

In [PR90], it is shown that the synthesis problem becomes undecidable even for simple concurrent architectures when the specification is given in linear temporal logic (LTL). The approach we propose avoids this source of undecidability by considering only a bounded part of the causal memory. As long as the specification is expressed in a logic for which model checking over Petri nets is decidable—such as LTL—our approach remains decidable. So, our approach is similar to existing approaches for bounded synthesis [FS13, HM19].

In [FS13], strategies are restricted to finite state automata of increasing size. In contrast, our method captures more precisely how a system player's memory state is derived from its causal history. We can specify which bounded part of the causal memory is the memory for the strategies. If two causal histories contain equivalent information in the bounded parts those causal histories are matched, i.e. any strategy is restricted to make the same decision if the bounded parts are equivalent. Compared to [FS13], the search for winning strategies here is more efficient if the bounded memory is suitable.

In [HM19], a previous approach to bounded synthesis for Petri games is presented. There, the existence of a winning strategy is encoded as a quantified Boolean formula (QBF), where the bound corresponds to number of causal histories which are distinguished. Comparing this method with ours on a set of parameterized benchmark families

of Petri games is encouraging for our approach. However, our approach suffers more due to the state space explosion. This might be caused by explicitly constructing the state space. We conjecture that existing approaches which tackle the state explosion problem for model checking like in [Val90, CGJ+00] are applicable to our approach.

Another line of work which could ease the state explosion problem are high-level Petri games [WCHP24, Wür24] and the exploitation of symmetries in high-level Petri games [Wür21].

1.1 Structure of this Thesis

Including the introduction, this thesis is divided into six chapters. In Chapter 2, we introduce mathematical notations and the mathematical foundations of Petri games.

In Chapter 3, we present the decidability results regarding the NEXP bound for 4-player Petri games and the NP-completeness for 2-player games. The algorithms presented to solve 2-player and 4-player Petri games, respectively, are an adaptation of the algorithm constructing a complete finite prefix of a Petri net presented in [ERV02]. There, a cut-off criterion is used to determine when a prefix is complete. We adapt the cut-off criterion to determine when a prefix of a strategy contains all possible behaviour of the players in a Petri game. So, if a prefix is winning in all cases we construct a winning strategy by repeating parts of the prefix. We introduce different cut-off criteria for the 2-player case and the 4-player case.

In Chapter 4, we show the undecidability of 6-player Petri games equipped with a local safety condition in a two-step reduction. First, we reduce the undecidable ω-Post correspondence problem (ω-PCP) [Fin15b, Gir86, Ruo85] to a tiling problem of an infinite plane. Then the tiling problem is reduced to the synthesis problem of 6-player Petri games.

In Chapter 5, we show how to construct a finite state space which keeps track of a bounded part of the possibly infinitely growing causal history. We reason that any bounded part of the causal history can be tracked. We show how to refine the causal memory model if we understand the memory of the players as a part of the specification. At last, we present experimental results where we use the finite state space which keeps track of the last known markings as the bounded part of the causal history to search for winning strategies. For that, the existence of a winning strategy is encoded as a boolean satisfiability problem (SAT). The results are compared on parameterized benchmark families of Petri games to [FGO15] and the approach of bounded synthesis in [HM19]. We understand our approach as bounded synthesis. However, the bound is not an increasing natural number for some parameters. Instead, the bound is defined by the bounded part of the causal history which is tracked.

1.2 List of Publications

This thesis is based on the following three publications.

1. Paul Hannibal and Ernst-Rüdiger Olderog. The synthesis problem for repeatedly communicating Petri games. In Luca Bernardinello and Laure Petrucci, editors, *Application and Theory of Petri Nets and Concurrency - 43rd International Conference, PETRI NETS 2022, Proceedings*, volume 13288 of *Lecture Notes in Computer Science*, pages 236–257. Springer, 2022
2. Paul Hannibal. (Un)decidability bounds of the synthesis problem for Petri games. In Antonis Achilleos and Dario Della Monica, editors, *Proceedings of the Fourteenth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2023, Udine, Italy, 18-20th September 2023*, volume 390 of *EPTCS*, pages 115–131, 2023
3. Paul Hannibal, Dennis Lisiecki, and Ernst-Rüdiger Olderog. Finite-memory strategies for Petri games. In Martin Fränzle, Jürgen Niehaus, and Bernd Westphal, editors, *Engineering Safe and Trustworthy Cyber Physical Systems: Essays Dedicated to Werner Damm on the Occasion of His 71st Birthday*, pages 183–200, Cham, 2026. Springer Nature Switzerland

The first publication shows the decidability of Petri games in exponential time (EXP) where the unfolding of the underlying net has a bound on the number of concurrent events which holds for all nodes. This result is not explicitly stated in this thesis. However, the results for 4-player Petri games subsume this result with slight adaptations.

The second publication contains the decidability results in Chapter 3 and the undecidability results in Chapter 4.

Finally, the third paper describes the construction of a finite state space of a Petri net which keeps track of the last known markings of each token. Afterwards, this is used to encode the existence of a winning strategy as a SAT-problem.

Chapter 2

Foundations

In this chapter, the foundations of the Petri game model are introduced. This includes branching processes and the unfolding of a Petri net as in [Eng91]. For reference, the reader is also referred to [EH08, BF88, Vog91]. Also, Petri games and their winning strategies are introduced as in [FO17].

2.1 Petri Nets and Branching Processes

Mathematical notation. The set of natural numbers throughout this thesis is $\mathbb{N} = \{1, 2, 3, \ldots\}$. The *power set* of a set A is denoted by $2^A = \{B \mid B \subseteq A\}$ and the *set of nonempty finite subsets* of A by $2^A_{nf} = \{B \mid B \subseteq A \text{ and } B \neq \emptyset \text{ and } B \text{ is finite.}\}$. For a function $f : X \to Y$ and a set $S \subseteq X$, we write $f(S)$ for the set $\{f(s) \mid s \in S\}$. The identity function is the function $id : X \to X$ such that $id(x) = x$ for all $x \in X$. A function $f : X \to Y$ restricted to $Z \subseteq X$ is the function $f|_Z : Z \to Y$ with $f|_Z(z) = f(z)$ for all $z \in Z$. A relation $R \subseteq X \times X$ restricted to $Z \subseteq X$ is the relation $R|_Z$ defined as $(x, y) \in R|_Z \Leftrightarrow x, y \in Z \wedge (x, y) \in R$. For functions $f : X \to Y$ and $g : Y' \to Z$ with $Y \subseteq Y'$, the *composition* $g \circ f : X \to Z$ (read: "g after f") maps each $x \in X$ to $g(f(x))$. For bijective functions $f : X \to Y$ the inverse function is f^{-1} such that $f \circ f^{-1} = id$ is the identity function on Y and $f^{-1} \circ f = id$ is the identity function on X.

A *multi-set* M over S is a function $M : S \to \mathbb{N}$. We also use the standard set operation symbols $\cup, \setminus$ and $\subseteq$ for the multi-set operations. The union of two multi-sets M_1 and M_2 over S is defined element-wise for $s \in S$ as $(M_1 \cup M_2)(s) = M_1(s) + M_2(s)$ and the disjunction is defined as $(M_1 \setminus M_2)(s) = \max\{M_1(s) - M_2(s), 0\}$. For two multi-sets M_1 and M_2 over S, we write $M_1 \subseteq M_2$ if $M_1(s) \leq M_2(s)$ holds for all $s \in S$. Every set S can be interpreted as a multi-set $S(s) = \begin{cases} 1 & s \in S \\ 0 & s \notin S \end{cases}$.

In the following, Petri nets are introduced. Since all Petri nets considered in this thesis are *safe* Petri nets, the definition of a Petri net is adapted to this fact in hindsight. The definition of a *safe* Petri net follows shortly after the definition of a Petri net.

Definition 2.1 (Petri net). A *Petri net* or *net* is a structure $N = (\mathcal{P}, \mathcal{T}, pre, post, In)$, where

1. $\mathcal{P}$ is the (possibly infinite) set of *places*,
2. $\mathcal{T}$ is the (possibly infinite) set of *transitions* that is disjoint to $\mathcal{P}$,
3. $pre : \mathcal{T} \to 2^{\mathcal{P}}_{nf}$ and $post : \mathcal{T} \to 2^{\mathcal{P}}_{nf}$,
4. and $In \subseteq \mathcal{P}$ is the initial marking.

$\lhd$

A more general definition of a Petri net that allows the preset and postset of a transition to be a multi-set can be found in [Rei85].

The flow mappings *pre* and *post* are extended to places as usual: $\forall p \in \mathcal{P} : pre(p) = \{t \in \mathcal{T} \mid p \in post(t)\}$ and $\forall p \in \mathcal{P} : post(p) = \{t \in \mathcal{T} \mid p \in pre(t)\}$. A *marking* M of a Petri net $\mathcal{N}$ is a multi-set over $\mathcal{P}$. In particular, In is a marking. By convention, a net named N has the components $(\mathcal{P}, \mathcal{T}, pre, post, In)$, and a net named N_0 has the components $(\mathcal{P}_0, \mathcal{T}_0, pre_0, post_0 In_0)$ and analogously for nets with decorated names like N_1, N_2, N_3.

A transition $t \in \mathcal{T}$ is *enabled* at marking M if $pre(t) \subseteq M$. If t is enabled, the transition t can be *fired*, such that the new marking is $M' = (M \setminus pre(t)) \cup post(t)$. This is denoted by $M|t\rangle M'$. The marking M' is also denoted by $M|t\rangle$. This notation is extended to sequences of enabled transitions $M|t_1 \dots t_n\rangle M_n = M|t_1\rangle M_1 \dots M_{n-1}|t_n\rangle M_n$ and $M|t_1 \dots t_n\rangle$, respectively. A marking M is *reachable* if there exists a sequence of enabled transitions $(t_k)_{k=\{1,\dots,n\}}$ such that $In|t_1 \dots t_n\rangle M$. This sequence can be empty. The set of all reachable markings of a net N is denoted as $\mathcal{R}(N)$. A Petri net N is called *safe*, if for all reachable markings $M(p) \leq 1$ holds for all $p \in \mathcal{P}$. Then, M is a subset of P. From now on, all Petri nets are safe and no more multi-sets will be used. The *size* of a Petri net N is $|\mathcal{P} \cup \mathcal{T}|$.

A *node* x is a place or a transition $x \in \mathcal{P} \cup \mathcal{T}$. The binary *flow relation* $\mathcal{F}$ on nodes is defined as follows: $x \mathcal{F} y$ if $x \in pre(y)$. A node $x \in \mathcal{P} \cup \mathcal{T}$ is a *causal predecessor* of y, denoted as $x \leq y$, if $x \mathcal{F}^+ y$ ($\mathcal{F}^+$ is the transitive closure of $\mathcal{F}$). We write $x < y$ if $x \leq y$ and $x \neq y$ holds. Furthermore, $x \leq x$ holds for all $x \in \mathcal{P} \cup \mathcal{T}$. Two nodes $x, y \in \mathcal{P} \cup \mathcal{T}$ are *causally related*, if $x \leq y$ or $y \leq x$ holds. We say x is a *causal successor* of y, if $y \leq x$ holds. The definition of the causal successor relation is only sensible in the context of occurrence nets which are introduced shortly after.

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

A Petri net N is *finitely preceded*, if for every node $x \in \mathcal{P} \cup \mathcal{T}$ the set $\{y \in \mathcal{P} \cup \mathcal{T} \mid y \leq x\}$ is finite. A Petri net N is *acyclic*, if the directed graph $(\mathcal{P} \cup \mathcal{T}, \mathcal{F})$ is acyclic which is equivalent to the causal relation being a partial order [ERV02]. The following definitions lead to the definition of a branching process.

Definition 2.2 (Occurrence net). An *occurrence net* is a Petri net N with the following properties:

- N is acyclic,
- finitely preceded,
- $\forall p \in \mathcal{P} : |pre(p)| \leq 1$,
- no node is in self-conflict,
- and $In = \{p \in \mathcal{P} \mid \ pre(p) = \emptyset\}$.

$\triangleleft$

A homomorphism from one Petri net to another maps each node to a node such that the preset and postset relations are preserved. Note that the causal successor relation $\leq$ is a partial order in an occurrence net (It is reflexive ($x \leq x$ in $\mathcal{F}^+$), antisymmetric (because $(\mathcal{P} \cup \mathcal{T}, \mathcal{F})$ is acyclic), and transitive (because the transitive closure of the flow relation is transitive)).

Definition 2.3 (Homomorphism on nets). For two nets N_1 and N_2, a *homomorphism* from N_1 to N_2 is a mapping $h : \mathcal{P}_1 \cup \mathcal{T}_1 \to \mathcal{P}_2 \cup \mathcal{T}_2$ with following properties:

- $h(\mathcal{P}_1) \subseteq \mathcal{P}_2$ and $h(\mathcal{T}_1) \subseteq \mathcal{T}_2$,
- h preservers the structure of the transitions, i.e. $\forall t \in \mathcal{T}_1 : h(pre_1(t)) = pre_2(h(t)) \wedge h(post_1(t)) = post_2(h(t))$

$\triangleleft$

An *isomorphism* is a bijective homomorphism. A homomorphism on nets is called *initial* homomorphism if $h|_{In_1}$ is a bijection between In_1 and In_2.

Definition 2.4 (Branching process). A *branching process* of a net N_0 is a pair $B = (N, h)$, where N is an occurrence net and h a homomorphism from N to N_0 such that:

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

$\triangleleft$

The property ($*$) ensures that there are no duplicates of a transition having the same preset and the same label. A branching process is called *initial* branching process, if h is an initial homomorphism.

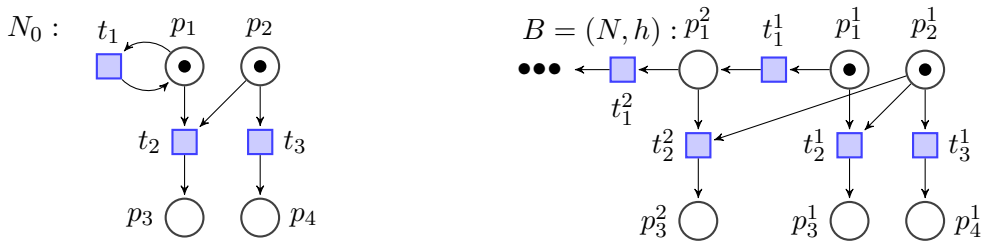


Figure 2.1: A Petri net N_0 on the left and a branching process $B = (N, h)$ of N_0 on the right.

Example 2.5 (Branching process). An example of a Petri net and a branching process is shown in Fig. 2.1. Places are shown as circles, transitions as boxes and the preset and postset relations as arrows. The initial marking $\{p_1, p_2\}$ is represented by the black dots, the *tokens*. The homomorphism h from N to N_0 is given as $h(p_j^i) = p_j$ and $h(t_j^i) = t_j$ for all places p_j^i and all transitions t_j^i. The transitions t_1^1 and t_2^1 are in conflict, i.e., $t_1^1 \# t_2^1$, and also $t_2^2 \# t_2^1$, $t_3^1 \# t_2^1$, $t_2^2 \# t_3^1$. Note that N exhibits infinitely many nondeterministic choices among the instances $t_2^1, t_2^2, \ldots$ of transition t_2 from place p_2^1. ◁

The places of a branching process are also referred to as *conditions* and the transitions of a branching process are also referred to as *events*. For any event e, $h(e)$ is referred to as the *label* of e. A firing sequence in a branching process is also called an *event sequence*. By convention, a branching process B has the components (N, h) and a branching process B_1 has the components (N_1, h_1), and analogously for other decorated names like B_2, B_3.

Definition 2.6 (Homomorphism on branching processes). For two branching processes B_1 and B_2 of a Petri net N_0, a *homomorphism* from B_1 to B_2 is a homomorphism h from N_1 to N_2 such that $h_2 \circ h = h_1$. ◁

The branching processes B_1 and B_2 are *isomorphic* if there exists an isomorphism from B_1 to B_2 which is denoted as $B_1 \cong B_2$. Note that the inverse function of a branching process isomorphism is also a branching process isomorphism.

A natural partial order on branching processes is defined in the following. In [Eng91], it is shown that this is a partial order.

Definition 2.7 (Subprocess relation of branching processes). Let B_1 and B_2 be two branching processes of a Petri net N_0. Then, B_1 approximates B_2, denoted as $B_1 \sqsubseteq B_2$, if there exists an injective homomorphism from B_1 to B_2. ◁

By convention, the subprocess homomorphism from B_1 to B_2 is denoted as $h_{B_2}^{B_1}$.

Now we define the *unfolding* of a Petri net as the maximal initial branching process. A branching process is called the *unfolding*, denoted as $u(N_0) = (N_{u(N_0)}, h_{u(N_0)})$, of a Petri net N_0 if it is a maximal initial branching process with respect to the subprocess relation $\sqsubseteq$ of branching processes of N_0.

The unfolding is unique up to isomorphism (renaming of nodes). By convention, $u(N_0)$ is also denoted as u and its components as N_u, h_u and thus their components as $\mathcal{P}_u, \mathcal{T}_u$, and so on. In [Eng91], it is shown that the following definition is an equivalent definition of the unfolding.

Definition 2.8 (Unfolding). A branching process B of a net N_0 is called the *unfolding* if for all sets $K \subseteq \mathcal{P}$ of pairwise concurrent conditions and all transitions $t_0 \in \mathcal{T}_0$ it holds that $h(K) = pre_0(t_0) \implies \exists t \in \mathcal{T} : pre(t) = K \wedge h(t) = t_0$. ◁

Informally speaking, there is an event in the unfolding every time such an event could be fired. By convention, any branching process B, B_1, B_2, ... is a branching process of the net N_0 if not stated otherwise and the unfolding of N_0 is denoted as u.

For every firing sequence $s = t_1 t_2 \ldots$ in a net N_0, there exists an event sequence $e_1 e_2 \ldots$ in u such that $s = h(e_1)h(e_2)\ldots$ [Vog91]. This event sequence of a firing sequence is denoted as $events(s)$. We write $e \in events(s)$ if $\exists e_i : e = e_i$.

In Fig. 2.1, the branching process B is the unfolding of the Petri net assuming that the dots to the left of the place p_1^2 indicate that the branching process continues infinitely in the same way.

We need the notions of configurations and cuts to work with branching processes.

Definition 2.9 (Configuration). A *configuration* C is a set of events that satisfies the following properties:

- $e \in C \Rightarrow \forall e' \leq e : e' \in C$ (C is downwards causally closed).
- $\forall e, e' \in C : \neg(e \# e')$ (C is conflict-free).

◁

A set of conditions is a *co-set* if its elements are pairwise concurrent. A co-set which is maximal w.r.t. set inclusion is a *cut* [Eng91]. By convention, co-sets and cuts are denoted as K, K_1, K_2, ... For a finite configuration C in a branching process B, the set $Cut(C) = (In \cup post(C)) \setminus pre(C)$ is a cut [Eng91]. Every reachable marking in a branching process is a cut and every cut is a reachable marking [BF88], i.e. the notions of cut and reachable marking coincide for a branching process. For a branching process B of a net N_0, the set $h(Cut(C))$ is a reachable marking in N_0. In particular, $Cut(C)$ is a reachable marking in a branching process [BF88]. Every reachable marking in a branching process defines a configuration (and vice versa) [ERV02].

For every cut K in a branching process B, the configuration $Cfg(K)$ is the configuration in B such that $Cut(Cfg(K)) = K$. By convention, Cfg is annotated by the index of the branching process the configuration belongs to, e.g. Cfg_1 yields a configuration of a branching process B_1 and analogously for other decorated names like B_2, B_3. Considering the unfolding u of N_0, it is notated as Cfg_u. By convention, the cut $Cut(C)$ is the cut of the branching process the configuration C belongs to, i.e. C is a subset of the events of that branching process. If this is ambiguous the branching process gets annotated the same way as for configurations.

The next definition is essential.

Definition 2.10 (Local configuration and last known cut). For a node x of a branching process B, the *local configuration* of x is the set $[x] = \{e \in \mathcal{T} \mid e \leq x\}$. The *last known cut* of x is the cut $lkc(x) = Cut([x])$. The *last known marking* of x is defined as $lkm(x) = h(lkc(x))$. $\lhd$

Note that the local configuration $[x] = \{e \in \mathcal{T} \mid e \leq x\}$ of a node x is indeed a configuration. It is downwards causally closed since the causal successor relation is transitive and it is conflict-free since no event is in self-conflict in an occurrence net.

In Fig. 2.3, an example of a last known cut and its last known marking is shown in the dotted boxes. The last known markings $lkm_{u(G)}(S_1^1) = \{S_1, A\}$ and $lkm_{u(G)}(S_1^2) = \{S_1, B\}$ differ.

Since the local configuration depends on the branching process of the node, the subscript of a branching process like B_1, B_2, etc. gets annotated as $[x]_1$, $[x]_2$, etc. by convention to indicate the branching process to which the local configuration belongs. A local configuration of an unfolding u is annotated as $[e]_u$. The last known cut and last known marking get annotated analogously.

Definition 2.11 (Future of a cut). The *future* of a cut $K \subseteq \mathcal{P}$ is the branching process $Fut(B, K)$ such that $\mathcal{P}_{Fut(B,K)} \cup \mathcal{T}_{Fut(B,K)} = \{x \in \mathcal{P}_B \cup \mathcal{T}_B \mid \forall p \in K : p \leq x \vee p || x\}$, the mappings $pre_{Fut(B,K)}$ and $post_{Fut(B,K)}$ are the mappings pre and $post$ restricted to $\mathcal{T}_{Fut(B,K)}$, and $In_{Fut(B,K)} = K$. $\lhd$

Generally, the *future* $Fut(B, K)$ is not an *initial* branching process of the underlying net N_0 of B but it is a branching process of the net N_0. By convention, we denote $Fut(B, Cut(C))$ by $Fut(B, C)$ for any configuration $C \subseteq \mathcal{T}$.

Informally speaking, the definition of the future of a cut coincides with the intuition that it is the branching process that follows after that cut. For two cuts K_1, K_2 the future $Fut(u, K_1)$ and the future $Fut(u, K_2)$ are isomorphic if $h_u(K_1) = h_u(K_2)$ [ERV02]. By convention, the isomorphism from $Fut(u, K_1)$ to $Fut(u, K_2)$ is denoted as $I^{Fut(u,K_1)}_{Fut(u,K_2)}$. By convention, if K_1 and K_2 are last known cuts of events e_1 and e_2, respectively, the isomorphism $I^{Fut(u,K_1)}_{Fut(u,K_2)}$ is denoted as $I^{[e_1]}_{[e_2]}$.

A definition of a partial order on the finite configuration follows. This is useful for inductive constructions. Beforehand, a *finite extension* of a configuration C is a set $E \subseteq \mathcal{T}$ such that $E \cap C = \emptyset$ and $C \cup E$ is a configuration. This is denoted as $C \oplus E$.

Definition 2.12 (Adequate order). A partial order $\preceq$ on the finite configurations of the unfolding u is an *adequate order* if

- $\preceq$ is well-founded (every non-empty subset of configurations has a minimal element),
- for all finite configurations $C_1, C_2 \subseteq \mathcal{T}_u$ it holds that $C_1 \subseteq C_2$ implies $C_1 \preceq C_2$, and
- for all finite configurations $C_1, C_2 \subseteq \mathcal{T}_u$ it holds that $\preceq$ is preserved by finite extensions, i.e. if $C_1 \preceq C_2$ and $h_u(Cut(C_1)) = h_u(Cut(C_2))$ then for all finite extensions E of C_1 it holds that $C_1 \oplus E \preceq C_2 \oplus I^{Fut(u,Cut(C_1))}_{Fut(u,Cut(C_2))}(E)$.

$\triangleleft$

For a total adequate order $\preceq$ on the configurations of a branching process, the configuration C_n, $n \in \mathbb{N}$, is the configuration such that $|\{C \mid C \preceq C_n\}| = n$.

By convention, $C_1 \prec C_2$ notates that $C_1 \preceq C_2$ and $C_1 \neq C_2$.

A total adequate order on the finite configurations of a safe Petri net is introduced in [ERV02]. Therefore, throughout this thesis is assumed that a total adequate order on the set of finite configurations exists and all examples use that total adequate order implicitly. This total adequate order is presented in the following.

Before the total adequate order on the configurations of a branching process B of a net N_0 can be defined itself, we need to define two auxiliary orders on the set of finite configurations. For the first one, let $\ll$ be an arbitrary total order on the set of transitions $\mathcal{T}_0$. For a configuration $C \subseteq \mathcal{T}$ let $\phi(C)$ be the sequence of transitions ordered according to $\ll$ which contains each transition t as often as there are events in C with label t. For example, if $t_1 \ll t_2 \ll t_3$ and $C = \{e_1^1, e_2^1, e_2^2, e_3^1\}$ $(h(e_j^i) = t_j)$ we get $\phi(C) = t_1t_2t_2t_3$. For two configuration C_1 and C_2, we write $\phi(C_1) \ll \phi(C_2)$ if $\phi(C_1)$ is lexicographically smaller than $\phi(C_2)$ w.r.t. $\ll$. For example, $\phi(\{e_1^1, e_2^1\}) = t_1t_2 \ll t_1t_3 = \phi(\{e_1^1, e_3^1\})$.

The second order is based on the Foata normal form of a configuration. The Foata normal form of a configuration C is a sequence of sets of events $FNF(C) = F_1F_2 \dots F_k$ such that for all $i \in \{1, \dots, k\}$ for all $e \in F_i$ holds that e is minimal w.r.t. $\leq$ in $F_i \cup F_{i+1} \cup \dots \cup F_k$. Informally speaking, the set of events $F_1, \dots, F_k$ represent a maximal concurrent execution. These sets of event can be obtained by slicing out the set of events which are minimal w.r.t. $\leq$ in each step starting with F_1. For example in Fig. 2.1, $FNF(\{t_1^1, t_1^2, t_3^1\}) = \{t_1^1, t_3^1\}\{t_1^2\}$. The Foata normal forms of two configurations C_1 and C_2 are compared with respect to $\ll$ as follows: $FNF(C_1) = F_1^1 \dots F_k^1 \ll F_1^2 \dots F_j^2 = FNF(C_2)$ if there exists $1 \leq i \leq k$ such that

1. for all $1 \leq l < i$ holds that $\phi(F_l^1) = \phi(F_l^2)$, and
2. $\phi(F_i^1) \ll \phi(F_i^2)$.

The total adequate order for safe nets is defined as follows.

Definition 2.13 (Total adequate order [ERV02]). Let C_1 and C_1 be finite configurations in the unfolding u of a net N_0. The *total adequate order* is defined such that $C_1 \preceq C_2$ holds if

1. $|C_1| < |C_2|$, or
2. $|C_1| = |C_2|$ and $\phi(C_1) \ll \phi(C_2)$, or
3. $\phi(C_1) = \phi(C_2)$ and $FNF(C_1) \ll FNF(C_2)$.

$\triangleleft$

The total adequate order compares the number of events of the configurations. If they have the same number of events, $\phi(C_1)$ and $\phi(C_2)$ are compared lexicographically. If this is also equal, the Foata normal forms of the configurations are compared.

Comparing the Foata normal forms is an edge case that does not occur in any example throughout this work. The arbitrary total order $\ll$ on the transitions of a net is always chosen to be the alphabetical order with indented subscript and superscript. The subscript is indented first followed by the superscript. However, the order of indenting the superscript and subscript is not relevant in any example since every underlying net in an example where the adequate order is considered there is no transition in any underlying net which has both superscript and subscript, i.e. there exists no label with a superscript.

Example 2.14 (Total adequate order). For the Petri net N_0 and its initial part of the unfolding B in Fig. 2.1, the total adequate order on the finite configurations is $\emptyset \preceq \{t_1^1\} \preceq \{t_2^1\} \preceq \{t_3^1\} \preceq \{t_1^1, t_1^2\} \preceq \{t_1^1, t_2^2\} \preceq \{t_1^1, t_3^1\} \preceq \ldots$ $\lhd$

A total adequate order on the set of finite configurations of a branching process induces a bijective mapping between the natural numbers and the set of finite configurations. There exists a total adequate order for every branching process of a safe net. This enables proofs by induction over the set of finite configurations of a branching process. The configuration C_n in the context of an inductive proof is the configuration such that there are n equal or lesser configurations, i.e. $C_1 = \emptyset$ is the base case of the induction, and the running variable of the induction is n. For the induction assumption, we assume that the property which we want to show holds for all configurations C that are equal or lesser than C_n w.r.t. the total adequate order. For the induction step, we show that the property holds for the next (after C_n) configuration regarding the total adequate order which is C_{n+1}. Such a proof is called a proof *by induction over the set of finite configuration of a branching process*. The branching process may vary and is explicitly named at the start of a proof.

2.1.1 Advanced Branching Process Properties

In this subsection, basic observations regarding branching processes are presented. The following observations are helpful to show the results in Chapter 3. The presented lemmas cover general branching process properties.

The first lemma states that the union of two local configurations of concurrent events is a configuration.

Lemma 2.15 (Configuration of concurrent events). *For two concurrent events $e_1, e_2 \in \mathcal{T}$ of a branching process B, the union of the local configurations $[e_1] \cup [e_2]$ is a configuration.*

Proof. It holds that $[e_1] \cup [e_2]$ is downward causally closed because $[e_1]$ and $[e_2]$ are downwards causally closed, respectively. The local configurations $[e_1]$ and $[e_2]$ are conflict-free and $e_1 || e_2$, i.e. e_1 and e_2 are not in conflict. If there are events $e \in [e_1]$ and $e' \in [e_2]$ which are in conflict it follows that e_1 and e_2 are in conflict since $e \leq e_1$ and $e' \leq e_2$ hold. This is a contradiction to $e_1 || e_2$. Thus, the set $[e_1] \cup [e_2]$ is conflict-free. $\square$

The next lemma states that a branching process isomorphism preserves the causal successor relation.

Lemma 2.16 (Isomorphism preserves successor relation). *Let h be a branching process isomorphism from a branching process B_1 to a branching process B_2. For all $e, e' \in \mathcal{T}_1$, it holds that $h(e) \leq h(e')$ if and only if $e \leq e'$.*

Proof. If $e \leq e'$ holds there exists a sequence of nodes $eb_1e_1 \dots e_{n-1}b_ne'$ such that $e\mathcal{F}_1b_1$, $b_n\mathcal{F}_1e'$ and $b_i\mathcal{F}_1e_i$ and $e_i\mathcal{F}_1b_{i+1}$ for all $1 \leq i < n$ by definition of the causal successor relation. Since h is an isomorphism there exists an isomorphism h^{-1} from B_2 to B_1 such that $h^{-1} \circ h = id = h \circ h^{-1}$. Applying h to the sequence $eb_1e_1 \dots e_{n-1}b_ne'$ yields a sequence of nodes $h(e)h(b_1) \dots h(b_n)h(e')$ in B_2. Since h is a homomorphism the structure of the transitions is preserved, i.e. $h(pre(e)) = pre(h(e))$ and $h(post(e)) = post(h(e))$ for all events $e \in \mathcal{T}_1$. Thus, $b\mathcal{F}_1e$ implies $h(b)\mathcal{F}_2h(e)$ and $e\mathcal{F}_1b$ implies $h(e)\mathcal{F}_2h(b)$ for all conditions $b \in \mathcal{P}_1$ and events $e \in \mathcal{T}_1$. Therefore, $h(e) \leq h(e')$ holds. Applying h^{-1} on such a sequence of nodes in B_2 analogously yields the implication the other way around. □

A branching process isomorphism induces an isomorphism on the set of finite configurations.

Lemma 2.17 (Isomorphism on configurations). *A branching process isomorphism h from a branching process B_1 to a branching process B_2 induces an isomorphic mapping of the configurations of B_1 to the configurations of B_2: for all configurations $C \subseteq \mathcal{T}_1$ the set $h(C) \subseteq \mathcal{T}_2$ is a configuration of B_2.*

Proof. Let $C \subseteq \mathcal{T}_1$ be a configuration in B_1. By Lemma 2.16, the isomorphism h preserves the causal successor relation $\leq$. It directly follows that $h(C)$ is downwards causally closed and conflict-free in B_2. Since h is an isomorphism, mapping each configuration C of B_1 to $h(C)$ yields an isomorphic mapping on the set of finite configurations from B_1 to the set of finite configurations of B_2. □

A branching process isomorphism also preserves the concurrency relation between nodes.

Lemma 2.18 (Isomorphism preserves concurrency). *Let h be a branching process isomorphism from a branching process B_1 to a branching process B_2. For all $e, e' \in \mathcal{T}_1$, $h(e)||h(e')$ holds if and only if $e||e'$ holds.*

Proof. By Lemma 2.16, the isomorphism h preserves the causal successor relation, i.e. $h(e)$ and $h(e')$ are not causally related.

It remains to show that $h(e)$ and $h(e')$ are not in conflict. By definition, the nodes $h(e)$ and $h(e')$ are in conflict if there exist two transitions $t_1, t_2 \in \mathcal{T}$, $t_1 \neq t_2$ with $pre(t_1) \cap pre(t_2) \neq \emptyset$, $t_1 \leq h(e)$ and $t_2 \leq h(e')$. The inverse of h is also an isomorphism. Therefore, applying Lemma 2.16 on $t_1 \leq h(e)$ and $t_2 \leq h(e')$ yields that if $h(e)$ and $h(e')$ are in conflict e and e' are also in conflict. This is a contradiction to $e||e'$. Overall, it follows that $h(e)||h(e')$. □

Lemma 2.16 and Lemma 2.18 are applicable to subprocess homomorphisms since a subprocess homomorphism is injective by definition and restricting the image set to the set of nodes that have a preimage yields an isomorphism.

The next lemma states that the future of a last known cut of a node contains all nodes which are concurrent to that node.

Lemma 2.19 (Future of a last known cut). *Let $Fut(B, [e])$ be the future of a local configuration of a branching process B. Then, it holds that for all $e' \in \mathcal{T} : e||e' \implies e' \in Fut(B, [e])$.*

Proof. Let $e, e' \in \mathcal{T}$ such that $e||e'$. We show that for all $b \in Cut([e])$ it holds that $b||e' \vee b \leq e'$ (which is equivalent to not being in a conflict to b).

We show the property by contradiction. Assume $e'\#b$ for some $b \in Cut([e])$. Then, since $|pre(b)| \leq 1$ (property of an occurence net) either $pre(b) = \{e_b\}$ for some $e_b \in \mathcal{T}$ or $pre(b) = \emptyset$ holds. It cannot be the case that $pre(b) = \emptyset$ because the definition of two nodes being in conflict requires a causal predecessor for both nodes. So, if $pre(b) = \{e_b\}$ it follows by the definition of nodes being in conflict that $e_b\#e'$. Because $b \in Cut([e])$ it holds that $e_b \in [e]$, thus $e_b \leq e$. Since $e_b \leq e$ it follows that $e\#e'$. Contradiction. □

The next lemma states that a local configuration of an event in the future of a last known cut can be written as a union of a set of events in that future and a set of events where the events are causal predecessor of a condition in that last known cut, i.e. a set of events which are not in that future.

Lemma 2.20 (Split configuration). *Let B be a branching process, $f \in \mathcal{T}$ and $e \in \mathcal{T}_{Fut(B,lkc(f))}$ an event in the future of $lkc(f)$. Then, $[e]$ is the following set union*

$$[e] = \bigcup_{\{b \in lkc(f) | b \leq e\}} [b] \cup \{e' \in \mathcal{T}_{Fut(B,lkc(f))} \mid e' \leq e\}.$$

Proof. Show set equivalence: For all $e_0 \in [e]$ either $e_0 \in \mathcal{T}_{Fut(B,lkc(f))}$ or $e_0 \notin \mathcal{T}_{Fut(B,lkc(f))}$ holds. If $e_0 \in \mathcal{T}_{Fut(B,lkc(f))}$ it holds that $e_0 \in \{e' \in \mathcal{T}_{Fut(B,lkc(f))} \mid e' \leq e\}$.

If $e_0 \notin \mathcal{T}_{Fut(B,lkc(f))}$: In the branching process B, there exists a sequence of causal successors $e_0 b_0 e_1 \dots b_{n-1} e_{n-1} b_n e_{n+1}$, $e_{n+1} = e$. Let b_i be a condition such that $e_i \notin \mathcal{T}_{Fut(B,lkc(f))}$ and $e_{i+1} \in \mathcal{T}_{Fut(B,lkc(f))}$. Such a condition exists since $e_0 \notin \mathcal{T}_{Fut(B,lkc(f))}$ and $e \in \mathcal{T}_{Fut(B,lkc(f))}$, and $lkc(f)$ is a maximal set of pairwise concurrent conditions. This b_i is a condition in $lkc(f)$ since b_i is in $\mathcal{P}_{Fut(B,lkc(f))}$ ($b_i \in pre(e_{i+1})$ and $e_{i+1} \in \mathcal{T}_{Fut(B,lkc(f))}$). It holds that $pre_{Fut(B,lkc(f))}(b) = \emptyset$ since $e_i \notin \mathcal{T}_{Fut(B,lkc(f))}$ (and the occurrence net property holds that $|pre(b)| \leq 1$). Thus, $e_0 \in [b_i] \subseteq \bigcup_{\{b \in lkc(f) | b \leq e\}} [b]$. □

The following lemmas are used in Chapter 5 to determine last known markings.

Lemma 2.21 (Conditions of a cut are maximal conditions). *Let B be a branching process and $C \subseteq \mathcal{T}$ a configuration in B. For the cut $Cut(C)$ it holds that $Cut(C) = \{b \in In \cup post(C) \cup pre(C) \mid b \text{ is maximal w.r.t. } \leq|_{In \cup post(C) \cup pre(C)}\}$.*

Proof. First, we show the subset relation that $Cut(C) \subseteq \{b \in In \cup post(C) \cup pre(C) \mid b \text{ is maximal w.r.t. } \leq|_{In \cup post(C) \cup pre(C)}\}$.

Let $b \in Cut(C)$. It holds that $In \cup post(C) \cup pre(C) = (((In \cup post(C)) \setminus pre(C)) \cup pre(C)$ (by standard set operations). We now show for the two sets $pre(C)$ and $((In \cup post(C)) \setminus pre(C)$ that b is maximal w.r.t. $\leq$ in each of those sets, which implies that b is maximal w.r.t. $\leq$ restricted to the union of those two sets.

Because $Cut(C) = (In \cup post(C)) \setminus pre(C)$ and $b \in Cut(C)$, there exists no $e \in C$ such that $b \in pre(e)$. This means, that b is maximal w.r.t. $\leq$ restricted to $C \cup \{b\}$. Since all conditions in $pre(C)$ are causal predecessors of an event in C and $\leq$ is a partial order (transitive), b is also maximal w.r.t $\leq$ restricted to $pre(C) \cup \{b\}$.

Since $Cut(C)$ is a co-set, b is not causally related to any $b' \neq b$, $b' \in Cut(C)$. Thus, b is also maximal w.r.t. $\leq$ restricted to $(In \cup post(C)) \setminus pre(C)$. Overall, b is maximal w.r.t. $\leq|_{In \cup post(C) \cup pre(C)}$

Second, we show the subset relation that $Cut(C) \supseteq \{b \in In \cup post(C) \cup pre(C) \mid b \text{ is maximal w.r.t. } \leq|_{In \cup post(C) \cup pre(C)}\}$.

Let b be maximal w.r.t. $\leq|_{In \cup post(C) \cup pre(C)}$. If there exists an $e \in C$ such that $b \leq e$, $b \leq post(e)$ follows which contradicts the assumption that b is maximal. Therefore, there exists no $e \in C$ such that $b \leq e$. Thus, it holds that $b \in In \cup post(C)$ and $b \notin pre(C)$. □

The second lemma here states that the last known cut of an event always contains the postset of that event.

Lemma 2.22 (Last known cut contains postset). *For a branching process B, the cut of a local configuration $Cut([b])$ of a condition b contains the conditions in $post(pre(b))$.*

Proof. If $b \in In$, trivially $pre(b) = \emptyset$ and the proposition holds.

If $b \notin In$, $|pre(b)| = 1$. Let e be the only event in $pre(b)$. Since $[b] = \{e \in \mathcal{T} \mid e \leq b\}$, e is the maximal event w.r.t. $\leq|_{[b]}$. Thus by Lemma 2.21, $post(pre(b)) = post(e) \subseteq Cut([b])$. □

The following lemma states that the last known cut of an event can be rewritten using the last known cuts of the conditions in the preset of that event.

Lemma 2.23 (Conditions of a last known cut). *Let B be a branching process and $e \in \mathcal{T}$. For the last known cut $lkc(e)$ it holds that $lkc(e) = (In \cup \bigcup_{b \in pre(e))} lkc(b) \cup post(e)) \setminus pre([e])$.*

Proof. Since $[e] = \{e' \in \mathcal{T} \mid e \leq e'\} = \bigcup_{b \in pre(e)} [b] \cup \{e\}$, we have $Cut(C) = (In \cup post(C)) \setminus pre(C) = (In \cup post(\bigcup_{b \in pre(e)} [b]) \cup post(e)) \setminus pre([e])$.

We show that $post(\bigcup_{b \in pre(e)} [b]) \setminus pre([e]) = (\bigcup_{b \in pre(e)} lkc(b)) \setminus pre([e])$. For this, we show both set inclusions.

Let $b' \in post(\bigcup_{b \in pre(e)} [b]) \setminus pre([e])$. Then, for some $b \in pre(e)$ it holds that $b' \in post([b])$. Since $[b] \subseteq [e]$ it holds that $pre([b]) \subseteq pre([e])$ and $b' \in lkc(b) = (In \cup post([b])) \setminus pre([b]) \supseteq (In \cup post([b])) \setminus pre([e])$ follows. Since $b' \in post(\bigcup_{b \in pre(e)} [b]) \setminus pre([e])$ and $b' \in lkc(b)$ for some $b \in pre(e)$ it holds that $b' \in (\bigcup_{b \in pre(e)} lkc(b)) \setminus pre([e])$.

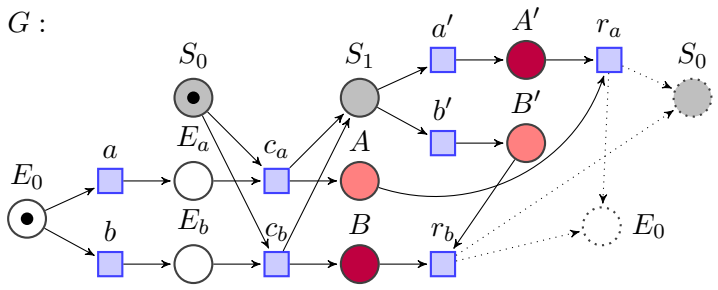


Figure 2.2: The Petri game "Same Choice".

For the other set inclusion, let $b' \in (\bigcup_{b \in pre(e)} lkc(b)) \setminus pre([e])$. Then, $b' \in lkc(b)$ for some $b \in pre(e)$ and $b' \notin pre([e])$. It follows that $b' \in post(\bigcup_{b \in pre(e)}[b])$. Since $b' \notin pre([e])$ it holds that $b' \in post(\bigcup_{b \in pre(e)}[b]) \setminus pre([e])$. □

2.2 Petri Games

We continue with the definition of a Petri game. We only consider Petri games on finite and safe Petri nets. The Petri net on which a Petri game is played is called the *underlying* Petri net of the Petri game.

Definition 2.24 (Petri game). A *Petri game* or game G on an underlying finite and safe Petri net N is a tuple $G = (\mathcal{P}^S, \mathcal{P}^E, \mathcal{T}, pre, post, In, \mathcal{B})$, where the set of places $\mathcal{P}$ of N is partitioned into disjoint sets of *system places*, $\mathcal{P}^S$, and *environment places*, $\mathcal{P}^E$, and where $\mathcal{B} \subseteq \mathcal{R}(N)$ is the set of *bad markings*. ◁

Example 2.25 (Petri game). Figure 2.2 shows an example of a Petri game in graphical representation. System places are shown in grey, environment places are kept white. The environment player chooses between the transitions a and b. The system player, after communicating with the environment player, must perform a corresponding choice between a' and b' to avoid the bad markings $\{A, B'\}$ (in red) and $\{B, A'\}$ (in purple). After the reset transitions r_a and r_b the players go back to the initial marking $\{S_0, E_0\}$. (The dotted places named E_0 and S_0 are identified with the initial places E_0 and S_0, to keep it visually simple.) ◁

By convention, a Petri game G has the components $(\mathcal{P}^S, \mathcal{P}^E, \mathcal{T}, pre, post, In, \mathcal{B})$ and the underlying net is N. A game G_0 has the components $(\mathcal{P}_0^S, \mathcal{P}_0^E, \mathcal{T}_0, pre_0, post_0, In_0, \mathcal{B}_0)$ and the underlying net is N_0, and analogously for games with decorated names like G_1, G_2, G_3.

The *unfolding* $u(G)$ of a Petri game G is like the unfolding $u(N)$ of the underlying Petri net of G. For a game G_0, the set of environment places and system places are extended to branching processes of its underlying net: we say a condition b of a branching process B of N_0 is a system condition if $h(b) \in \mathcal{P}_0^S$ and it is an environment condition

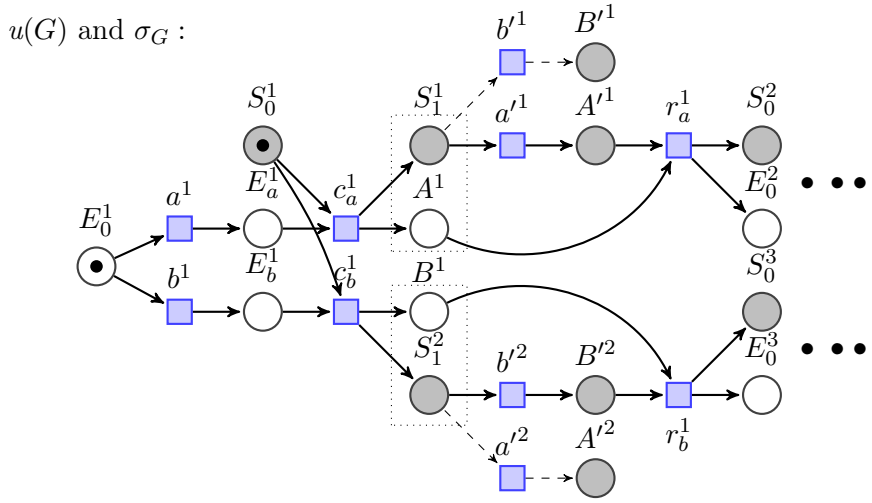


Figure 2.3: Initial part of the unfolding of the net of the Petri game "Same Choice".

if $h(b) \in \mathcal{P}_0^E$. The set of places $\mathcal{P}$ of the branching process B is separated into a set of system conditions $\mathcal{P}^S = \{b \mid h(b) \in \mathcal{P}_0^S\}$ and a set of environment conditions $\mathcal{P}^E = \{b \mid h(b) \in \mathcal{P}_0^E\}$.

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

Definition 2.26 (Winning strategy). A *winning strategy* σ of a Petri-game G_0 is a branching process $\sigma = (N, h)$ of N_0 satisfying the following properties:

1. **Justified refusal**: Let $C \subseteq \mathcal{P}$ be a set of pairwise concurrent places and $t \in \mathcal{T}_0$ a transition with $h(C) = pre_0(t)$. Then there is a transition $t' \in \mathcal{T}$ with $h(t') = t$ and $pre(t') = C$ or there exists a place $p \in C$ with $h(p) \in \mathcal{P}_0^S$, such that $t \notin h(post(p))$.
2. **Safety**: For all reachable markings $M \in \mathcal{R}(N)$ it holds that $h(M) \notin \mathcal{B}$.
3. **Determinism**: For all $p \in \mathcal{P}$ with $h(p) \in \mathcal{P}_0^S$ and for all reachable markings $M \in \mathcal{R}(N)$ with $p \in M$ there exists at most one transition $t \in post(p)$, which is enabled in M.
4. **Deadlock avoiding**: For all reachable markings $M \in \mathcal{R}(N)$ there exists an enabled transition in M, if a transition is enabled in $h(M)$ in the underlying net N_0.

$\triangleleft$

By convention, a winning strategy σ has the same components as the branching process B (without an index) and analogously for decorated names like σ_1, σ_2 (like B_1, B_2). We refer to a token on a system place as a system player and to a token on an environment place as an environment player.

The *justified refusal* property ensures that a system player allows all instances of an outgoing transition or no instance at all. It also states that an environment player does

not restrict its outgoing events. The *safety* property ensures that no bad markings are reachable. The *determinism* property ensures that for each system place at most one transition is enabled in every reachable marking. The *deadlock avoiding* property ensures that the system allows at least one transition in every reachable marking if an enabled transition exists in that marking.

Example 2.27. In Fig. 2.3, an initial part of the unfolding of Petri game G in Fig. 2.2 is shown and inside as a subprocess, an initial part of a winning strategy is obtained by erasing all nodes reached with dashed arrows. ◁

The next definition characterizes Petri nets that keep a constant number of tokens and each token moves only within its own partition.

Definition 2.28 (Concurrency preserving and token-partitioning nets). A Petri net N is *concurrency preserving* if and only if $\forall t \in \mathcal{T} : |pre(t)| = |post(t)|$ holds. Furthermore, N is called *partitioning preserving* if N is concurrency preserving and there exists a partition $P = \dot{\bigcup}_{i=1,\dots,n} P_i$ of $\mathcal{P}$ such that $\forall t \in \mathcal{T} : \forall i \in \{1,\dots,n\} : |pre(t) \cap P_i| = |post(t) \cap P_i|$. Such a partition P is called a *token partition* if and only if $|P_i \cap In| = 1$ for all $i = 1,\dots,n$. Then, N is *token-partitioning*. By convention, the number of tokens (or players) is denoted as ν. ◁

Note that for a token-partitioning Petri net every reachable marking contains exactly one place of each partition. For example, the net in Fig. 2.2 is token-partitioning with partition $P = P_0 \dot{\cup} P_1 = \{E_0, E_a, E_b, A, B\} \dot{\cup} \{S_0, S_1, A', B'\}$.

By convention, the partition of a Petri net N, N_1 or N_2 is denoted always as P without any index. If there are two or more partitions that need to be distinguished this is stated explicitly. Furthermore, the set of indices of a partition P is denoted as $I_P = \{1,\dots,\nu\}$. By convention, $\nu = |I_P|$ is the number of tokens. If P is token-partitioning the indices refer to the tokens. From now on, all Petri nets are token-partitioning if not stated otherwise. We use the indices i, j and k to refer to a token in $\{1,\dots,\nu\}$ if not stated otherwise. For a set of places M, if it holds that $|M \cap P_i| = 1$ we write $p_i(M)$ for the single place $p_i \in M \cap P_i$ as defined below:

Definition 2.29 (Token's projection). For a set $M \subseteq \mathcal{P}$, *token j's projection* is defined if and only if $|M \cap P_j| = 1$. Then, it is defined as $p_j(M) = s$, where $s \in M \cap P_j$. ◁

We extend a partition P (and so the token's projection) of a Petri net N_0 to the conditions $\mathcal{P}$ of a branching process B naturally as follows: $\forall b \in \mathcal{P} : b \in P_i \Leftrightarrow h(b) \in P_i$. A token i participates in a transition t if and only if $pre(t) \cap P_i \neq \emptyset$. The set of participating tokens of a transition is defined as follows.

Definition 2.30 (Partition index mapping). The mapping $Ind : \mathcal{T} \to 2^P$ is defined as $Ind(t) = \{i \in P \mid pre(t) \cap P_i \neq \emptyset\}$. ◁

Note that the mapping Ind is also applicable on events since we extended the partition to the conditions of any branching process.

We show that all conditions of one token i in $In \cup pre(C) \cup post(C)$ are causally related for any configuration C.

Lemma 2.31 (Causally related conditions). *Let N_0 be a token-partitioning net with partition P and B a branching process of N_0. For a finite configuration $C \subset \mathcal{T}$ and a fixed $i \in I_P$, the causal successor relation is a total order on the set $(In \cup pre(C) \cup post(C)) \cap P_i$.*

Proof. Since the branching process B is token-partitioning, every reachable marking in a branching process consists of exactly one condition from every P_i, $i \in I_P$. If there exist two different conditions $b, b' \in P_i$, $i \in I_P$, such that $b||b'$ there exists a reachable marking M and $b, b' \in M$ as an implication of Lemma 2.15. This is a contradiction to B being token-partitioning. Thus, all conditions $b, b' \in P_i$ are either causally related or in conflict. Therefore, also all events $e_1, e_2 \in \mathcal{T}$ with $i \in Ind(e_1)$, $i \in Ind(e_2)$, $i \in I_P$, are either causally related or in conflict. Since every configuration is conflict-free by definition, all events $e_1, e_2 \in C$ with $i \in Ind(e_1)$ and $i \in Ind(e_2)$ are causally related. It follows that all conditions of token i in $In \cup pre(C) \cup post(C)$ are causally related. □

2.2.1 Advanced Petri Game Properties

In the following, an equivalent definition of a winning strategy is presented. For that, a definition of a strategy is introduced and it is defined when it is a winning strategy.

Definition 2.32 (Strategy). A *strategy* of a Petri game G_0 is a mapping $\tau : \mathcal{P}_u \to 2^{\mathcal{T}}$ such that for all $b \in \mathcal{P}_u : \tau(b) \subseteq post(h_u(b))$. ◁

For a condition b, the set $\tau(b)$ contains the labels of the events that are allowed to be fired. If all preconditions of an event allow its label the event is part of the strategy's branching process as defined in the following.

Definition 2.33 (Strategy's branching process). The branching process B_τ of a strategy τ is the branching process such that $\forall e \in \mathcal{T}_u : e \in \mathcal{T}_\tau \Leftrightarrow \forall e' \in [e]_u : \forall b \in pre_u(e') : h_u(e') \in \tau(b)$. ◁

A strategy τ of a game G_0 is *winning* if its branching process B_τ is a winning strategy. In the following, it is shown that the winning properties of B_τ can be checked in each of its configurations by looking at the strategy τ.

Lemma 2.34 (Sufficient winning properties). *The branching process B_τ of a strategy τ of a game G_0 satisfies the winning properties if the following properties hold.*

1. ***Unrestricted environment**: $\forall b \in \mathcal{P}_u : h_u(b) \in \mathcal{P}^E \implies (\tau(b) = post(h_u(b)))$.*
2. ***Safety**: For all configurations $C \subseteq \mathcal{T}_\tau$, it holds that $h_u(Cut(C)) \notin \mathcal{B}$.*
3. ***Determinism**: For all configurations $C \subseteq \mathcal{T}_\tau$, it holds that $\forall b \in Cut(C) : h_u(b) \in \mathcal{P}_0^S \implies \exists^{\leq 1} e \in post(h_u(b)) : (pre_u(e') \subseteq Cut(C) \wedge \forall b' \in pre_u(e) : h_u(e) \in \tau(b'))$.*
4. ***Deadlock avoiding**: For all configurations $C \subseteq \mathcal{T}_\tau$, it holds that $\exists e \in \mathcal{T}_u : pre_u \subseteq Cut(C) \implies \exists e' \in \mathcal{T}_u : (pre_u(e') \subseteq Cut(C) \wedge \forall b \in pre_u(e') : h_u(e') \in \tau(b))$.*

Proof. The unrestricted environment property implies the justified refusal property of Def. 2.26: By the unrestricted environment property every environment condition must allow all labels of events that are in its postset. By the definition of B_τ, an event is in $\mathcal{T}_\tau$ if and only if $\forall e' \in [e]_u : \forall b \in pre_u(e') : h_u(e') \in \tau(b)$. Now, there are two cases to distinguish if an event is not in $\mathcal{T}_\tau$. If there is a $e' \in [e]_u$, $e' \neq e$ such that $e' \notin \mathcal{T}_\tau$ there exists no co-set in B_τ that enables e and the justified refusal property holds. If there exists no such e', by definition of B_τ there exists a condition in the preset of e that does not allow the label of e. Since all environment conditions allow the label of e, that condition is a system condition. Again, by definition of B_τ there is no event with the label of e in the postset of that system condition. Thus, the justified refusal property holds.

The safety property of Def. 2.26 follows directly by the definition of the safety property here.

The determinism property here implies the determinism property of Def. 2.26: Let C be a configuration in B_τ. By definition of B_τ all events in the configuration C are allowed by all their preconditions. The determinism property states that for every system condition in $Cut(C)$ there exists at most one event in its postset which is enabled in $Cut(C)$ and all conditions in the preset of that event allow the label of that event. By definition of B_τ those are exactly the events that are in $\mathcal{T}_\tau$. Thus, the determinism property of Def. 2.26 holds.

The deadlock avoiding property in Lemma 2.34 implies the deadlock avoiding property of Def. 2.26: Let C be a configuration in B_τ. By definition of B_τ all events in the configuration C are allowed by all their preconditions. The deadlock avoiding property here states that there must be an event that is allowed by all its preconditions if an event is enabled in $Cut(C)$. By definition of B_τ, such an event is in $\mathcal{T}_\tau$ if all preconditions allow its label. Therefore, the deadlock avoiding property of Def. 2.26 holds if the deadlock avoiding property here holds. □

In Chapter 3, the winning properties are checked in each configuration according to Lemma 2.34. A configuration $C \subseteq \mathcal{T}_\tau$ is winning if it satisfies the properties 1-4 of Lemma 2.34:

1. $\forall b \in Cut(C) : h_u(b) \in \mathcal{P}^E \implies \tau(b) = post(h_u(b))$,
2. $h_u(Cut(C)) \notin \mathcal{B}$,
3. $\forall b \in Cut(C) : h_u(b) \in \mathcal{P}_0^S \implies \exists^{\leq 1} e \in post(h_u(b)) : (pre(e) \subseteq Cut(C) \forall b' \in pre_u(e) : h_u(e) \in \tau(b'))$, and
4. $\exists e \in \mathcal{T}_u : pre_u \subseteq Cut(C) \implies \exists e' \in \mathcal{T}_u : (pre_u(e') \subseteq Cut(C) \wedge \forall b \in pre_u(e') : h_u(e') \in \tau(b))$.

Furthermore, to be able to use this definition of a winning strategy we proceed by showing the existence of a strategy as in Def. 2.26 is equivalent to the existence of a strategy satisfying the winning conditions as in Lemma 2.34.

Lemma 2.35 (Existence of winning strategies). *In a Petri game G_0 of an underlying net N_0, there exists a winning strategy as in Def. 2.26 if and only if there exists a strategy τ as in Def. 2.32 where B_τ satisfies the conditions in Lemma 2.34.*

Proof. If there exists a strategy τ as in Def. 2.32 where B_τ satisfies the conditions in Lemma 2.34, by Lemma 2.34 B_τ is a winning strategy as in Def. 2.26.

If there exists a winning strategy B as in Def. 2.26, a strategy τ is defined as follows assuming that the subprocess homomorphism from B to u is the identity function:

$$\tau(b) = \begin{cases} h(post(b)) & \text{if } b \in \mathcal{P}^S \\ post_0(h(b)) & \text{if } b \in \mathcal{P}^E \cup \mathcal{P}^E_u \\ \emptyset & \text{else} \end{cases}.$$

By definition of B_τ follows that every event in $\mathcal{T}$ is an event in $\mathcal{T}_\tau$. Since B satisfies the justified refusal property, for every event of the unfolding that is not in $\mathcal{T}$ there exists a system precondition that does not have any event with the same label in its postset or not all preconditions of that event are in $\mathcal{P}$. Therefore, every event of $\mathcal{T}_\tau$ is an event in $\mathcal{T}$. Thus, B_τ and B are isomorphic and the statement holds. □

2.3 Asynchronous Automata and Traces

A closely related formalism to Petri games are asynchronous games played on Zielonka automata [Zie87]. Instead of places and transitions, a Zielonka automata has states and actions, and instead of tokens, a fixed number of processes is used. Each process has its own set of states and an action may have multiple participating processes changing their states like a transition may have multiple participating tokens changing their places. In the model of asynchronous games, *traces* based on a dependency relation of the actions are used to express concurrency and causal dependencies opposed to Petri games, where branching processes are used to express concurrency and causal dependencies. We also refer to asynchronous games on Zielonka automata as control games. We introduce the notion of *traces* for Petri nets. The notions of traces and configurations coincide, i.e. each trace defines a configuration and vice versa. Informally speaking, a trace is a firing sequence which is extended by a causal successor relation between its transitions according to the causal successor relation of the events in its event sequence in the unfolding.

Definition 2.36 (Dependency relation). For a Petri net N the dependency relation $D \subseteq \mathcal{T} \times \mathcal{T}$ is defined as follows: $(t_1, t_2) \in D :\Leftrightarrow post(t_1) \cap pre(t_2) \neq \emptyset \ \vee \ pre(t_1) \cap post(t_2) \neq \emptyset$. The independency relation is defined as $\bar{D} = \mathcal{T} \times \mathcal{T} \setminus D$. ◁

For example in the net of Fig. 2.1, $(t_1, t_2) \in D$ and $(t_1, t_3) \in \bar{D}$ hold.

Definition 2.37 (Trace). The trace equivalence relation of a net N is the smallest equivalence relation on $\mathcal{T}^*$ which commutes independent transitions, i.e. for all $t_1, t_2 \in \mathcal{T}$ and $w_1, w_2 \in \mathcal{T}^*$ if $(t_1, t_2) \in \bar{D} \Rightarrow w_1 t_1 t_2 w_2 \equiv w_1 t_2 t_1 w_2$. The equivalence classes of the set of all firing sequences in a net are called *traces*. We say a trace π is a trace of the net N. An element of a trace is a *linearization* of that trace. ◁

A firing sequence is a linearization of a trace. For a firing sequence s, its trace is denoted as $\overline{s}$. Every trace π defines a configuration C and vice versa [Vog91]. The trace of a configuration C is denoted as $\pi(C) = \overline{h(e_1)h(e_2)\ldots}$, where for all $e_i, e_j \in C$ holds that $i < j$ if $e_i < e_j$. The configuration of a trace $\pi = t_1 t_2 \ldots$ is denoted as $\mathit{Cfg}(\pi) = \{e \in \mathit{events}(s) \mid s \in \pi\}$. Note that one linearization s of a trace π suffices in the definition of $\mathit{Cfg}(\pi)$.

In Fig. 2.1, $(t_2, t_3) \in \bar{D}$ holds although there are no concurrent events with the labels t_2 and t_3 in the branching process B of Fig. 2.1 but there are conflicting events t_2^1 and t_3^1. In Fig. 2.1, $(t_1, t_3) \in \bar{D}$ and the events t_1^1 and t_3^1 are concurrent. So, the independency relation $\bar{D}$ does not distinguish between concurrent and conflicting transitions. However, it is not necessary to distinguish between concurrent and conflicting transitions in a trace because a trace does not contain any transitions that are in conflict. On the other hand, when reasoning about the relation of different traces it is necessary to distinguish between concurrent and conflicting events. While the safety property of a winning strategy can be verified considering only one trace at a time, the deadlock avoiding, determinism and justified refusal property require that multiple traces are checked jointly. In the model of asynchronous games on Zielonka automata, this is avoided by defining a strategy in a way such that the justified refusal property is always satisfied and most commonly using a reachability property (or parity winning condition etc.) instead of the safety, deadlock avoiding and determinism property. Informally speaking, transferring that definition of a strategy to Petri games defines a strategy as a mapping from the set of system conditions of the unfolding to the power set of transitions of the underlying net, i.e. each system condition is mapped to a set of transitions that are always allowed to be fired. Essentially, this is how we define a strategy in Def. 2.32. If all system conditions in the preset of an event have the label of that event in their set of allowed transitions that event is added to the branching process of the strategy. This is equivalent to a strategy satisfying the winning properties as introduced in Lemma 2.34.

The main difference between the two game models is the distinction between controllable and uncontrollable places in a Petri game, as opposed to the distinction between controllable and uncontrollable actions in an asynchronous game where the actions in an asynchronous game correspond to the transitions in the Petri game. The formalism of the Petri net game model is more general in the sense that tokens can be created or deleted while the number of processes is fixed in a Zielonka automata. However, restricting the set of tokens to a fixed number of tokens and their own partitions as for token-partitioning nets reduces the formal complexity significantly. Therefore, the results here are presented for games with underlying nets that are token-partitioning.

2.4 Labelled Transition Systems

In Chapter 5, strategies with finite causal memory are considered. The state space of such a strategy is expressed as a *labelled transition system*. A *labelled transition system* (abbr. LTS) is 4-tuple (Q, q_I, Σ, δ), where Q is the set of states, q_I is the initial state, Σ is the set of inputs and $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation. For $(q, a, q') \in \delta$, we

write $q \xrightarrow{a} q'$. A *firing sequence* of an LTS T is a (possibly empty) sequence of inputs $a_1 \dots a_n$ such that there exist $q_1, \dots, q_n \in Q$ where $q_I \xrightarrow{a_1} q_1 \xrightarrow{a_2} q_2 \dots \xrightarrow{a_n} q_n$.

Chapter 3

Decidability Results

In this chapter, we show that the synthesis problem for token-partitioning Petri games with up to 4 players under a global safety condition is decidable in non-deterministic exponential time (NEXP), and NP-complete in the 2-player case. A Petri game G is called a *k-player Petri game*, $k \in \mathbb{N}$, if and only if for all reachable markings $M \in \mathcal{R}(G)$ it holds that $|M| \leq k$. 2-player Petri games can be seen as a natural generalisation of infinite games on graphs with a safety winning condition in the sense that a graph game has only one token moving in the graph while in a 2-player Petri game two tokens move. Additionally to the number of tokens, this generalisation introduces communication between the two tokens.

We establish decidability by identifying cuts where a winning strategy can be repeated such that a finite prefix of a winning strategy is sufficient to construct a (possibly infinite) winning strategy by repeating parts of this finite prefix. The idea of finding such cuts where a winning strategy can be repeated is the same idea used in [FG18] and [FO17]. However, opposed to [FG18, FO17] we do not reduce Petri games to 2-player games on finite graphs.

A 2-player game on a finite graph is an infinite game on a directed graph where the set of vertices is divided into a set of system vertices and a set of environment vertices. opposed to Petri game, there always exists exactly one token. If the token is on an environment vertex the environment player chooses an edge among the outgoing edges of that vertex to which vertex the token moves. Then, the player to whom the next vertex belongs chooses an outgoing edge from that vertex, and so on. Typically, the goal of the system player is to avoid certain vertices or to reach a certain vertex. Both players have exact complete knowledge about the game state, i.e. they know the current vertex and the sequence of vertices which have been visited so far. Such games with a safety condition are decidable in less than quadratic time in the number of vertices and edges of the graph. There also exist more complex winning conditions such as Büchi or parity conditions.

So, instead of a reduction to 2-player games with complete information we adapt the algorithm in [ERV02] where a complete finite prefix of a Petri net is constructed. There, completeness is in the sense of reachability, i.e. a complete prefix contains all

reachable markings of the net. More precisely, the complete prefix is a branching process and for every reachable marking in the net there exists a cut in the branching process such that the labels of the cut are the reachable marking. The algorithm constructing a complete finite prefix builds the unfolding of a net event by event while checking if a cut-off criterion is satisfied. If the cut-off criterion is satisfied for an event the event is not added to the prefix because the cut-off criterion indicates that every reachable marking in the future of that event is also reachable in the future of an event already in the prefix. There, the cut-off criterion is satisfied if the last known markings of two events coincide. However, this cut-off criterion is not sufficient for 4-player Petri games and even for 2-player Petri games. We define suitable cut-off criteria for the 2-player and the 4-player case. Those are refinements of the cut-off criterion for reachability [ERV02] since the equality of the last known markings is a necessary condition for our two cut-off criteria. We construct a winning prefix of a strategy analogously to the construction of a complete finite prefix with an adapted cut-off criterion. We show that a winning strategy exists if such a winning prefix exists. For that, we extend a winning prefix by copying parts of it to be the future of the cut-off events.

Informally speaking, there are two key factors which make the synthesis problem for 4-player Petri games decidable. Firstly, if there exists a bound in the unfolding of a net such that for all nodes the number of concurrent events is less than that bound the synthesis problem for Petri games is decidable [HO22]. Secondly, if there are only 4 tokens in a net the number of concurrent events which need to be distinguished by a strategy are bounded. Combining these two observations leads to decidability in the case of 4-player Petri games.

In [FO17], it is shown that the synthesis problem for Petri games with one environment player and an arbitrary bounded number of system players is EXPTIME-complete under a local safety condition. There, so called *mcuts* are introduced where a winning strategy can be repeated. Informally speaking, an mcut is a cut where all system players progressed maximally such that the one environment player participates in all transitions that could be fired next. This enables a reduction to a 2-player graph game with complete information because the system players do not have more information about the decisions of the environment player in the graph game than in the Petri game when planning from mcut to mcut. The system players of a Petri game are condensed to one player in the graph game. The other player in the graph game is the environment player. In [Hec21], this result is generalized to a global safety condition taking double exponential time to solve.

In [FG18], it is shown that Petri games with one system player and an arbitrary bounded number of environment players is EXPTIME-complete. There, a winning strategy of the one system player can be repeated if her last known marking repeats.

We identify two new kinds of cuts where a winning strategy can be repeated preserving a winning strategy. We call these cuts *partial repetition cuts* (Section 3.1) and *synchronisation equivalent cuts* (Section 3.3), respectively. Partial repetition cuts are used to solve 2-player Petri games as presented in Section 3.2. For the 4-player case, both kinds of cuts are used as presented in Section 3.3. As mentioned as a key factor

why the synthesis problem for 4-player Petri games is decidable, we use partial repetition cuts to show that the number of concurrent events which needs to be distinguished by a strategy is bounded in the 4-player case. The main contribution presented in this chapter is the nondeterministic exponential time (NEXP) complexity of 4-player Petri games. Also, showing that the synthesis problem for 2-player Petri games is NP-complete is a novel result.

In [Gim17], a related result is shown that the synthesis problem for 4-process control games on Zielonka automata with a local termination condition is decidable. The translation [BFH19] of this result to Petri games gives a decidability result for 4-player Petri games with a local termination condition. There is no complexity bound for the 4-player case given in [Gim17] and significant parts of the proof are left to the reader. The translation to Petri games already generates an exponential blow up in the number of nodes [BFH19].

Fig. 3.1 shows a Petri game G_0 with 4 players modelling the control of a room with two doors that must not be opened at the same time so that the two places of the bad marking $\{O_1, O_2\}$ are not reached simultaneously. For example, such a safety property is necessary for the control of a clean room, or a ship lock. Obviously, the control of a clean room is more complex and the example captures only a safety property of such a control. The example is kept simple to keep it at a reasonable size. The initial places of the two environment players are C_1 and C_2, respectively. The places C_1 and C_2 mean that the first door (C_1) and the second door (C_2) are closed. The first door remains closed on the places P_1 and W_1 and analogously does the second door. Both environment players can proceed to open their door via p_1 and p_2, respectively. After receiving a request to open the first door via transition r_1 the first system player on place S_{12} can decide to go on with communicating (transition t) or without communicating (transition n_1) to the second system player, then on place S_{22}. On place S_{13} the system player chooses between opening the door (o_1) or denying the request (d_1) for the environment player waiting on W_1. From place S_{14} the system player can close the door it has opened before (c_1). The second system player has the same options after receiving a request via r_2. After that, if both players open their doors the bad marking $\{O_1, O_2\}$ is reached. Every time both system players have received a request at least one system player has to deny the request in order to win the game.

Fig. 3.2 shows an initial part of the unfolding of the Petri game G_0 in Fig. 3.1. The nodes that are not greyed out form an initial part of a winning strategy $\sigma_{G_0} = (N, h)$. The homomorphism h is defined as $h(x_j^i) = x_j$ for all $x \in \mathcal{P} \cup \mathcal{T}$, i.e. by removing the superscript. Here, the system players agree on communicating via t^1. The first player decides to open its door via transition o_1^2 while the other door remains closed via transition d_2^2. After the transition c_1^2 the first door is closed again. The following is not shown in Fig. 3.2 anymore: after both players have received another opening request via r_1^2 and r_2^2, respectively, the winning strategy could continue opening door 2 and keeping door 1 closed since the different causal histories allow different decisions.

According to the definition of a winning strategy it is also possible that a winning strategy denies all requests to open a door. Extending the Petri game and adding addi-

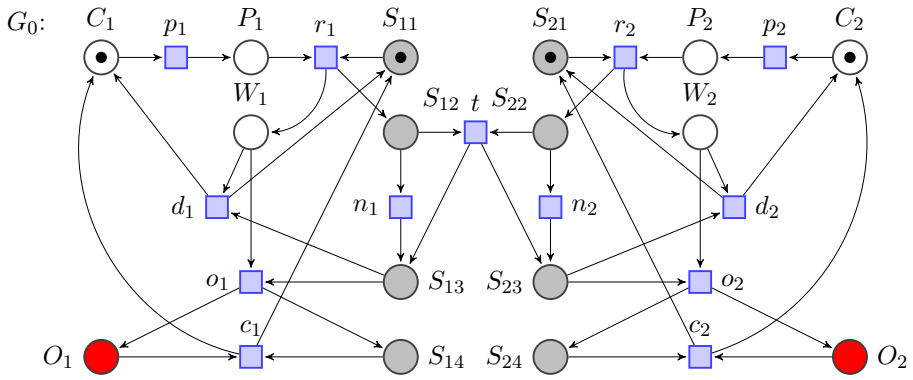


Figure 3.1: A 4-player Petri game G_0. The bad marking $\{O_1, O_2\}$ is reached if the system players decide to open both doors simultaneously.

tional bad markings if both requests are denied prevents such a solution. This is not done here to keep the example small. Furthermore, the definition of a winning strategy does not imply any fairness condition that both doors are opened eventually. A modification of this Petri game where a winning strategy satisfies some fairness condition would be significantly larger.

3.1 Partial Repetition

In this section, *partial repetition cuts* are introduced where a winning strategy can be repeated preserving a strategy to be winning. The basic idea is that the future of one partial repetition cut is copied to be the (afterwards isomorphic) future of a corresponding partial repetition cut. Therefore, partial repetition cuts come in pairs and are referred to as pairs. The cuts of partial repetition cuts are last known cuts of events. Therefore, such a pair of events is referred to as a pair of partial repetition events or just two partial repetition events.

The copying process works in both directions. However, the cuts are ordered w.r.t. a total adequate order and it is always copied from the lesser cut to the greater cut (w.r.t. the total adequate order). By convention, a pair of partial repetition cuts is also referred to as a partial repetition and we say a node x is preceded by a partial repetition if there exists a pair of partial repetition cuts (e_1, e_2), $e_1 \prec e_2$ such that $e_2 \leq x$. These cuts are used to construct a winning strategy from a given bounded prefix of a winning strategy by copying from the prefix over and over again.

Informally speaking, a partial repetition occurs, if a subset of players take a joint transition, then proceed without communicating to any of the remaining other players until all players of this subset synchronise at a joint transition for a second time. Then, if the last known markings are the same after both joint transition, a partial repetition occurred.

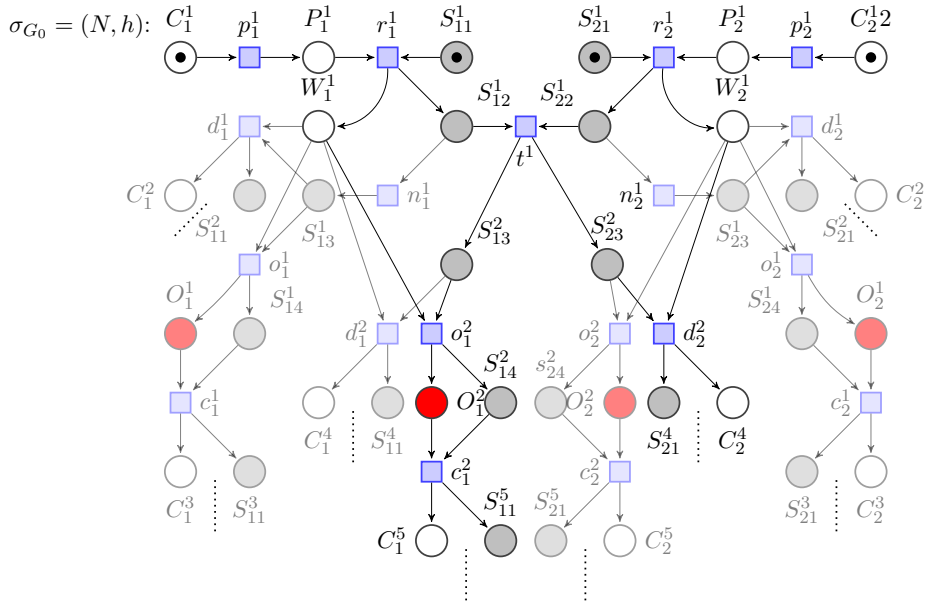


Figure 3.2: An initial part of the unfolding of the Petri game G_0 in Fig. 3.1. The branching process σ_{G_0} contains all nodes that are not greyed out.

Definition 3.1 (Partial repetition cuts). Let $e_1, e_2 \in \mathcal{T}$ be two events of a branching process B. The cuts $lkc(e_1)$ and $lkc(e_2)$ are *partial repetition cuts*, denoted $prc(B, e_1, e_2)$, if $lkm(e_1) = lkm(e_2)$ and $lkc(e_1) \setminus post(e_1) = lkc(e_2) \setminus post(e_2)$. ◁

Example 3.2 (Partial repetition cuts). In Fig. 3.3, a pair of partial repetition cuts in a branching process of the game G_0 in Fig. 3.1 is shown. In the depicted branching process, the second system player denies (d_2^1) the first request (r_2^1) to open the door without communicating with the other system player via transition t (in G). Here, the last known cuts of r_2^1 and r_2^2 are $lkc(r_2^1) = \{C_1^1, S_{11}^1, S_{22}^1, W_2^1\}$ and $lkc(r_2^2) = \{C_1^1, S_{11}^1, S_{22}^2, W_2^2\}$, so $lkm(r_2^1) = lkm(r_2^2)$. A partial repetition, $prc(B_{prc}, r_2^1, r_2^2)$, occurs since $lkc(r_2^1) \setminus post(r_2^1) = \{C_1^1, S_{11}^1\} = lkc(r_2^2) \setminus post(r_2^2)$ also holds. Therefore, if this branching process is a subprocess of a winning strategy, a strategy could proceed after r_2^2 in the same way as after r_2^1. That is to also deny all future requests after r_2^2 and still be a winning strategy. In this example $r_2^1 \leq r_2^2$ holds. However, it is also possible that the events of a pair of partial repetition cuts are in conflict. It is not possible that the two events of a partial repetition are concurrent. This would contradict the fact that the underlying net is a safe net. This is formally shown in the next lemma. ◁

As mentioned in the example of partial repetition cuts, Example 3.2, the two events of partial repetition cuts are either causally related or in conflict, i.e. they cannot be concurrent. Informally speaking, this relation can be seen particularly well for token-partitioning Petri nets. The condition of a pair of partial repetition events $prc(B, e_1, e_2)$

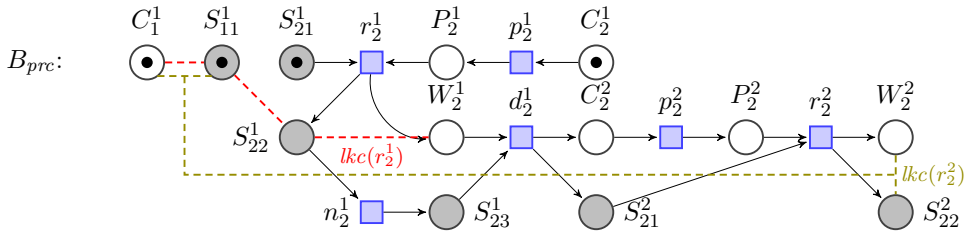


Figure 3.3: Example of a partial repetition $prc(B_{prc}, r_2^1, r_2^2)$ in a branching process B_{prc} of game G_0 in Fig. 3.1.

that $lkc(e_2) \setminus post(e_2) = lkc(e_1) \setminus post(e_1)$ implies that the tokens which participate in e_1 and e_2, respectively, have the same indices. If e_2 and e_1 are concurrent there exists an index such that there are two tokens of that index participating in e_1 and e_2, respectively. This contradicts the property of a token-partitioning net that every reachable marking contains exactly one token in each partition. The lemma follows.

Lemma 3.3 (Relation of partial repetitions). *For a branching process B of a net N_0 and $e_1, e_2 \in \mathcal{T}$, if $prc(B, e_1, e_2)$ it holds that e_1 and e_2 are causally related or in conflict.*

Proof. If $e_1 = e_2$, the events are causally related and the property holds. In the following, we assume $e_1 \neq e_2$.

By definition of $prc(B, e_1, e_2)$, it holds that $lkm(e_1) = lkm(e_2)$ and $lkc(e_1) \setminus post(e_1) = lkc(e_2) \setminus post(e_2)$. By Lemma 2.22, $post(e_1) \subseteq lkc(e_1)$. Now, it holds that $h(post(e_1)) = h(lkc(e_1)) \setminus h(lkc(e_1) \setminus post(e_1)) = h(lkc(e_2)) \setminus h(lkc(e_2) \setminus post(e_2)) = h(post(e_2))$. If e_1 and e_2 are concurrent, $[e_1] \cup [e_2]$ is a configuration by Lemma 2.15. Then, $h(Cut([e_1] \cup [e_2]))$ is a reachable marking in N_0. Since e_1 and e_2 are concurrent, from Lemma 2.22 follows that $post(e_1) \subseteq Cut([e_1] \cup [e_2])$ and $post(e_2) \subseteq Cut([e_1] \cup [e_2])$. Since $e_1 \neq e_2$ and $h(post(e_1)) = h(post(e_2))$ the cut $Cut([e_1] \cup [e_2])$ contains two conditions with the same label. Since $h(Cut([e_1] \cup [e_2]))$ is a reachable marking, this is a contradiction to N_0 being a safe net. Therefore, it holds that e_1 and e_2 are causally related or in conflict. □

Lemma 3.3 states that the two events of a pair of partial repetition events cannot be concurrent. Note that the two events are either causally related or in conflict, not both. Two events being causally related and in conflict implies that one of the events is in self-conflict which is prohibited by the definition of an occurrence net (Def. 2.2).

By convention, the used total adequate order is always the adequate order presented in [ERV02] which is the total adequate order of Def. 2.13, where the total order $\ll$ on the transitions of the net is the alphabetical order with indented superscript and subscript. The subscript is indented first followed by the superscript. However, in the following examples no transition in the underlying net has both subscript and superscript.

Later, we show that a winning strategy can be constructed starting with a suitable winning prefix which has polynomial size in the case of 2-player Petri games. The algorithm to construct such a winning prefix is similar to the algorithm in [ERV02] where

a complete finite prefix of the unfolding of a Petri net is constructed. In [ERV02]. the complete finite prefix B_{prefix} is a branching process of the net N and it is complete regarding reachability of the Petri net, i.e. for every reachable marking $M \in \mathcal{R}(N)$ there exists a cut $K \subseteq \mathcal{P}_{prefix}$ such that $h_{prefix}(K) = M$.

The algorithm in [ERV02] constructing a complete finite prefix starts with a branching process which only consists of its initial marking. In each step, the branching process is extended by an event unless that event satisfies a cut-off criterion. For reachability, this cut-off criterion is the *last known marking*, i.e. an event is not added to the prefix if there already exists an event with the same last known marking. The algorithm ends if all events which are possible extensions of the prefix satisfy the cut-off criterion.

This cut-off criterion for reachability in Petri nets used in [ERV02] is not sufficient for 2-player Petri games which the next example illustrates. For 2-player Petri games, it is sufficient to use partial repetition cuts as the cut-off criterion, i.e. an event is not added to the prefix if there already exists another event such that those two events are a pair of partial repetition cuts. Comparing those two cut-off criteria we notice that partial repetition requires a second condition besides the equality of the last known markings. For partial repetitions, the last known cuts must also be equivalent except for the postsets of the two events of a pair of partial repetition events.

Example 3.4 (Reachability cut-off vs. 2-player cut-off). In Fig. 3.4, a Petri game "Double Same Choice" which is denoted as G_2 and a winning strategy σ_2 of G_2 are shown. In order to win the game, both system players have to repeat the choice (a or b) of the environment player. The system places S_2^1 and S_2^2 in σ_2 have the same last known marking which is $\{S_1, S_2\}$. However, the system player on S_2^2 cannot copy the choice a of the system player on S_2^1 because otherwise the bad marking $\{A_2', B'\}$ is reachable. This shows that the cut-off criterion for reachability in Petri nets is not sufficient for 2-player Petri games. Note that the events r_a^1 and r_b^1 are no partial repetition events since the condition S_1^1 in the last known cut of S_2^1 and the condition S_1^2 in the last known cut of S_2^2 differ. ◁

In the following, some properties of partial repetitions are presented. The first lemma states that if there is a partial repetition $prc(B, e_1, e_2)$ for two events e_1, e_2, a subprocess homomorphism from the future of $lkc(e_2)$ in B to the future of $lkc(e_1)$ (or vice versa) is the identity function restricted to the set of nodes that are concurrent to e_2.

Reminder. A total adequate order on the set of finite configurations of a branching process induces a bijective mapping between the natural numbers and the set of finite configurations. There exists a total adequate order for every branching process of a safe net. This enables proofs by induction over the set of finite configurations of a branching process. The configuration C_n in the context of an inductive proof is the configuration such that there are n equal or lesser configurations, i.e. $C_1 = \emptyset$ is the base case of the induction, and the running variable of the induction is n. For the induction assumption, we stipulate that the property which we want to show holds for all configurations C that are equal or lesser than C_n w.r.t. the total adequate order. For the induction step, we show that the property holds for the next (after C_n) configuration regarding the total

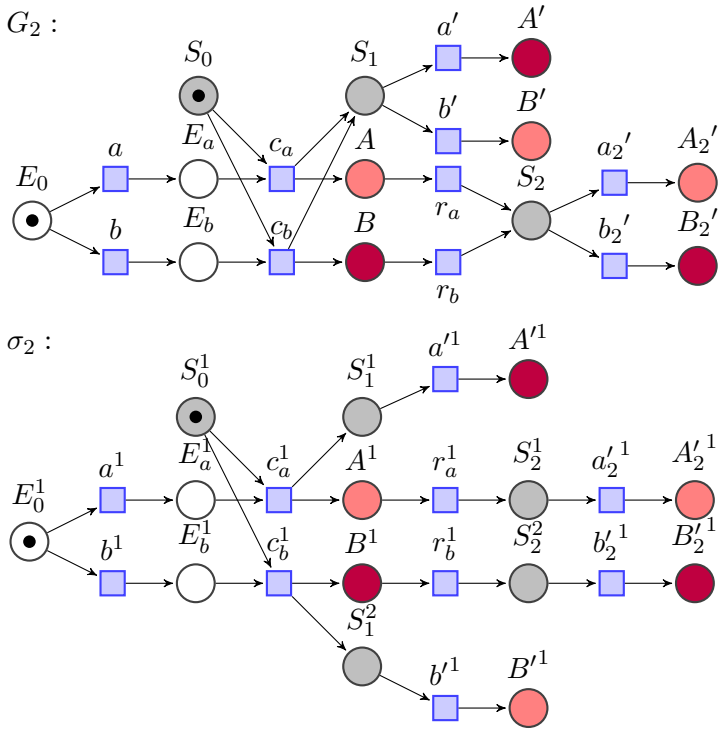


Figure 3.4: The Petri game "Double Same Choice". Each marking where both places are either red or purple is a bad marking.

adequate order which is C_{n+1}. Such a proof is called a proof *by induction over the set of finite configuration of a branching process*. The branching process may vary and is explicitly named at the start of a proof.

Lemma 3.5 (Identity of partial repetition cuts). *Let B be a branching process and $e_1, e_2 \in \mathcal{T}$ such that $prc(B, e_1, e_2)$ holds. If $Fut(B, [e_2])$ is a subprocess of $Fut(B, [e_1])$, for the subprocess homomorphism $h^{Fut(B,[e_2])}_{Fut(B,[e_1])}$ it holds that*

$$\forall x \in \mathcal{P}_{Fut(B,[e_2])} \cup \mathcal{T}_{Fut(B,[e_2])} : (x||e_2 \implies h^{Fut(B,[e_2])}_{Fut(B,[e_1])}(x) = x).$$

Proof by induction over the set of finite configurations of $Fut(B, e_2)$. For the induction base case, the conditions of the last known cut which are concurrent to e_2 are considered: For $b \in lkc(e_2) \setminus post(e_2)$ it holds that $h^{Fut(B,[e_2])}_{Fut(B,[e_1])}(b) = b$ because $lkc(e_2) \setminus post(e_2) = lkc(e_1) \setminus post(e_1)$ by definition of $prc(B, e_1, e_2)$. (and because $h^{Fut(B,[e_2])}_{Fut(B,[e_1])}$ is a branching process homomorphism which preserves the labels of the conditions.)

Induction assumption: For all $x \in \mathcal{P}_{Fut(B,[e_2])} \cup \mathcal{T}_{Fut(B,[e_2])}$ if $x||e_2$ and $[x] \preceq C_n$ it holds that $h^{Fut(B,[e_2])}_{Fut(B,[e_1])}(x) = x$.

Induction step: if there exists $e \in C_{n+1}$ such that there is no lesser configuration that contains e, i.e. $[e] = C_{n+1}$, and $e||e_2$ we proceed as follows: by induction assumption it holds that for any node x that $h^{Fut(B,[e_2])}_{Fut(B,[e_1])}(x) = x$ if $x||e_2$ and $[x] \preceq C_n$. Since $e||e_2$ holds it also holds that for all $b \in pre(e) : b||e_2$. For all $b \in pre(e)$ holds that $[b] \preceq C_n$. Thus, it holds that for all $b \in pre(e) : h^{Fut(B,[e_2])}_{Fut(B,[e_1])}(b) = b$. Using the subprocess homomorphism of $h^{Fut(B,[e_2])}_{Fut(B,[e_1])}$ yields: $pre(h^{Fut(B,[e_2])}_{Fut(B,[e_1])}(e)) = h^{Fut(B,[e_2])}_{Fut(B,[e_1])}(pre(e)) = pre(e)$, which implies $h^{Fut(B,[e_2])}_{Fut(B,[e_1])}(e) = e$.

Now, it holds that $h^{Fut(B,[e_2])}_{Fut(B,[e_1])}(post(e)) = post(h^{Fut(B,[e_2])}_{Fut(B,[e_1])}(e)) = post(e)$ which implies that for all $b \in post(e)$ it holds that $h^{Fut(B,[e_2])}_{Fut(B,[e_1])}(b) = b$. □

For example in Fig. 3.3, the subprocess homomorphism $h^{Fut(B_{prc},[r_2^2]_{prc})}_{Fut(B_{prc},[r_2^1]_{prc})}$ from the future of $lkc_{prc}(r_2^2)$ to the future of $lkc_{prc}(r_2^1)$ maps C_1^1 and S_{11}^1 to themselves, respectively. From there on, it is easy to see that all nodes which are concurrent to r_2^2 (e.g. the next causal successors of C_1^1 and S_{11}^1) are mapped to themselves.

The next lemma states that the set of concurrent events of e_1 and e_2 in the unfolding are the same if $prc(u, e_1, e_2)$ holds. This is closely related to the previous Lemma 3.5. By convention, the isomorphism $I^{Fut(u,[e_2]_u)}_{Fut(u,[e_1]_u)}$ is abbreviated as $I^{[e_2]}_{[e_1]}$.

Lemma 3.6 (Concurrent events of partial repetition cuts). *For a net N_0 and $e_1, e_2 \in \mathcal{T}_u$, if $prc(u, e_1, e_2)$ holds it holds that for all $e \in \mathcal{T}_u : e||e_2 \implies e||e_1$.*

Proof. Consider the isomorphism $I^{[e_2]}_{[e_1]}$. By Lemma 2.19, the futures $Fut(u, [e_2]_u)$ and $Fut(u, [e_1]_u)$ contain all events that are concurrent to e_2 and e_1, respectively. According to Lemma 3.5, it holds that $I^{[e_2]}_{[e_1]}(e) = e$ if $e||e_2$. By Lemma 2.18, the isomorphism preserves concurrency. Since $e||e_2$ implies $\forall b \in post(e_2) : e||b$ (and vice versa), we get $e||e_2 \implies e||post(e_2) \implies I^{[e_2]}_{[e_1]}(e)||I^{[e_2]}_{[e_1]}(post(e_2)) \implies e||post(e_1) \implies e||e_1$. (For any node $x \in \mathcal{T}_u \cup \mathcal{P}_u$ and a set of nodes $X \subseteq \mathcal{T}_u \cup \mathcal{P}_u$, $x||X$ holds if and only if $\forall x' \in X : x||x'$ holds.) □

The next lemma is more intrigued. Assuming for three events e, e_1, e_2 that $prc(u, e_1, e_2)$ and $e||e_2$ hold, it states the following: Applying the isomorphism $I^{[e_2]}_{[e_1]}$ on a cut that is in the future $Fut(u, [e]_u)$ and in the future $Fut(u, [e_2]_u)$ yields a cut that is still in the future $Fut(u, [e]_u)$. This lemma is used later on when it is necessary to show that another isomorphism can be applied after applying $I^{[e_2]}_{[e_1]}$, e.g. an isomorphism $I^{[e]}_{[e']}$ from the future of e to the future of some suitable e'.

Lemma 3.7 (Futures of partial repetition cuts). *Let N_0 be a net, $e, e_1, e_2 \in \mathcal{T}_u$ such that $prc(u, e_1, e_2)$, $e||e_2$, and $C \subseteq \mathcal{T}_u$ a configuration. If $Cut_u(C) \subseteq \mathcal{P}_{Fut(u,[e_2]_u)} \cap \mathcal{P}_{Fut(u,[e]_u)}$ it holds that $I^{[e_2]}_{[e_1]}(Cut_u(C)) \subseteq \mathcal{P}_{Fut(u,[e]_u)}$.*

$$\begin{array}{ccc} Fut(u, Cut([e_2] \cup [e_2'])) & \xrightarrow{I^{[e_2]}_{[e_1]}} & Fut(u, Cut([e_1]_u \cup [e_2']_u)) \\ \Big\downarrow{\scriptstyle I^{[e_2']}_{[e_1']}} & & \Big\downarrow{\scriptstyle I^{[e_2']}_{[e_1']}} \\ Fut(u, Cut([e_2]_u \cup [e_1']_u)) & \xrightarrow[I^{[e_2]}_{[e_1]}]{} & Fut(u, Cut([e_1]_u \cup [e_1']_u)) \end{array}$$

Figure 3.5: Commutative diagram of the isomorphisms $I^{[e_2]}_{[e_1]}$ and $I^{[e_2']}_{[e_1']}$ if the events e_2 and e_2' of the unfolding are concurrent and $prc(u, e_1, e_2)$ and $prc(u, e_1', e_2')$ hold.

Proof. Using the definition of the future of a configuration, we show $\forall b' \in Cut_u(C) : \forall b \in lkc_u(e) : I^{[e_2]}_{[e_1]}(b')||b \vee b \leq I^{[e_2]}_{[e_1]}(b')$.

Case $e_2||b'$**:** By Lemma 3.5, $I^{[e_2]}_{[e_1]}(b') = b'$ and $b' \in \mathcal{P}_{Fut(u,[e]_u)}$ holds by assumption.

Case $e_2 \leq b'$**:** There exists a $b_{e_2} \in post(e_2)$ such that $b_{e_2} \leq b'$. By Lemma 2.16, $I^{[e_2]}_{[e_1]}(b_{e_2}) \leq I^{[e_2]}_{[e_1]}(b')$ holds. Note that $I^{[e_2]}_{[e_1]}(b_{e_2}) \in post(e_1)$ holds.

By Lemma 3.6, $e||e_2$ implies $e||e_1$. Then, by Lemma 2.19, e_1 is in $\mathcal{T}_{Fut(u,[e]_u)}$. Thus, for all $b \in lkc_u(e)$ either $b||e_1$ or $b \leq e_1$ holds.

Subcase $b||e_1$**:** By Lemma 3.5, $I^{[e_2]}_{[e_1]}(b) = b$ holds. Thus, $b'||b \vee b \leq b'$ implies $I^{[e_2]}_{[e_1]}(b')||b \vee b \leq I^{[e_2]}_{[e_1]}(b')$ by Lemma 2.18 and Lemma 2.16.

Subcase $b \leq e_1$**:** Here, $b \leq I^{[e_2]}_{[e_1]}(b_{e_2}) \leq I^{[e_2]}_{[e_1]}(b')$ holds by Lemma 2.16. □

For the next lemma, it is shown that for two pairs $(e_1, e_2), (e_1', e_2')$ of partial repetition events the isomorphisms $I^{[e_2]}_{[e_1]}$ and $I^{[e_2']}_{[e_1']}$ commute if e_2 and e_2' are concurrent. This lemma is essential for the construction of a winning strategy. Already for the 2-player case, it may occur that there are two pairs of partial repetition events (e_1, e_2), (e_1', e_2') where e_2 and e_2' are concurrent. The construction of a winning strategy copies the future of one of those events. This lemma is used to show that both choices from which future is copied lead to the same strategy.

In Fig. 3.5, a commutative diagram for the isomorphisms $I^{[e_2]}_{[e_1]}$ and $I^{[e_2']}_{[e_1']}$ illustrates the statement of the following Lemma 3.8.

Lemma 3.8 (Partial repetition homomorphisms commute). *Let N_0 be a net and $e_1, e_2, e_1', e_2' \in \mathcal{T}_u$ events of its unfolding. If $e_2||e_2'$, $prc(u, e_1, e_2)$ and $prc(u, e_1', e_2')$ hold, for all conditions $b \in \mathcal{P}_{Fut(u,[e_2]_u)} \cap \mathcal{P}_{Fut(u,[e_2']_u)}$ holds that $I^{[e_2]}_{[e_1]}(I^{[e_2']}_{[e_1']}(b)) = I^{[e_2']}_{[e_1']}(I^{[e_2]}_{[e_1]}(b))$.*

Proof by induction over the set of finite configurations of $Fut(u, Cut_u([e_2]_u \cup [e_2']_u))$. Because e_2 and e_2' are concurrent $[e_2]_u \cup [e_2']_u$ is a configuration in u by Lemma 2.15.

Induction base case: For $b \in Cut([e_2]_u \cup [e_2']_u)$ the following cases are distinguished:

Case $b \in post(e_2)$**:** We show that $I^{[e_2]}_{[e_1]}(I^{[e_2']}_{[e_1']}(b)) = I^{[e_2]}_{[e_1]}(b) = I^{[e_2']}_{[e_1']}(I^{[e_2]}_{[e_1]}(b))$, i.e. the isomorphism $I^{[e_2']}_{[e_1']}$ is the identity before and after applying $I^{[e_2]}_{[e_1]}$.

It holds that $b||e_2'$ since $e_2||e_2'$. By Lemma 3.5, $I^{[e_2']}_{[e_1']}(b) = b$. Thus, $I^{[e_2]}_{[e_1]}(I^{[e_2']}_{[e_1']}(b)) = I^{[e_2]}_{[e_1]}(b)$ holds.

By Lemma 2.18, $b||e_2'$, implies $I^{[e_2]}_{[e_1]}(b)||I^{[e_2]}_{[e_1]}(e_2')$. By Lemma 3.5, $I^{[e_2]}_{[e_1]}(e_2') = e_2'$. Again, by Lemma 3.5, $I^{[e_2']}_{[e_1']}(I^{[e_2]}_{[e_1]}(b)) = I^{[e_2]}_{[e_1]}(b)$. Thus, $I^{[e_2]}_{[e_1]}(I^{[e_2']}_{[e_1']}(b)) = I^{[e_2]}_{[e_1]}(b) = I^{[e_2']}_{[e_1']}(I^{[e_2]}_{[e_1]}(b))$ holds.

Case $b \in post(e_2')$**:** This case is analogous to the previous case swapping e_2 and e_2'.

Case $b \in lkc(e_2) \setminus post(e_2) \vee b \in lkc(e_2') \setminus post(e_2')$**:** Here, b is concurrent to e_2 or e_2' and the property holds analogously to one of the two previous cases.

Induction assumption: The property holds for all finite configurations C of u such that $C \preceq C_n$.

Induction step: Let C_{n+1} be the next configuration regarding $\preceq$. If there exists an $e \in C_{n+1}$ such that there is no lesser configuration that contains e, i.e. $[e] = C_{n+1}$, we proceed as follows. Otherwise the property holds by induction assumption for all conditions in the cut of C_{n+1}.

Using the property of a net homomorphism that it preserves the presets (and postsets) of events, it holds that $pre(I^{[e_2']}_{[e_1']}(I^{[e_2]}_{[e_1]}(e))) = I^{[e_2]}_{[e_1]}(I^{[e_2']}_{[e_1']}(pre(e))) = I^{[e_2']}_{[e_1']}(I^{[e_2]}_{[e_1]}(pre(e))) = pre(I^{[e_2']}_{[e_1']}(I^{[e_2]}_{[e_1]}(e)))$ by induction assumption and $I^{[e_2]}_{[e_1]}(I^{[e_2']}_{[e_1']}(e)) = I^{[e_2']}_{[e_1']}(I^{[e_2]}_{[e_1]}(e))$ follows. Thus the property follows also for all conditions in the cut of C_{n+1}. □

We conclude this subsection with an observation regarding the isomorphism $I^{[e_2]}_{[e_1]}$ if $prc(u, e_1, e_2)$ holds. We observe that this isomorphism restricted to the postset of e_2 is a bijection between $post(e_2)$ and $post(e_1)$.

Lemma 3.9 (Postset bijection of partial repetition events). *Let N be a net and $e_1, e_2 \in \mathcal{T}_u$ such that $prc(u, e_1, e_2)$. It holds that $I^{[e_2]}_{[e_1]}(post(e_2)) = post(e_1)$.*

Proof. Since $lkc_u(e_2) \setminus post_u(e_2) = lkc_u(e_1) \setminus post_u(e_1)$ and $lkm_u(e_2) = lkm_u(e_1)$ hold it follows that $h_u(post(e_2)) = h_u(post(e_1))$. Since $I^{[e_2]}_{[e_1]}$ is a branching process isomorphism $I^{[e_2]}_{[e_1]}(post(e_2)) = post(e_1)$ follows. □

3.2 2-player Petri Games

In this section, NP-hardness is established by a reduction of the 3-SAT problem to 2-player Petri games. Afterwards, a construction of a winning strategy of a 2-player game is introduced where the construction starts with a winning prefix that has polynomially bounded size.

The 3-SAT problem is well known to be an NP-complete problem [Coo71] (It is NP-hard and in NP). Here, the exact 3-SAT problem is used where every clause consists of *exactly* 3 literals (and not at most 3 literals) which is also NP-complete [Coo71]. An instance of the exact 3-SAT problem is the boolean satisfiability problem of a boolean formula F in conjunctive normal form where every clause consist of exactly three literals,

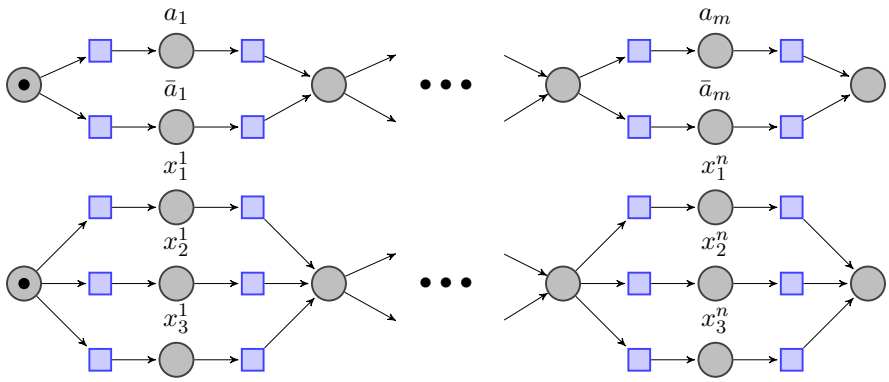


Figure 3.6: 3-SAT problem $F = (x_1^1 \vee x_2^1 \vee x_3^1) \wedge \ldots \wedge (x_1^n \vee x_2^n \vee x_3^n)$ with literals $a_1, \ldots, a_m$ as a Petri game.

e.g. $F = (x_1^1 \vee x_2^1 \vee x_3^1) \wedge \ldots \wedge (x_1^n \vee x_2^n \vee x_3^n)$, where x_j^i, $i = 1, \ldots, n$ and $j = 1, 2, 3$, is a positive or negative (when overlined) literal from a set of literals $\{a_1, \ldots, a_m\}$. A boolean assignment is a mapping $\mathcal{A} : \{a_1, \ldots, a_m\} \to \{\mathit{true}, \mathit{false}\}$. The formula F is satisfiable if there exists a boolean assignment $\mathcal{A}$ such that F evaluates to *true* under $\mathcal{A}$.

Lemma 3.10 (NP-hardness of 2-player Petri games). *There exists a polynomial-time reduction of the exact 3-SAT problem to the synthesis problem of 2-player Petri games.*

Proof. Let $F = (x_1^1 \vee x_2^1 \vee x_3^1) \wedge \ldots \wedge (x_1^n \vee x_2^n \vee x_3^n)$ be an instance of the exact 3-SAT problem in conjunctive normal form where all clauses consist of exactly three literals and where x_j^i, $i = 1, \ldots, n$ and $j = 1, 2, 3$, is a positive or negative (when overlined) literal from a set of literals $\{a_1, \ldots, a_m\}$.

Fig. 3.6 shows the Petri game of F where a winning strategy exists if and only if F is satisfiable. In Fig. 3.6, the top system player chooses the truth values for $a_1, \ldots a_m$ sequentially. For example, taking the upper transition to the place a_1 assigns the value *true* to a_1 while going to the place $\bar{a}_1$ assigns the value *false* to a_1. The bottom system player has to choose a literal for each clause according to the truth values chosen by the top player. The bad markings are $\{\{a_k, x_j^i\} \mid x_j^i = \bar{a}_k, j \in \{1, 2, 3\}, i \in \{1, \ldots, n\}, k \in \{1, \ldots, m\}\}$. Note that we assume $a = \bar{\bar{a}}$. So, every time the bottom player chooses a negation of a_i (or $\bar{a}_i$) chosen by the top system player a bad marking is reached, e.g. the top player chooses a_1 and the bottom player x_3^1 where $x_3^1 = \bar{a}_1$.

Thus, if F is satisfiable the top system player chooses the transitions according to a boolean assignment that satisfies F. The bottom system player chooses the literal that is true under that boolean assignment in each clause. This yields a winning strategy in the Petri game of F.

Conversely, the top player's choices of a winning strategy yield a boolean assignment that satisfies F since both system players do not know when the transitions of the other player are fired such that it has to be correct for all possible orders of execution. For

example, the top player could have chosen the truth value for a_m already before the bottom player chooses the literal for the first clause and the other way around. □

Note that the 2-player case is NP-hard even without an environment player. The matching upper bound is established next.

Starting with a strategy that is winning until partial repetition cuts are reached, a winning strategy is constructed by copying the winning parts of the strategy. Again, we fix the total adequate order $\preceq$ on the finite configurations of the unfolding as the total adequate order presented in [ERV02] (also introduced in Def. 2.13).

Before defining a winning two-player prefix the set of events until partial repetition cuts are reached is defined for a branching process. Therefore, 2-player cut-off events and 2-player non-cut-off events are introduced. The set of non-cut-off events of a branching process is sufficient to check if a given strategy is a winning prefix.

Definition 3.11 (2-player cut-off event). Let B be a branching process. An event $e \in \mathcal{T}$ is a *2-player cut-off event* if there exists $e' \in \mathcal{T}$ such that

1. $prc(B, e', e)$, and
2. $[e'] \prec [e]$.

◁

For an event $e \in \mathcal{T}$, the smallest (w.r.t. $\preceq$) e' that satisfies the properties 1 and 2 in the definition of a 2-player cut-off event (Def. 3.11) is denoted as $R(e)$. Later on, the 2-player cut-off events are used to determine where a winning prefix gets repeated.

Comparing the cut-off events used in [ERV02] for complete finite prefixes and the 2-player cut-off events, we notice that a 2-player cut-off event satisfied an additional condition. This condition is the condition of partial repetition events e_1, e_2, $prc(B, e_1, e_2)$, that $lkc(e_1) \setminus post(e_1) = lkc(e_2) \setminus post(e_2)$. Example 3.4 shows that this condition is necessary.

Definition 3.12 (2-player non-cut-off events). For a branching process B of a token-partitioning net with two tokens the set of *2-player non-cut-off events* $NCO(B)$ is the set $NCO(B) = \{e \in \mathcal{T} \mid \neg\exists e' \in \mathcal{T} : e' \leq e \wedge e' \text{ is a 2-player cut-off event.}\}$ ◁

So, a non-cut-off event is an event that is not causally preceded by a cut-off event. A winning 2-player prefix τ must be winning in all configurations that are a subset of the set of non-cut-off events $NCO(B_\tau)$.

Definition 3.13 (Winning 2-player prefix). A strategy τ is a *winning 2-player prefix* if for all configurations $C \subseteq \mathcal{T}_\tau$ it holds that if $C \subseteq NCO$ the branching process B_τ is winning in C. ◁

Example 3.14 (Winning 2-player prefix). A winning two-player prefix B_τ of the Petri game "Same Choice" of Fig. 2.2 is shown in Fig. 3.7. The last known cuts of the events r_a^1 and r_b^1 are partial repetition cuts, i.e. $prc(B_\tau, r_a^1, r_b^1)$ holds. The prefix

does not contain any events that are causal successors of r_b^1 since $[r_a^1]_\tau \prec [r_b^1]_\tau$ and $prc(B_\tau, r_a^1, r_b^1)$ hold and the strategy τ is chosen in a way that no more events are allowed than necessary to satisfy the property of a winning two-player prefix, i.e. $\tau(S_0^3) = \emptyset$ and $\tau(E_0^3) = \emptyset$. Analogously, the prefix ends after the events c_a^2 and c_b^2 because $prc(B_\tau, c_a^1, c_a^2)$ and $prc(B_\tau, c_b^1, c_b^2)$ hold, respectively. ◁

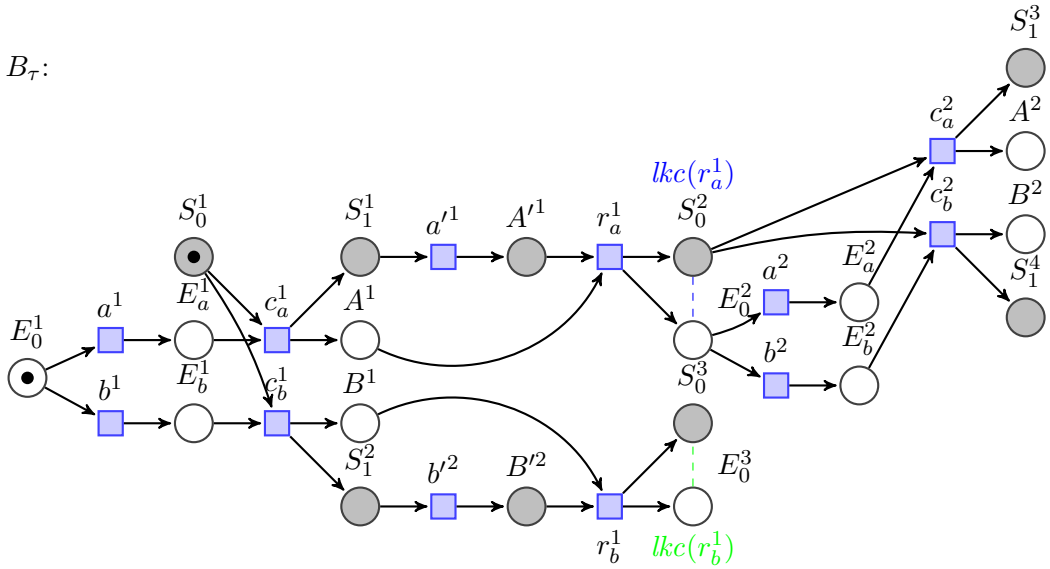


Figure 3.7: The branching process B_τ of a winning 2-player prefix τ of the game G, the "Same Choice" game, in Fig. 2.2.

For the Example 3.14 in Fig. 3.7, it is easy to see how the winning two-player prefix can be continued to be a winning strategy: the future of $[r_b^1]_\tau$ is constructed by copying the future of $[r_a^1]_\tau$. Analogously, the futures of $[c_a^2]_\tau$ and $[c_b^2]_\tau$ are copied from their partial repetitions $[c_a^1]_\tau$ and $[c_b^1]_\tau$, respectively.

The copying process is done step by step as an induction over set of finite configurations of the unfolding. In each step, the strategy is altered for the conditions whose local configuration is the next configuration regarding the adequate order, where τ_1 is the winning 2-player prefix. In the proof, an inductive sequence of strategies $\tau_1, \tau_2, \ldots$ is constructed. In the following τ_n is one of the constructed strategies and τ_{n+1} is the constructed strategy of the next induction step.

For example, consider the configuration $\{b^1, c_b^1, b'^2, r_b^1\}(= [r_b^1]_\tau)$ of B_τ in Fig. 3.7: this configuration violates the deadlock avoiding property. In the induction step of that configuration, $C_{n+1} = [r_b^1]_\tau$, the strategy τ_n is altered in the conditions S_0^3 and E_0^3 such that two events, one with label a and the other with label b are added in the postset of E_0^3 in the altered strategy τ_{n+1}. This is done because τ_{n+1} is defined as $\tau_{n+1}(E_0^3) = \tau(E_0^2) = \{a, b\}$ due to $I_{[r_a^1]}^{[r_b^1]}(E_0^3) = E_0^2$ (the nodes of B_τ are the same nodes

as in the unfolding). Since the configuration from which the strategy is copied satisfies the winning properties the next configuration ($[r_b^1]_\tau$) also satisfies the winning properties after the induction step. Here, the deadlock avoiding property is no longer violated after adding those two events. Note that the future of r_b^1 is constructed isomorphically to the future of r_a^1, i.e. the futures of r_b^1 and r_a^1 are isomorphic after the induction is completed. However, informally speaking, after induction step n the futures of r_a^1 and r_b^1 only coincide up to the configuration C_n. More precise, for the subprocess homomorphism $h^{Fut(u,[r_b^1])}_{Fut(u,[r_a^1])}$ from the future of r_b^1 to the future of r_a^1 it holds that for all conditions $b \in \mathcal{P}_{Fut(u,[r_b^1])}$ such that $[b] \preceq C_n$ the strategy τ_n is preserved, i.e. $\tau_n(b) = \tau_n(h^{Fut(u,[r_b^1])}_{Fut(u,[r_a^1])}(b)) = \tau(I^{[r_b^1]}_{[r_a^1]}(b))$.

The example does not cover the case when two players do not take a joint transition reaching different pairs of partial repetition cuts concurrently. In that case, there are two pairs of partial repetition cuts $prc(B, e_1, e_2)$ and $prc(B, e_1', e_2')$ where e_2 and e_2' are concurrent. (By Lemma 3.6, $e_2||e_1'$, $e_1||e_2'$ and $e_1||e_1'$ also hold then). If the cut of the configuration C_{n+1} in the induction step is both a cut in the future of e_2 and in the future of e_2', we need to choose from which future we copy the strategy. Here, the property of Lemma 3.8 that the subprocess homomorphisms of two pairs of such partial repetition cuts commute is crucial. If both events e_2 and e_2' are causal predecessors of the event we need to copy, it does not matter which partial repetition is used to copy the strategy, both ways result in the same strategy. If only one event of e_2 and e_2' is a causal predecessor we copy from the future of that event. Otherwise, we do not actually alter the strategy for concurrent conditions considering Lemma 3.5.

Before the construction of a winning strategy starting with a winning 2-player prefix is presented, the set of pairs of events is defined where the winning prefix has to be repeated.

Definition 3.15 (2-player partial repetitions). For a winning 2-player prefix τ, the set of repetitions is defined as the set $REP(B_\tau) = \{(R(e), e) \mid e \in \mathcal{T}_\tau \wedge \neg\exists e' \in \mathcal{T}_\tau : e' < e \wedge e' \text{ is a 2-player cut-off event.}\}$. ◁

Now, the construction of a winning 2-player strategy starting with a winning 2-player prefix is presented. The proof is by induction over the set of finite configurations of the unfolding. The induction property is split into three parts.

Property 1 states that the constructed winning strategy is winning in all finite configurations up to the n-th configuration regarding the total adequate order $\preceq$.

Property 2 states that the n-th constructed strategy τ_n is preserved by all isomorphism $I^{[e_2]}_{[e_1]}$ of repetitions $(e_1, e_2) \in REP(B_\tau)$ of the winning prefix B_τ if the local configuration of a condition b is less than the configuration C_n considered in the induction step, i.e. $\tau_n(b) = \tau_n(I^{[e_2]}_{[e_1]}(b))$. Note that the subprocess homomorphism $h_u^{B_\tau}$ of a branching process B_τ of any strategy τ to the unfolding is the identity function by definition of τ and B_τ respectively. This means, that there is no renaming of nodes between B_τ and the unfolding. Therefore, all configurations of B_τ are also configurations of the unfolding.

Property 3 states that the construction only changes the strategy τ_n for conditions that are causal successors of an event e_2 where $(e_1, e_2) \in REP(B_\tau)$. This preserves

the property of the winning prefix τ, which is used to start the construction, that it is winning in all configurations which do not contain a 2-player cut-off event.

Lemma 3.16 (Continued winning 2-player strategy). *If there exists a winning 2-player prefix τ in a 2-player Petri game G there exists a winning strategy in G.*

Proof by induction over the set of finite configurations of u. We construct a winning strategy starting with a winning 2-player prefix τ by induction over the set of finite configurations of the unfolding.

Induction assumption:

1. B_{τ_n} satisfies the winning properties for all configurations C_i, $i \leq n$,
2. for all $b \in \mathcal{P}_{\tau_n} : [b]_u \preceq C_n \implies (\forall (e_1, e_2) \in REP(B_\tau) : b \in \mathcal{P}_{Fut(B_{\tau_n}, [e_2]_{\tau_n})} \implies \tau_n(b) = \tau_n(I^{[e_2]}_{[e_1]}(b)))$.
3. For all $b \in \mathcal{P}_{B_\tau}$ holds that $\tau_n(b) = \tau(b) \vee \exists (e_1, e_2) \in REP(B_\tau) : e_2 \leq b$.

Induction base case: The construction starts with $\tau_0 = \tau$. The winning properties hold in $C_0 = \emptyset$, because the cut of C_0 is the initial marking and by definition of a winning 2-player prefix the winning properties hold in the initial marking. The second property holds since for all pairs $(e_1, e_2) \in REP(B_\tau)$ the homomorphism $h^{Fut(u,[e_2]_u)}_{Fut(u,[e_1]_u)}$ maps each initial condition to itself if that condition is in $\mathcal{P}_{Fut(u,[e_2]_u)}$ by Lemma 3.5. The third property holds since $\tau_0 = \tau$.

Induction step: Let C_{n+1} be the next configuration of the unfolding regarding $\preceq$.

Case $C_{n+1} \not\subseteq \mathcal{T}_{B_{\tau_n}}$: Here, the configuration C_{n+1} is not a configuration in the so far constructed strategy B_{τ_n}, i.e. there exists at least one event in the configuration that is not allowed by the strategy τ_n. Since the induction considers all finite configurations of the unfolding and we do not know which configurations of the unfolding are configurations in the constructed strategy, this case might occur. However, the strategy does not need to be altered. We define $\tau_{n+1} = \tau_n$ and the induction properties hold by induction assumption.

Case $C_{n+1} \subseteq \mathcal{T}_{B_{\tau_n}} \wedge \neg \exists e_2 \in C_{n+1} : \exists e_1 \in \mathcal{T}_\tau : (e_1, e_2) \in REP(B_\tau)$: Here, the configuration C_{n+1} is a configuration of the winning prefix that does not contain any cut-off event of the repetitions $REP(B_\tau)$. Therefore, the strategy remains the same, $\tau_{n+1} = \tau_n$. Induction property 1 holds by definition of a winning 2-player prefix and induction property 3 holds by induction assumption. For all conditions b with $[b]_u = C_{n+1}$, induction property 2 follows from Lemma 3.5. For all other conditions, induction property 2 holds by induction assumption.

Case $C_{n+1} \subseteq \mathcal{T}_{B_{\tau_n}} \wedge \exists e_2 \in C_{n+1} : \exists e_1 \in \mathcal{T}_\tau : (e_1, e_2) \in REP(B_\tau)$: Here, the configuration C_{n+1} is a configuration of B_{τ_n} and contains a cut-off event e_2 of a repetition in $REP(B_\tau)$. Let $(e_1, e_2) \in REP(B_\tau)$ be such a pair. For all $b \in Cut(C_{n+1})$, the strategy τ_{n+1} is defined as $\tau_{n+1}(b) = I^{[e_2]}_{[e_1]}(b)$ and $\tau_{n+1}(b) = \tau_n(b)$ otherwise.

The second induction property holds for the pair (e_1, e_2) for all $b \in Cut(C_{n+1})$ by definition of τ_{n+1} and by induction assumption for all other conditions.

From Lemma 3.5 it follows, that the third induction property is preserved for all conditions in $Cut(C_{n+1})$ that are not causally preceded by e_2. Thus, induction property 3 holds.

For the pair (e_1, e_2), it remains to show that the winning properties are satisfied in $Cut(C_{n+1})$: The last known $lkc(e_1)$ is a cut in B_{τ_n} by induction property 3 (and by the definition of $REP(B_\tau)$). Since induction property 2 and $e_1 \prec e_2$ hold for the pair (e_1, e_2), it follows that $I^{[e_2]}_{[e_1]}(Cut(C_{n+1})))$ is a cut in B_{τ_n}. By induction property 1, that cut satisfies the winning properties. By induction property 2 the strategy is equal in that cut and $Cut(C_{n+1})$. Hence, C_{n+1} satisfies the winning properties in $B_{\tau_{n+1}}$.

For all other pairs $(e'_1, e'_2) \in REP(B_\tau)$ such that there exists a condition b in $Cut(C_{n+1})$ that is also a condition in $\mathcal{P}_{Fut(u,[e'_2]_u)}$, it remains to show that the second induction property holds. By definition of $REP(B_\tau)$, the events e_2 and e'_2 are not causally related (or they are equal). The two cases $e_2||e'_2$ and $e_2\#e'_2$ are distinguished:

Subcase $e_2||e'_2$ and $prc(B_\tau, e'_1, e'_2)$: By Lemma 3.8, it holds for all conditions $b \in C_{n+1} \cap \mathcal{P}_{Fut(u,[e'_2]_u)}$ that $I^{[e_2]}_{[e_1]}(I^{[e'_2]}_{[e'_1]}(b)) = I^{[e'_2]}_{[e'_1]}(I^{[e_2]}_{[e_1]}(b))$. Since $I^{[e_2]}_{[e_1]}(b)$ is an isomorphism and by Lemma 3.7 follows that ${I^{[e_2]}_{[e_1]}}^{-1}(I^{[e'_2]}_{[e'_1]}(I^{[e_2]}_{[e_1]}(b))) = I^{[e'_2]}_{[e'_1]}(b)$. To show that $\tau_{n+1}(b) = \tau_{n+1}(I^{[e'_2]}_{[e'_1]}(b))$ holds, it is shown that for the left side of the equation $\tau_{n+1}(b) = \tau_{n+1}({I^{[e_2]}_{[e_1]}}^{-1}(I^{[e'_2]}_{[e'_1]}(I^{[e_2]}_{[e_1]}(b))))$ holds. As shown before, $\tau_{n+1}(b) = \tau_{n+1}(I^{[e_2]}_{[e_1]}(b))$ holds. By Lemma 3.7, $I^{[e'_2]}_{[e'_1]}$ can be applied on $I^{[e_2]}_{[e_1]}(b)$. The equality still holds after applying $I^{[e'_2]}_{[e'_1]}$ by induction assumption. Afterwards, the inverse ${I^{[e_2]}_{[e_1]}}^{-1}$ can be applied by Lemma 3.7 since $e_1||e'_2$ holds by Lemma 3.6. Then, applying the inverse of $I^{[e_2]}_{[e_1]}$ also preserves the equality by induction assumption since the local configuration of $I^{[e'_2]}_{[e'_1]}(I^{[e_2]}_{[e_1]}(b))$ is less or equal C_{n+1} w.r.t. $\preceq$.

Subcase $e_2\#e'_2$ and $prc(e'_1, e'_2)$: Since e_2 and e'_2 are in conflict for all $b \in C_{n+1} \cap \mathcal{P}_{Fut(u,[e'_2]_u)}$ holds that $b||e_2$ and $b||e'_2$. This is a contradiction since there exists no set of three pairwise concurrent nodes since there are only two tokens. □

In the following, a nondeterministic algorithm yielding a winning 2-player prefix is presented. The nondeterminism of this algorithm solely concerns the choices of the system players. The algorithm resembles the algorithm that constructs a complete finite prefix regarding reachability of a Petri net presented in [ERV02]. The main difference is the cut-off criterion. The cut-off criterion for 2-player games uses partial repetition cuts instead of just the equivalence of last known markings.

Note that an event is added to the prefix if and only if the guessed strategy allows that event to be fired. An event is allowed to be fired if its label is in the strategy of all its system preconditions. (For an environment precondition is assumed that all events are allowed.)

The presented algorithm that constructs a winning prefix adds one event after another to the prefix until there are no more events to be added. This is the case after cut-off events and if the guessed strategy does not allow an event to be fired.

We formally define the set of events that could be added next. Informally speaking, a set of events that could be added next to a branching process is defined. This is the set of *possible extensions* as defined as follows.

Definition 3.17 (Possible extensions). For a branching process B of a net N_0, the set of *possible extensions* $PE(B)$ is the set of events $e \in \mathcal{T}_u$ such that

1. $pre_u(e) \subseteq \mathcal{P}$, and
2. $e \notin \mathcal{T}$.

$\triangleleft$

Note that this definition of the possible extensions is only sensible assuming that the subprocess homomorphism from B to u is the identity function (no renaming of nodes).

Now, constructing a winning 2-player prefix is presented in Algorithm 1. The instruction *guess* $\tau(b)$ assigns a random subset of $h_u(post_u(b))$ to $\tau(b)$ unless b is an environment condition. Then, the whole set $h_u(post_u(b))$ is assigned to $\tau(b)$. By convention, for any set of places $X \subseteq \mathcal{P}_u$, the set X^E denotes $X \cap \mathcal{P}_u^E$ and X^S denotes $X \cap \mathcal{P}_u^S$.

In the next step, the maximal size of the set $NCO(B_\tau)$ of a winning two-player prefix τ is determined. Checking if any strategy τ is a winning 2-player prefix can be done by checking if τ is winning in all configurations that are a subset of $NCO(B_\tau)$.

Lemma 3.18 (Size of winning 2-player prefix). *For a winning two-player prefix τ constructed by Algorithm 1 the set of non-cut-off events $NCO(B_\tau)$ contains at most $2|\mathcal{P}_0| \cdot (|\mathcal{T}_0| + 1)$-many events.*

Proof. If a token takes $|\mathcal{P}_0|$ events without a joint event a partial repetition occurs because a label of a condition is repeated and the condition of the other player in its last known cut remained the same. So, after at most $2|\mathcal{P}_0|$-many events (there are only two tokens) either a joint event occurs or the conditions of a winning two-player prefix are met.

After at most $(|\mathcal{T}_0| + 1)$-many joint events there exists two joint events with the same label. The cuts of those events are partial repetition cuts since the labels of their presets are equal and there are no other tokens in their last known cuts.

Overall, the resulting size of $NCO(B_\tau)$ is at most $2|\mathcal{P}_0| \cdot (|\mathcal{T}_0| + 1)$. □

This section concludes with the Theorem about the synthesis problem of 2-player games being an NP-complete problem.

Theorem 3.19 (2-player Petri games). *The synthesis problem for 2-player Petri games is NP-complete.*

Proof. By Lemma 3.16, a winning strategy exists if a winning 2-player prefix exists.

A winning strategy is also a winning prefix by definition of a winning 2-player prefix.

By Lemma 3.18, the number of non-cut-off events in B_τ is quadratically bounded in the size of the underlying Petri net. There are at most quarticly many configurations containing only non-cut-off events. Checking if a configuration is winning takes linear time. Overall, the time complexity is at worst quintic in the size of the underlying Petri net.

By Lemma 3.10, 2-player Petri games are NP-hard and NP-completeness follows. □

Algorithm 1: NP-algorithm constructing a winning 2-player prefix

Input: Token-partitioning 2-player Petri game G_0
Output: Strategy τ, and "*yes*" or "*no*" stating if τ is a winning 2-player prefix

forall $b \in \mathcal{P}_u$ **do**
  $\tau(b) := \emptyset$
forall $b \in In_u$ **do**
  `guess` $(\tau(b))$
$B_{prefix} := (In_u, \emptyset, \emptyset, \emptyset, In_u)$;
$PE := PE(B_{prefix})$, $cutoff := \emptyset$, $forbidden := \emptyset$;
while $PE \neq \emptyset$ **do**
  choose event $e \in PE$ which is mimimal with respect to $\preceq$;
  if $[e]_u \cap cutoff = \emptyset \wedge \forall b \in pre(e) : h_u(e) \in \tau(b)$ **then**
    `addEvent` (B_{prefix}, e);
    `/* Guessing strategy for added system conditions. */`
    forall $b \in post_u(e)$ **do**
      `guess` $(\tau(b))$
    if *e is 2-player cut-off event* **then**
      $cutoff := cutoff \cup \{e\}$;
    `/* Generating possible extensions for new` B_{prefix}`. */`
    $PE := PE(B_{prefix}) \setminus forbidden$;
  else
    $PE := PE \setminus \{e\}$;
    $forbidden := forbidden \cup \{e\}$;
`/* Check if guessed prefix is a winning 2-player prefix. */`
output := τ, "*yes*" ;
forall *For all configurations* $C \subseteq \mathcal{T}_\tau \setminus cutoff$ **do**
  if *A winning condition does not hold in* C **then**
    output := τ, "*no*";
procedure `addEvent`(B_{prefix}, e)**:**
  $\mathcal{T}_{prefix} := \mathcal{T}_{prefix} \cup \{e\}$;
  $\mathcal{P}_{prefix} := \mathcal{P}_{prefix} \cup post(e)$;
  forall $b \in post_u(e)$ **do**
    $post_{pre} := post_{pre} \cup (e, b)$;
  forall $b \in pre_u(e)$ **do**
    $pre_{pre} := pre_{pre} \cup (e, b)$;
  return;

3.3 4-player Petri Games

In this section, we show that 4-player Petri games are decidable within nondeterministic exponential time (NEXP). For this, another kind of cuts are introduced where a winning strategy can be repeated. Those cuts are called *synchronisation equivalent cuts*. As for partial repetition cuts, synchronisation equivalent cuts come in pairs. The idea of those cuts is to identify game states that are similar to a simultaneous synchronisation of all 4 players. However, not all 4 players have to synchronise at *one* joint event. Instead of one joint event, a subprocess, called *synchronisation segment*, of the future of an event is considered. We observe for joint events where all 4 players participate that a winning strategy can be repeated if the last known markings of two such events coincide. This case is included by partial repetition cuts. However, a generalisation of partial repetition cuts is necessary.

Definition 3.20 (Synchronisation segment). The *synchronisation segment* $Sg(B, e)$ of an event $e \in \mathcal{T}$ of a branching process B is defined as the maximal branching process (w.r.t. the subprocess relation $\sqsubseteq$) such that

1. $Sg(B, e) \sqsubseteq Fut(B, [e])$ and
2. $\forall e' \in \mathcal{T}_{Sg(B,e)} : ((e'||e) \wedge (\neg\exists e'' \in \mathcal{T}_{Sg(B,e)} : [e''] \prec [e'] \wedge prc(B, e'', e')))$.

$\lhd$

Condition 1 of the definition of a synchronisation segment $Sg(B, e)$ states that a synchronisation segment is a subprocess of the future of the event e in the branching process B. Condition 2 states that all events in the synchronisation segment $Sg(B, e)$ are concurrent to e and are not preceded by a partial repetition. Thus, a synchronisation segment contains all concurrent events that do not satisfy the 2-player cut-off criterion, the partial repetition. Later on, we define the 4-player cut-off criterion as a generalization of the 2-player cut-off criterion, i.e. every partial repetition also satisfies the 4-player cut-off criterion.

Example 3.21 (Synchronisation segment). An example of a synchronisation segment is shown in Fig. 3.8. The branching process inside the dashed blue line shows the nodes of the synchronisation segment $Sg(\sigma_{G_0}, t^1)$. (σ_{G_0} is shown in Fig. 3.2). The initial marking is $In_{Sg(\sigma_{G_0},t^1)} = \{W_1^1, S_{13}^2, S_{23}^2, W_2^1\}$. This segment does not contain any event. In particular, it does not contain o_1^2 and d_2^2 since $t^1 \leq o_1^2$ and $t^1 \leq d_2^2$, i.e. those events are not concurrent to t^1. $\lhd$

We continue with another example of a synchronisation segment.

Example 3.22 (Synchronisation segment). In Fig. 3.9, another synchronisation segment is shown. Here, the synchronisation segment $Sg(B_{Sg}, r_1^1)$ contains all concurrent events until the partial repetition $prc(B_{Sg}, r_2^1, r_2^2)$ occurs. $\lhd$

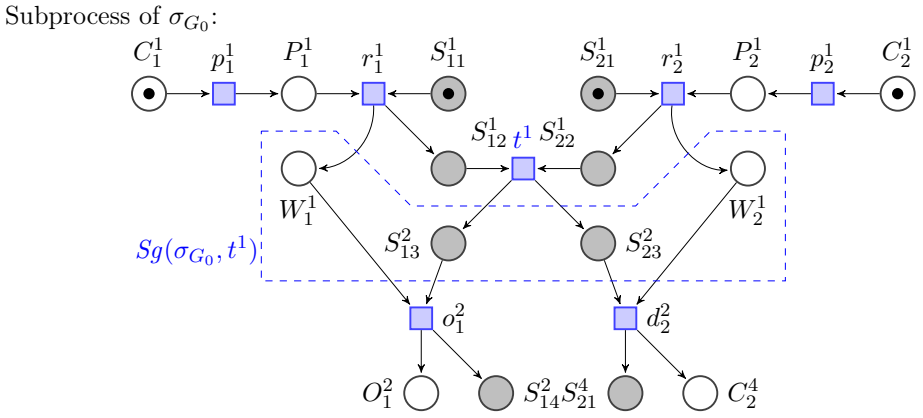


Figure 3.8: A subprocess of σ_{G_0} of Fig. 3.2 is shown. The branching process inside the dashed blue line is the synchronisation segment of t^1.

The next definition describes when two synchronisation segments $Sg(B_\tau, e_1)$ and $Sg(B_\tau, e_2)$, $e_1, e_2 \in \mathcal{T}_\tau$, are equivalent with respect to a strategy τ. This means the synchronisation segments are isomorphic and the isomorphism on the segments preserves the strategy for all conditions which are concurrent to e_2.

Definition 3.23 (Segment equivalence). Let τ be a strategy of a Petri game G. Two events $e_1, e_2 \in \mathcal{T}_\tau$ are called *segment equivalent* if

1. $Sg(B_\tau, e_1) \cong Sg(B_\tau, e_2)$, and
2. $\forall b \in \mathcal{P}_{Sg(B_\tau, e_2)} \setminus post_\tau(e_2) : \tau(b) = \tau(I^{Sg(B_\tau, e_2)}_{Sg(B_\tau, e_1)}(b))$.

◁

Note that segment equivalence is transitive since a composition of isomorphism yields an isomorphism which still preserves the strategy τ. This means that segment equivalence is an equivalence relation on the set of events which justifies its name. (It is also reflexive (identity isomorphism) and symmetric (reverse isomorphism).)

In the next step we define synchronisation equivalent cuts. As for partial repetition cuts synchronisation equivalent cuts are last known cuts of events. Synchronisation equivalent cuts are a generalisation of partial repetition cuts, i.e. if two cuts are partial repetition cuts they are also synchronisation equivalent cuts. However, the last known cuts of two events e_1 and e_2 can also be synchronisation equivalent cuts if three other conditions are satisfied. (The events e_1 and e_2 are events of a branching process B_τ of a strategy τ.)

Condition 1 requires that both events have an equal number of preconditions which is at least 2. The size of synchronisation segment of an event with a single precondition might not be finite in a 4-player game because it is not guaranteed to reach partial

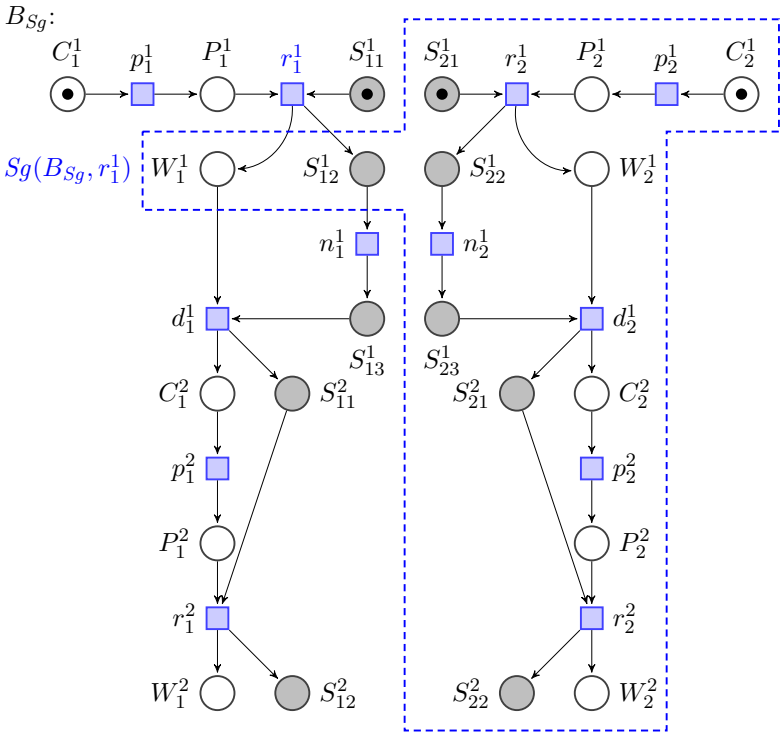


Figure 3.9: A branching process B_{Sg} of the Petri game G_0 of Fig. 3.1 is shown. The branching process inside the dashed blue line is the synchronisation segment $Sg(B_{Sg}, r_1^1)$.

repetition cuts if there are 3 players which communicate arbitrarily. Therefore, synchronisation equivalent cuts do not rely on the synchronisation segments of events with a single precondition since it is not guaranteed that there eventually is another event which is segment equivalent. A strategy could also be repeated if those synchronisation segments are arbitrarily large (or even infinite). However, those cases do not need to be considered because it is still guaranteed to get to synchronisation equivalent cuts in a 4-player game which we show later on in Lemma 3.43.

Condition 2 demands that the two events e_1 and e_2 are segment equivalent.

Condition 3 states that for all events e in the segment of e_2 where $|pre(e)| \geq 2$ it holds that e and $I^{Sg(B_\tau,e_2)}_{Sg(B_\tau,e_1)}(e)$ are segment equivalent, i.e. the isomorphism $I^{Sg(B_\tau,e_2)}_{Sg(B_\tau,e_1)}$ preserves the segment equivalence class of events e in its synchronisation segment. This is the most complex condition of the three conditions. Later on, in Example 3.25, it is illustrated why this condition is necessary.

Definition 3.24 (Synchronisation equivalent cuts). Let B_τ be a branching process of a strategy τ and $e_1, e_2 \in \mathcal{T}_\tau$. The events e_1 and e_2 are called *synchronisation equivalent events* denoted as $sqc(B_\tau, e_1, e_2)$ if

– $prc(B_\tau, e_1, e_2)$,

or

1. $|pre(e_2)| = |pre(e_1)| \geq 2$, and
2. e_1 and e_2 are segment equivalent, and
3. $\forall e \in \mathcal{T}_{Sg(B,e_2)} : (|pre(e)| \geq 2 \implies (e \text{ and } I^{Sg(B_\tau,e_2)}_{Sg(B_\tau,e_1)}(e) \text{ are segment equivalent.}))$.

The last known cuts $lkc(e_1)$, $lkc(e_2)$ of synchronisation equivalent events are called synchronisation equivalent cuts, the local configurations $[e_1]$, $[e_2]$ are called synchronisation equivalent configurations and e_1 and e_2 are called synchronisation equivalent events. $\lhd$

Condition 3 of Def 3.24 states that the segment equivalence class of an event $e \in \mathcal{T}_{Sg(B,e_2)}$ with at least two preconditions must be preserved by the isomorphism $I^{Sg(B_\tau,e_2)}_{Sg(B_\tau,e_1)}$ between the synchronisation segments of e_2 and e_1.

Looking at Def. 3.24: from Lemma 3.5 it follows that $prc(B_\tau, e_1, e_2)$ implies the conditions 2 and 3 since the subprocess homomorphism $h^{Fut(B_\tau,[e_2])}_{Fut(B_\tau,[e_1])}$ is the identity function on the events of $Sg(B, e_2)$. So, the condition $prc(B_\tau, e_1, e_2)$ aims at events with a single precondition, i.e. a token that takes no joint event at all will also get to synchronisation equivalent cuts. In all other cases, the conditions 1-3 also hold if $prc(B_\tau, e_1, e_2)$ holds. The conditions are split in that way depending on their number of preconditions since a synchronisation segment of an event with a single precondition is not guaranteed to be bounded or even finite in a 4-player Petri game which is unsuitable for a cut-off criterion since it is not guaranteed to be satisfied eventually.

In the following, we present an example which shows that conditions 1 and 2 of Def. 3.24 are not sufficient and there is a need for condition 3. We reason that condition 3 is a suitable condition. The example shows that repeating a winning strategy after cuts that only satisfy the conditions 1 and 2 of Def. 3.24 does not yield a winning strategy.

Example 3.25 (Synchronisation equivalent cuts). In Fig. 3.10, a token-partitioning 4-player Petri game G_3 and four branching processes of G_3 are shown, where each branching process represents exactly one transition sequence (with maximally many transitions) in G_3. The system players win the Petri game G_3 if the only bad marking $\{B_1, B_2\}$ is avoided, i.e. at least one system player has to choose A_1 or A_2, respectively.

In the initial marking four transitions are enabled: u_1, u_2, v_1 and v_2. In any transition sequence (with maximally many transitions) of G_3 either u_1 or u_2 is fired and either v_1 or v_2 is fired. This can be seen in the four branching processes $B^{u_i v_j}$, $i, j = 1, 2$. For example, the branching process $B^{u_1 v_1}$ (top left) contains an event with label u_1 and an event with label v_1. (The label of a node is obtained by removing the superscript.)

In G_3, after firing a transition s_1 or s_2 the system player on place S_1 has to decide between A_1 and B_1. The transition s_1 can be fired if a token is on the place V_2 after firing v_2 and the transition s_2 can be fired if a token is on the place V_1 after firing v_1. The lower part is symmetrical where the system player has to decide between A_2 and B_2 after reaching S_2 via s_3 (and u_1 beforehand) or via s_4 (and u_2 beforehand).

Obviously, there exist several winning strategies for G_3. The most obvious winning strategy is to always choose A_1 and A_2. However, to show that the conditions 1 and 2 of Def. 3.24 are not a sufficient criterion to repeat a winning strategy another winning strategy is considered.

The winning strategy considered here is represented by the four branching processes $B^{u_i v_j}$, $i, j = 1, 2$. Assembling these four branching processes by forming the set union of their nodes and their preset and postset relation yields a winning strategy: for each of the four combination $\{u_1, u_2\} \times \{v_1, v_2\}$ the branching process $B^{u_i v_j}$ describes the winning strategy. For example, if u_1^1 and v_1^1 are fired the system player on $S_1^{u_1 v_1}$ chooses B_1 while the system player on $S_2^{u_1 v_1}$ chooses A_2 avoiding the bad marking. This is shown in $B^{u_1 v_1}$ (top left). Note that the branching process overlap for the events with labels u and v, e.g. u_1^1 (top left) and u_1^1 (top right) are the same event.

In B^{u_1, v_2} (top right), after u_1^1 and v_2^1 are fired the system player on $S_2^{u_1 v_2}$ chooses B_2 while the system player on $S_1^{u_1 v_2}$ chooses A_1 also avoiding the bad marking.

In B^{u_2, v_1} (bottom left), after u_2^1 and v_1^1 are fired the system player on $S_1^{u_2 v_1}$ chooses B_1 while the system player on $S_2^{u_2 v_1}$ chooses A_2 also avoiding the bad marking.

Finally in B^{u_2, v_2} (bottom right), after u_2^1 and v_2^1 are fired both system players decide for option A avoiding the bad marking.

Now, we alter the strategy by repeating it after every pair of last known cuts that satisfy 1 and 2 of Def. 3.24. For $B^{u_1 v_1}$ (top left) and $B^{u_1 v_2}$ (top right) is nothing to repeat since neither s_2^1 and s_1^1 are synchronisation equivalent nor s_3^1 and s_3^2 are segment equivalent.

For $B^{u_1 v_1}$ (top left) and $B^{u_2 v_1}$ (bottom left), the events s_3^1 (top left) and s_4^1 (bottom left) are segment equivalent since the last known markings are the same, especially the place V_1, and the system players on $S^{u_1 v_1}$ and on S^{u_2, v_1} decide for B_1. However, copying the strategy after $S_2^{u_2 v_1}$ from $S_2^{u_1 v_1}$ does not alter the strategy since the system player on both places already go for option A_2. Considering $B^{u_1 v_2}$ (top right) together with $B^{u_1 v_1}$ (top left) and $B^{u_2 v_1}$ (bottom left) there are no more segment equivalent cuts.

The problem that arises without condition 3 in Def. 3.24 is showcased when repeating the strategy for a condition in the branching process $B^{u_2 v_2}$ (bottom right) if only 1 and 2 are satisfied and 3 is not. The events s_1^2 ($B^{u_2 v_2}$, bottom right) and s_2^2 ($B^{u_2 v_1}$, bottom left) are segment equivalent. In particular, the last known markings coincide in the places U_2 and E_2. Therefore, we copy the strategy after $S_1^{u_2 v_2}$ from $S_1^{u_2 v_1}$ which leads to $S_1^{u_2 v_2}$ choosing B_1. (Note that the strategy is copied although s_1^2 and s_2^2 are in conflict.) Condition 3 requires additionally that the events s_4^2 (bottom right) and s_4^1 (bottom left) are segment equivalent which is not the case since their last known markings differ in V_2 ($V_2^1 \in \mathit{lkc}(s_4^2)$) and V_1 ($V_1^1 \in \mathit{lkc}(s_4^1)$).

The events s_4^2 ($B^{u_2 v_2}$, bottom right) and s_3^2 ($B^{u_1 v_2}$, top right) are segment equivalent. In particular, the last known markings coincide in the places V_2 and E_1. Therefore, we copy the strategy after $S_2^{u_2 v_2}$ from $S_2^{u_1 v_2}$ which leads to $S_2^{u_2 v_2}$ choosing B_2. Now, the bad marking $\{B_1, B_2\}$ is reachable in $B^{u_2 v_2}$. Condition 3 is not satisfied for s_4^2 (bottom right) and s_3^2 (top right) because s_1^2 (bottom right) and s_1^1 (top right) are not segment equivalent since the conditions $U_2^1 \in \mathit{lkc}(s_1^2)$ and $U_1^1 \in \mathit{lkc}(s_1^1)$ have different labels. $\lhd$

Example 3.25 shows that another condition besides conditions 1 and 2 in Def. 3.24 is necessary as a criterion to repeat a winning strategy. We found condition 3 that for all $e \in \mathcal{T}_{Sg(B, e_2)} : |\mathit{pre}(e)| \geq 2 \implies (e \text{ and } I^{Sg(B_\tau, e_2)}_{Sg(B_\tau, e_1)}(e))$ are segment equivalent to be a suitable condition. In Example 3.25, condition 3 does not hold when copying the

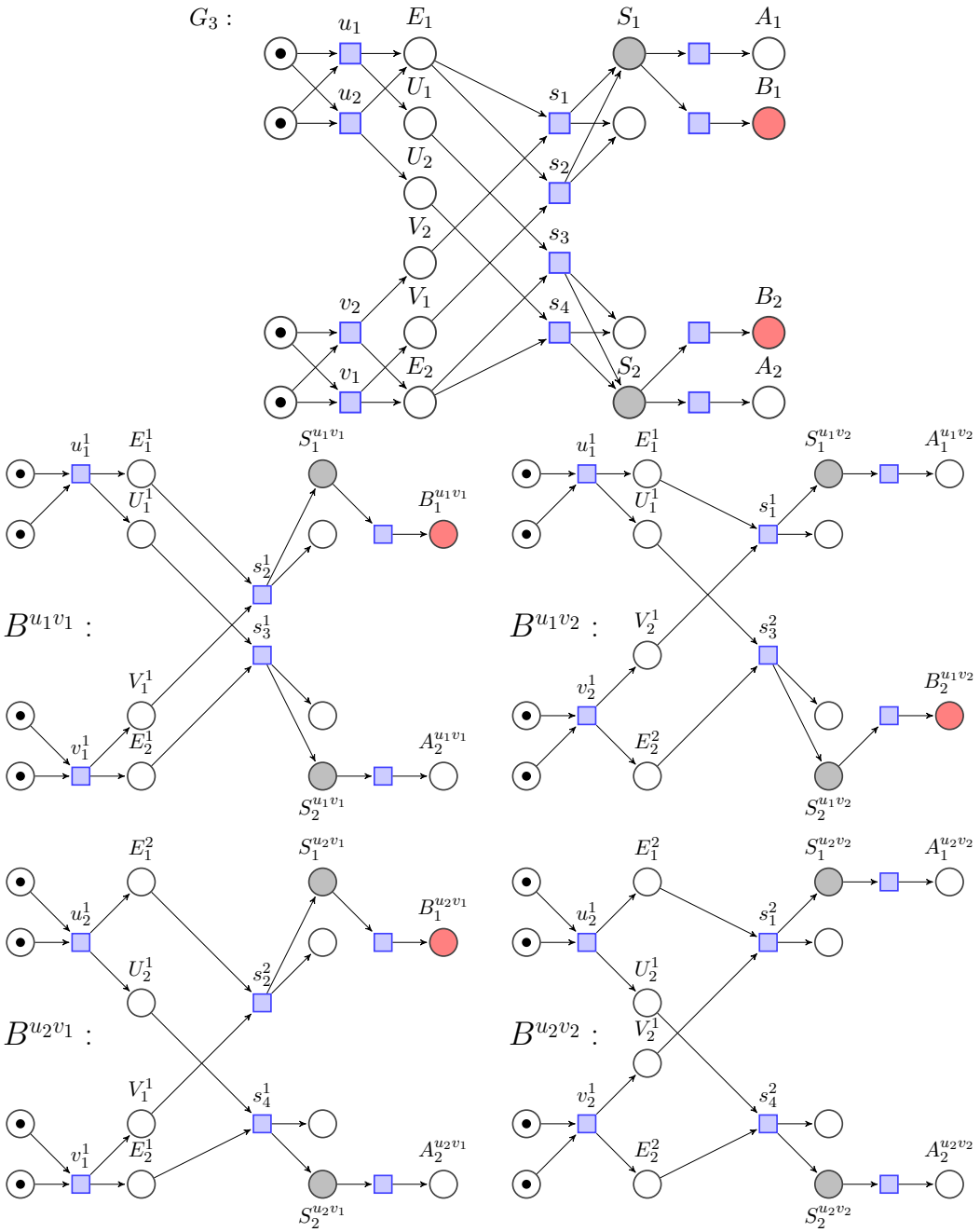


Figure 3.10: A Petri game G_3 and four branching processes $B^{u_iv_j}$, where $i, j = 1, 2$. The bad marking is $\{B_1, B_2\}$.

strategy which leads to the case that the futures of two concurrent events (s_1^2 and s_4^2) are copied from two different futures: the future of s_1^2 ($B^{u_2v_2}$, bottom right) is copied from

the future of s_2^2 ($B^{u_2v_1}$, bottom left) and the future of s_4^2 ($B^{u_2v_2}$, bottom right) is copied from the future of s_3^2 ($B^{u_1v_2}$, top right). Condition 3 does not hold for s_1^2 (bottom right) and s_2^2 (bottom left) in this example because of the conditions V_2^1 (bottom right) and V_1^1 (bottom left): condition 3 requires the events s_4^2 (bottom right) and s_4^1 (bottom left) to be segment equivalent, too. However, the condition V_2^1 (bottom right) is in the last known cut of s_4^2 while condition V_1^1 (bottom left) is in the last known cut of s_4^1. Since the labels of V_1^1 and V_2^1 differ the events s_4^2 and s_4^1 are not segment equivalent.

Condition 3 does also not hold for the second copying step where the future of s_4^2 (bottom right) is copied from the future of s_3^2 (top right). Here, the conditions U_2^1 (bottom right) and U_1^1 (top right) prevent that the events s_1^2 (bottom right) and s_1^1 (top right) are segment equivalent as required by condition 3.

With condition 3, for all such events e which are in the synchronisation segment of e_2 it is guaranteed that their future is repeatable from $I^{Sg(B_\tau,e_2)}_{Sg(B_\tau,e_1)}(e)$ which is an event in the future of e_1, i.e. it is guaranteed to copy from the same future, namely the intersection of the futures of e_1 and $I^{Sg(B_\tau,e_2)}_{Sg(B_\tau,e_1)}(e)$, $Fut(B_\tau, Cut([e_1] \cup [I^{Sg(B_\tau,e_2)}_{Sg(B_\tau,e_1)}(e)]))$.

The condition 3 ensures that such an event e and its image under the isomorphism $I^{Sg(B_\tau,e_2)}_{Sg(B_\tau,e_1)}$ are segment equivalent. We keep the idea that a strategy is repeatable after segment equivalent events. However, the additional condition prevents copying from different cuts as in Example 3.25. Informally speaking, we do not copy a strategy after every segment equivalent event. Instead, a strategy is only copied if it is guaranteed to copy from a single cut. For e_2 and $e \in \mathcal{T}_{Sg(B,e_2)}$ as in the previous passage, the future of the cut $Cut([e_2] \cup [e])$ is copied from the future of the single cut $Cut([e_1] \cup [I^{Sg(B_\tau,e_2)}_{Sg(B_\tau,e_1)}(e)])$.

In the following subsection, some key properties of synchronisation equivalent cuts are shown.

3.3.1 Properties of Synchronisation Equivalent Cuts

In this subsection we present properties of synchronisation equivalent cuts (Def. 3.24). For 4-player games, we show that if e and e_2 are concurrent and in the synchronisation segment of each other, and $|pre(e)| = 2 = |pre(e_2)|$, then $sqc(B, e_1, e_2)$ holds if and only if $sqc(B, e, I^{Sg(B,e_2)}_{Sg(B,e_1)}(e)))$ holds. This way, we guarantee to copy the future of a single cut constructing a winning strategy instead of copying from two different cuts, one for e_2 and one for e as shown in Example 3.25. In this subsection, the branching process B is a branching process of a strategy τ where we omit the index τ.

The first Lemma 3.26 towards the previously described property states the following. Given three events e, e_2, e_2' of a branching process B such that $e||e_2$, $e||e_2'$, and the same set of tokens participate in e_2 and e_2', and every token of the net participates in e_2 or e it holds that the last known cuts of e_2 and e_2' are equal except for their postsets. This is the property of a pair of partial repetition events $prc(B, e_1, e_2)$ that $lkc(e_1) \setminus post(e_1) = lkc(e_2) \setminus post(e_2)$. Since the synchronisation segment of any event is bounded by partial repetitions this lemma shows how the partial repetitions bound the synchronisation segments in the case when all players – besides those participating in e – take a joint transition in the synchronisation segment of e.

Beyond that, the following Lemma 3.26 aims at the case in a 4-player game where there are two events e_2, e_2' in the synchronisation segment of an event e, such that $|pre(e_2)| = |pre(e_2')| = |pre(e)| = 2$, i.e. all players participate either in e or in e_2 (e_2' respectively). If there is an event e_1 such that $sqc(B, e_1, e_2)$ holds we want to show that $sqc(B, e, I^{Sg(B,e_2)}_{Sg(B,e_1)}(e)))$ also holds: Especially for this, we need to show that condition 3 of Def. 3.24 holds. It states that the segment equivalence of events (where there are at least 2 preconditions) in a synchronisation segment must be preserved by $I^{Sg(B,e)}_{Sg(B,I^{Sg(B,e_2)}_{Sg(B,e_1)}(e))}$. Here, those events are e_2 and e_2'. For e_2, it suffices to show that $I^{Sg(B,e)}_{Sg(B,I^{Sg(B,e_2)}_{Sg(B,e_1)}(e))}(e_2) = e_1$, since e_2 and e_1 are segment equivalent by definition of $sqc(B, e_1, e_2)$. To show that the segment equivalence class of e_2' is also preserved by $I^{Sg(B,e)}_{Sg(B,I^{Sg(B,e_2)}_{Sg(B,e_1)}}$, we use that the segment equivalence of e_2 is preserved by this isomorphism.

With the help of the next Lemma 3.26, we show that the events in the synchronisation segments of e_2 and e_2' are equal, i.e. the synchronisation segments only differ in the postsets of e_2 and e_2'. Later on, this is used to show that the synchronisation segment of e_2' is also preserved by $I^{Sg(B,e)}_{Sg(B,I^{Sg(B,e_2)}_{Sg(B,e_1)})(e)}$.

Overall, the reasoning here starts with the segment equivalence of e_2 and e_1 by definition of $sqc(B, e_1, e_2)$. Since e_2 is an event in the synchronisation segment of e we need to show that e_2 and $I^{Sg(B,e)}_{Sg(B,I^{Sg(B,e_2)}_{Sg(B,e_1)}(e))}(e_2)$ are segment equivalent to satisfy condition 3 of Def. 3.24. This is done by showing that $I^{Sg(B,e)}_{Sg(B,I^{Sg(B,e_2)}_{Sg(B,e_1)}(e))}(e_2) = e_1$. For other events in the synchronisation segment of e, here e_2', the synchronisation equivalence of e_2' and $I^{Sg(B,e)}_{Sg(B,I^{Sg(B,e_2)}_{Sg(B,e_1)}(e))}(e_2')$ is shown via the synchronisation segment of e_2: we show that the set of events of the synchronisation segments of e_2 and e_2' coincide. We also show that the set of events of the synchronisation segments of $I^{Sg(B,e)}_{Sg(B,I^{Sg(B,e_2)}_{Sg(B,e_1)}(e))}(e_2)$ and $I^{Sg(B,e)}_{Sg(B,I^{Sg(B,e_2)}_{Sg(B,e_1)}(e))}(e_2')$ coincide. Then we use the segment equivalence of e_2 and e_1 to show that e_2' and $I^{Sg(B,e)}_{Sg(B,I^{Sg(B,e_2)}_{Sg(B,e_1)}(e_2'))}$ are also segment equivalent. Then, condition 3 of $sqc(B, e, I^{Sg(B,e_2)}_{Sg(B,e_1)}(e)))$ holds.

In Fig. 3.11, the following Lemma 3.26 is illustrated. The events e_2 and e_2' are concurrent to e and every token of the net either participates in e or e_2 (or e_2' respectively). Then, $lkc(e_2) \setminus post(e_2) = lkc(e_2') \setminus post(e_2')$ holds.

Lemma 3.26 (Last known cut of concurrent events). *Let B be a token-partitioning branching process with partition P and $e, e_2', e_2 \in \mathcal{T}$. If $e||e_2'$, $e||e_2$ and $Ind(e_2') = Ind(e_2) = I_P \setminus Ind(e)$ it holds that $lkc(e_2') \setminus post(e_2') = lkc(e_2) \setminus post(e_2)$.*

Proof. We show for all $i \in Ind(e) : p_i(lkc(e_2')) = p_i(lkc(e_2))$. By Lemma 2.20, $[e_2'] = \bigcup_{\{b \in lkc(e) | b \leq e_2'\}} [b] \cup \{e' \in \mathcal{T}_{Fut(B,lkc(e))} \mid e' \leq e_2'\}$. If there exists $f \in \{e' \in \mathcal{T}_{Fut(B,lkc(e))} \mid$

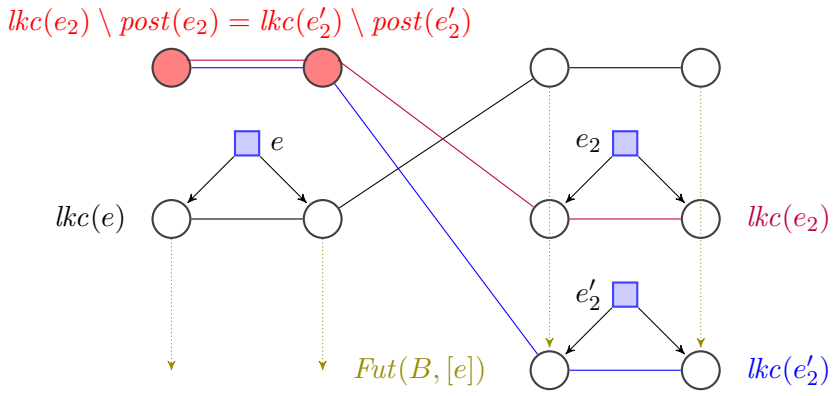


Figure 3.11: Graphical depiction of Lemma 3.26. The events e_2 and e'_2 are concurrent and in the future of e and satisfy that every token of the net either participates in e or e_2 (e'_2, respectively). Then, it holds that $lkc(e_2) \setminus post(e_2) = lkc(e'_2) \setminus post(e'_2)$.

$e' \leq e'_2\}$ from Lemma 2.31 follows that $e \leq f$. Thus, $e \leq f \leq e'_2$. This is a contradiction to $e||e'_2$. Therefore, it holds that $\{e' \in \mathcal{T}_{Fut(B,lkc(e))} \mid e' \leq e'_2\} = \emptyset$. Analogously it holds for e_2 that $\{e' \in \mathcal{T}_{Fut(B,lkc(e))} \mid e' \leq e_2\} = \emptyset$. Since $Ind(e'_2) = Ind(e_2) = I_P \setminus Ind(e)$, it holds that $\{b \in lkc(e) \mid b \leq e'_2\} = lkc(e) \setminus post(e) = \{b \in lkc(e) \mid b \leq e_2\}$ by Lemma 2.31. Therefore, $[e_2]$ and $[e'_2]$ contain the same set of events where a token from $Ind(e)$ participates. It follows that $lkc(e'_2) \setminus post(e'_2) = lkc(e_2) \setminus post(e_2)$ holds. □

For example in Fig. 3.9, if we set $e = r_1^1$, $e'_2 = r_2^1$ and $e_2 = d_2^1$ it holds that $lkc_{Sg}(r_2^1) \setminus post(r_2^1) = lkc_{Sg}(d_2^1) \setminus post(d_2^1) = \{C_1^1, S_{11}^1\}$.

Next, we show that the equivalence of last known cuts of events e_1 and e_2 (except for their postsets) is preserved by any isomorphism $I_{[f_1]}^{[f_2]}$ if e_1 and e_2 are both in the future of f_2, i.e. the property of partial repetition events $prc(B, e_1, e_2)$ that $lkc(e_2) \setminus post(e_2) = lkc(e_1) \setminus post(e_1)$ is preserved. If the postset of e_2 and e_1 additionally have the same labels, we show that the equivalence of the last known markings is also preserved, i.e. partial repetition events are preserved. Note that an isomorphism $I_{[f_1]}^{[f_2]}$ exists if $lkm(f_2) = lkm(f_1)$.

In Fig. 3.12, the following Lemma 3.27 is illustrated. It states that the equivalence $lkc(e_2) \setminus post(e_2) = lkc(e_1) \setminus post(e_1)$ for two events e_1 and e_2 in the future of an event f_2 is preserved by the isomorphism $I_{[f_1]}^{[f_2]}$.

Lemma 3.27 (Isomorphism on last known cuts). *Let N be a net and $f_1, f_2 \in \mathcal{T}_u$ such that $lkm_u(f_1) = lkm_u(f_2)$ and $e_1, e_2 \in \mathcal{T}_{Fut(u,[f_2]_u)}$. If $lkc(e_1) \setminus post(e_1) = lkc(e_2) \setminus post(e_2)$ holds then $lkc(I_{[f_1]}^{[f_2]}(e_1)) \setminus post(I_{[f_1]}^{[f_2]}(e_1)) = lkc(I_{[f_1]}^{[f_2]}(e_2)) \setminus post(I_{[f_1]}^{[f_2]}(e_2))$ holds.*

Proof. The set equivalence $lkc_u(I_{[f_1]}^{[f_2]}(e_1)) \setminus post_u(I_{[f_1]}^{[f_2]}(e_1)) = lkc_u(I_{[f_1]}^{[f_2]}(e_2)) \setminus post_u(I_{[f_1]}^{[f_2]}(e_2))$ is shown for the conditions in $\mathcal{P}_{Fut(u,[f_2]_u)} \setminus lkc_u(f_2)$ and for the conditions in $(\mathcal{P}_u \setminus$

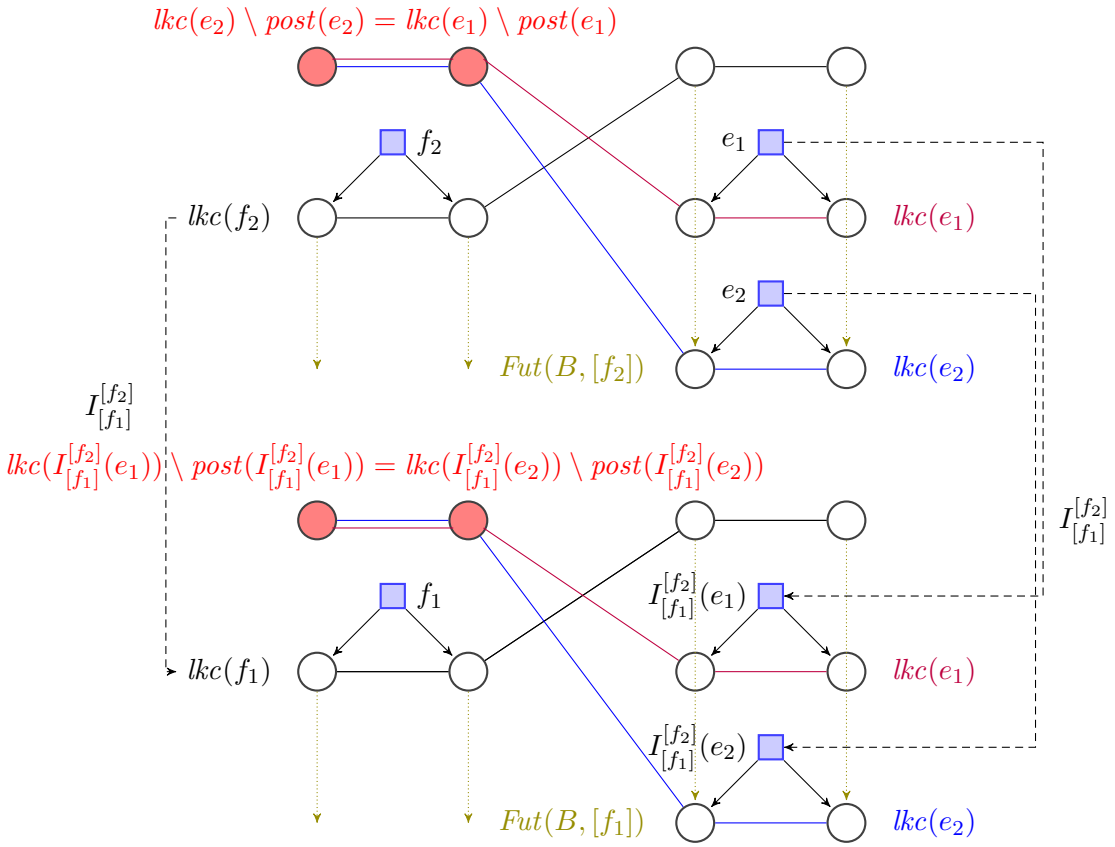


Figure 3.12: Graphical depiction of Lemma 3.27. The events e_2 and e_1 are in the future of f_2 and satisfy the property of partial repetition events that $lkc(e_2) \setminus post(e_2) = lkc(e_1) \setminus post(e_1)$. In the depicted setting, Lemma 3.27 states that $lkc(I^{[f_2]}_{[f_1]}(e_1)) \setminus post(I^{[f_2]}_{[f_1]}(e_1)) = lkc(I^{[f_2]}_{[f_1]}(e_2)) \setminus post(I^{[f_2]}_{[f_1]}(e_2))$ holds.

$\mathcal{P}_{Fut(u,[f_2]_u)}) \cup lkc_u(f_2)$ separately. We start with the former.

$$
\begin{aligned}
&(lkc_u(e_2) \setminus post_u(e_2) \cap (\mathcal{P}_{Fut(u,[f_2]_u)} \setminus lkc_u(f_2)) \\
&\overset{\text{Def. } lkc_u}{=} ((In_u \cup post_u([e_2]_u) \setminus pre_u([e_2]_u)) \setminus post_u(e_2)) \cap (\mathcal{P}_{Fut(u,[f_2]_u)} \setminus lkc_u(f_2)) \\
&\overset{\text{Lemma 2.20}}{=} ((In_u \cup post_u(\bigcup_{\{b \in lkc_u(f_2) \mid b \leq e_2\}} [b]_u \cup \{e \in \mathcal{T}_{Fut(u,[f_2]_u))} \mid e \leq e_2\}) \\
&\setminus pre_u(\bigcup_{\{b \in lkc_u(f_2) \mid b \leq e_2\}} [b]_u \cup \{e \in \mathcal{T}_{Fut(u,[f_2]_u)} \mid e \leq e_2\}) \setminus post_u(e_2) \\
&\cap (\mathcal{P}_{Fut(u,[f_2]_u)} \setminus lkc_u(f_2))) \\
&\overset{\text{Remark 1}}{=} (post_u(\{e \in \mathcal{T}_{Fut(u,[f_2]_u)} \mid e \leq e_2\}) \setminus pre_u(\{e \in \mathcal{T}_{Fut(u,[f_2]_u)} \mid e \leq e_2\})) \setminus post_u(e_2) \\
&\overset{\text{Analogously}}{=} (post_u(\{e \in \mathcal{T}_{Fut(u,[f_2]_u)} \mid e \leq e_1\}) \setminus pre_u(\{e \in \mathcal{T}_{Fut(u,[f_2]_u)} \mid e \leq e_1\})) \setminus post_u(e_1)
\end{aligned}
$$

Remark 1: Here, the separation of the configuration $[e_2]$ of Lemma 2.20 is crucial. This separation coincides with the separation of the conditions made at the start of this proof such that only the conditions in $\mathcal{P}_{Fut(u,[f_2]_u)} \setminus lkc_u(f_2)$ are considered, i.e. it holds that $(In_u \cup post_u(\bigcup_{\{b \in lkc_u(f_2) | b \leq e_2\}} [b]_u)) \cap (\mathcal{P}_{Fut(u,[f_2]_u)} \setminus lkc_u(f_2)) = \emptyset$ and $(In_u \cup pre_u(\bigcup_{\{b \in lkc_u(f_2) | b \leq e_2\}} [b]_u)) \cap (\mathcal{P}_{Fut(u,[f_2]_u)} \setminus lkc_u(f_2)) = \emptyset$. It also holds that $(post_u(\{e \in \mathcal{T}_{Fut(u,[f_2]_u)} \mid e \leq e_2\}) \setminus pre_u(\{e \in \mathcal{T}_{Fut(u,[f_2]_u)} \mid e \leq e_2\})) \cap (\mathcal{P}_{Fut(u,[f_2]_u)} \setminus lkc_u(f_2)) = (post_u(\{e \in \mathcal{T}_{Fut(u,[f_2]_u)} \mid e \leq e_2\}) \setminus pre_u(\{e \in \mathcal{T}_{Fut(u,[f_2]_u)} \mid e \leq e_2\}))$.

Applying $I^{[f_2]}_{[f_1]}$ on both sides of the last equation yields:

$$
\begin{aligned}
& I^{[f_2]}_{[f_1]}(post_u(\{e \in \mathcal{T}_{Fut(u,[f_2]_u)} \mid e \leq e_2\}) \setminus pre_u(\{e \in \mathcal{T}_{Fut(u,[f_2]_u)} \mid e \leq e_2\}) \setminus post_u(e_2)) \\
\overset{\text{Lemma 2.16}}{=}\ & post_u(\{e \in \mathcal{T}_{Fut(u,[f_1]_u)} \mid e \leq I^{[f_2]}_{[f_1]}(e_2)\}) \setminus pre_u(\{e \in \mathcal{T}_{Fut(u,[f_1]_u)} \mid e \leq I^{[f_2]}_{[f_1]}(e_2)\}) \\
& \qquad \setminus post_u(I^{[f_2]}_{[f_1]}(e_2))) \\
\overset{\text{Remark 1 reversed}}{=}\ & (In_u \cup post_u(\bigcup_{\{b \in lkc_u(f_1) | b \leq I^{[f_2]}_{[f_1]}(e_2)\}} [b]_u \cup \{e \in \mathcal{T}_{Fut(u,[f_1]_u)} \mid e \leq I^{[f_2]}_{[f_1]}(e_2)\}) \\
& \setminus pre_u(\bigcup_{\{b \in lkc_u(f_1) | b \leq I^{[f_2]}_{[f_1]}(e_2)\}} [b]_u \cup \{e \in \mathcal{T}_{Fut(u,[f_1]_u)} \mid e \leq I^{[f_2]}_{[f_1]}(e_2)\}) \setminus post_u(I^{[f_2]}_{[f_1]}(e_2)) \\
& \qquad \cap (\mathcal{P}_{Fut(u,[f_1]_u)} \setminus lkc_u(f_1))) \\
& = (lkc_u(I^{[f_2]}_{[f_1]}(e_2)) \setminus post_u(I^{[f_2]}_{[f_1]}(e_2))) \cap (\mathcal{P}_{Fut(u,[f_1]_u)} \setminus lkc_u(f_1)) \\
& = (lkc_u(I^{[f_2]}_{[f_1]}(e_1)) \setminus post_u(I^{[f_2]}_{[f_1]}(e_1))) \cap (\mathcal{P}_{Fut(u,[f_1]_u)} \setminus lkc_u(f_1)). \\
& \qquad = \dots \\
= I^{[f_2]}_{[f_1]}(& post_u(\{e \in \mathcal{T}_{Fut(u,[f_2]_u)} \mid e \leq e_1\}) \setminus pre_u(\{e \in \mathcal{T}_{Fut(u,[f_2]_u)} \mid e \leq e_1\}) \setminus post_u(e_1))
\end{aligned}
$$

Remark 1 reversed: Here, we apply Lemma 2.20 and add the empty set which we removed in remark 1, roughly speaking. The difference is that the empty set here regards $I^{[f_2]}_{[f_1]}(e_2)$ and $lkc_u(f_1)$ instead of e_2 and $lkc_u(f_2)$ as before. It was necessary to remove this set beforehand to apply the isomorphism $I^{[f_2]}_{[f_1]}$.

For the latter set of conditions in $(\mathcal{P}_u \setminus \mathcal{P}_{Fut(u,[f_2]_u)}) \cup lkc_u(f_2)$, it remains to show that $(lkc_u(I^{[f_2]}_{[f_1]}(e_2)) \setminus post_u(I^{[f_2]}_{[f_1]}(e_2))) \cap ((\mathcal{P}_u \setminus \mathcal{P}_{Fut(u,[f_1]_u)}) \cup lkc_u(f_1))$ is the same set as $(lkc_u(I^{[f_2]}_{[f_1]}(e_1)) \setminus post_u(I^{[f_2]}_{[f_1]}(e_1))) \cap ((\mathcal{P}_u \setminus \mathcal{P}_{Fut(u,[f_1]_u)}) \cup lkc_u(f_1))$. In the next step, we apply equivalent transformations to show the set equivalence.

$$
\begin{aligned}
&(lkc_u(I^{[f_2]}_{[f_1]}(e_2)) \setminus post_u(I^{[f_2]}_{[f_1]}(e_2)) \cap ((\mathcal{P}_u \setminus \mathcal{P}_{Fut(u,[f_1]_u)}) \cup lkc_u(f_1)) \\
&\overset{\text{Def. } lkc_u}{=} ((In_u \cup post_u([I^{[f_2]}_{[f_1]}(e_2)])) \setminus pre_u([I^{[f_2]}_{[f_1]}(e_2)]) \setminus \\
&\quad post_u(I^{[f_2]}_{[f_1]}(e_2))) \cap ((\mathcal{P}_u \setminus \mathcal{P}_{Fut(u,[f_1]_u)}) \cup lkc_u(f_1)) \\
&\overset{\text{Lemma 2.20}}{=} (In_u \cup post_u(\bigcup_{\{b \in lkc_u(f_1) \mid b \leq I^{[f_2]}_{[f_1]}(e_2)\}} [b]_u \cup \{e \in \mathcal{T}_{Fut(u,[f_1]_u)} \mid e \leq I^{[f_2]}_{[f_1]}(e_2)\}) \\
&\setminus pre_u(\bigcup_{\{b \in lkc_u(f_1) \mid b \leq I^{[f_2]}_{[f_1]}(e_2)\}} [b]_u \cup \{e \in \mathcal{T}_{Fut(u,[f_1]_u)} \mid e \leq I^{[f_2]}_{[f_1]}(e_2)\}) \setminus post_u(I^{[f_2]}_{[f_1]}(e_2)) \\
&\quad \cap ((\mathcal{P}_u \setminus \mathcal{P}_{Fut(u,[f_1]_u)}) \cup lkc_u(f_1)) \\
&\overset{\text{Remark 2}}{=} = (In_u \cup post_u(\bigcup_{\{b \in lkc_u(f_1) \mid b \leq I^{[f_2]}_{[f_1]}(e_2)\}} [b]_u)) \\
&\setminus (pre_u(\bigcup_{\{b \in lkc_u(f_1) \mid b \leq I^{[f_2]}_{[f_1]}(e_2)\}} [b]_u) \cup \{b \in lkc_u(f_1) \mid b \leq I^{[f_2]}_{[f_1]}(e_2)\}))
\end{aligned}
$$

Remark 2: Here, we simplify the set by removing empty sets. Since $((\mathcal{P}_u \setminus \mathcal{P}_{Fut(u,[f_1]_u)}) \cup lkc_u(f_1))$ contains no conditions of $\mathcal{P}_{Fut(u,[f_1]_u)}$ except for the conditions in $lkc_u(f_1)$ the conditions in $post_u(\{e \in \mathcal{T}_{Fut(u,[f_1]_u)} \mid e \leq I^{[f_2]}_{[f_1]}(e_2)\})$ are not in the set, i.e. it holds that $post_u(\{e \in \mathcal{T}_{Fut(u,[f_1]_u)} \mid e \leq I^{[f_2]}_{[f_1]}(e_2)\}) \cap ((\mathcal{P}_u \setminus \mathcal{P}_{Fut(u,[f_1]_u)}) \cup lkc_u(f_1)) = \emptyset$.

Furthermore, there is no condition in $post_u(I^{[f_2]}_{[f_1]})(e_2))$, which is also a condition in $((\mathcal{P}_u \setminus \mathcal{P}_{Fut(u,[f_1]_u)}) \cup lkc_u(f_1))$, i.e. it holds that $post_u(I^{[f_2]}_{[f_1]})(e_2)) \cap ((\mathcal{P}_u \setminus \mathcal{P}_{Fut(u,[f_1]_u)}) \cup lkc_u(f_1)) = \emptyset$.

At last regarding remark 2: the intersection of the sets $pre_u(\{e \in \mathcal{T}_{Fut(u,[f_1]_u)} \mid e \leq I^{[f_2]}_{[f_1]}(e_2)\}))$ and $((\mathcal{P}_u \setminus \mathcal{P}_{Fut(u,[f_1]_u)}) \cup lkc_u(f_1))$ only contains conditions which are in $lkc_u(f_1)$, i.e. it holds that $pre_u(\{e \in \mathcal{T}_{Fut(u,[f_1]_u)} \mid e \leq I^{[f_2]}_{[f_1]}(e_2)\})) \cap ((\mathcal{P}_u \setminus \mathcal{P}_{Fut(u,[f_1]_u)}) \cup lkc_u(f_1)) = \{b \in lkc_u(f_1) \mid b \leq I^{[f_2]}_{[f_1]}(e_2)\})$.

Applying the same equivalent transformations on the right side of the original equation, i.e. on $(lkc_u(I^{[f_2]}_{[f_1]}(e_1)) \setminus post_u(I^{[f_2]}_{[f_1]}(e_1))) \cap ((\mathcal{P}_u \setminus \mathcal{P}_{Fut(u,[f_1]_u)}) \cup lkc_u(f_1))$, yields that it is sufficient to show that $\{b \in lkc_u(f_1) \mid b \leq I^{[f_2]}_{[f_1]}(e_2)\} = \{b \in lkc_u(f_1) \mid b \leq I^{[f_2]}_{[f_1]}(e_1)\}$ to get the set equivalence. From Lemma 2.16 follows that it is sufficient to show $\{b \in lkc_u(f_2) \mid b \leq e_2\} = \{b \in lkc_u(f_2) \mid b \leq e_1\}$.

Assume there exists a condition b such that $b \in \{b \in lkc_u(f_2) \mid b \leq e_2\}$ and $b \notin \{b \in lkc_u(f_2) \mid b \leq e_1\}$ (proof by contradiction). Since $b \leq e_2$, there exists a sequence of causal successors $be^1b^1e^2 \dots e_2$. Therefore, $b \notin lkc_u(e_2) \setminus post_u(e_2)$ holds.

Because $b \not\leq e_1$, $b \in lkc_u(f_2)$ and $e_1 \in \mathcal{T}_{Fut(u,[f_2]_u)}$, $b||e_1$ holds. ($b\#e_1$ is a contradiction to $e_1 \in \mathcal{T}_{Fut(u,[f_2]_u)}$.)

By Lemma 2.19, $b \in Fut(u,[e_1]_u)$. Since $lkc_u(e_1)$ is a maximal set of pairwise concurrent conditions, there exists $b' \in lkc_u(e_1) \setminus post_u(e_1) : b' \leq b$. It follows that $b' \leq b \leq e_2$. Thus, $b' \notin lkc_u(e_2) \setminus post_u(e_2)$, which is a contradiction to $lkc(e_1) \setminus post(e_1) = lkc(e_2) \setminus post(e_2)$. □

The next lemma states that the set of concurrent events of two events e_1, e_2 are equal if $lkc(e_2') \setminus post(e_2') = lkc(e_2) \setminus post(e_2)$ holds. Again, this lemma aims at the case in a 4-player game where there are two events e_2' ande_2 in the synchronisation segment of an event e. From Lemma 3.27 it follows for the two events e_2 and e_2' in the synchronisation segment of e that $lkc(e_2') \setminus post(e_2') = lkc(e_2) \setminus post(e_2)$.

Lemma 3.28 (Set of concurrent events). *Let B be a branching process and $e_2', e_2 \in \mathcal{T}$. If $lkc(e_2') \setminus post(e_2') = lkc(e_2) \setminus post(e_2)$ it holds that $\{x \in \mathcal{T} \cup \mathcal{P} \mid x||e_2'\} = \{x \in \mathcal{T} \cup \mathcal{P} \mid x||e_2\}$.*

Proof by induction over the set of finite configurations of $Fut(B,[e_2])$. By Lemma 2.19 it holds that all nodes of B which are concurrent to e_2 are nodes in $Fut(B,[e_2])$. Therefore, the proposition can be shown by induction over the finite configurations of $Fut(B,[e_2])$.

Induction assumption: $\forall x \in \mathcal{T}_{Fut(B,[e_2])} \cup \mathcal{P}_{Fut(B,[e_2])} : (x||e_2 \wedge [x]_{Fut(B,[e_2])} \preceq C_n) \implies x||e_2'$.

Induction base case: $C_1 = \emptyset$. Since $lkc(e_2') \setminus post(e_2') = lkc(e_2) \setminus post(e_2)$, for all $b \in lkc(e_2) \setminus post(e_2)$ it holds that $b||e_2'$ since all conditions in $lkc(e_2') \setminus post(e_2')$ are concurrent to e_2'.

Induction step: Let C_{n+1} be the next finite configuration regarding $\preceq$. If there exists no $e \in Fut(B,[e_2])$ such that $[e]_{Fut(B,[e_2])} = C_{n+1}$ the induction property holds by induction assumption. Otherwise, let e be such an event. We show that $e||e_2$ implies $e||e_2'$. If $e||e_2$ it holds that $\forall b \in pre(e) : b||e_2$. By induction assumption it holds that $\forall b \in pre(e) : b||e_2'$. It follows that $e||e_2'$ and $\forall b \in post(e).b||e_2'$. □

Now, we can show that the set of events of the synchronisation segments of two events e_2', e_2, where $lkc(e_2') \setminus post(e_2') = lkc(e_2) \setminus post(e_2)$, are equal since every event e in the synchronisation segment of e_2 is concurrent to e_2 by definition. This is an implication of Lemma 3.28 that the set of concurrent events of e_2 and e_2' are equal here.

Lemma 3.29 (Segment equivalence). *Let B be a token-partitioning branching process and $e_1, e_2 \in \mathcal{T}$. If $lkc(e_1) \setminus post(e_1) = lkc(e_2) \setminus post(e_2)$ it holds that $\{x \in \mathcal{T}_{Sg(B,e_1)} \cup \mathcal{P}_{Sg(B,e_1)} \mid x||e_1\} = \{x \in \mathcal{T}_{Sg(B,e_2)} \cup \mathcal{P}_{Sg(B,e_2)} \mid x||e_2\}$.*

Proof. By Lemma 3.28, the sets of concurrent nodes of e_1 and e_2, respectively, are the same. By Def. 3.20, all events in $\mathcal{T}_{Sg(B,e_1)}$ are concurrent to e_1 and all events in $\mathcal{T}_{Sg(B,e_2)}$ are concurrent to e_2. It follows that the set of events $\mathcal{T}_{Sg(B,e_1)}$ and $\mathcal{T}_{Sg(B,e_2)}$ are equal. Thus, it holds that $\{x \in \mathcal{T}_{Sg(B,e_1)} \cup \mathcal{P}_{Sg(B,e_1)} \mid x||e_1\} = \{x \in \mathcal{T}_{Sg(B,e_2)} \cup \mathcal{P}_{Sg(B,e_2)} \mid x||e_2\}$. □

Now, we show that $sqc(B, e_1, e_2)$, $e \in \mathcal{T}_{Sg(B,e_2)}$, $|pre(e_2)| = |pre(e)| = 2$, $|I_P| = 4$, and $I^{[e]}_{I^{[e_2]}_{[e_1]}(e)}(e_2) = e_1$ imply $sqc(B, e, I^{[e_2]}_{[e_1]}(e))$. The additional condition $I^{[e]}_{I^{[e_2]}_{[e_1]}(e)}(e_2) = e_1$ holds if e_2 is in the synchronisation segment of e. However, the lemma is formalized more general and we show that the event e_2 is in the synchronisation segment of e when constructing a winning strategy from a 4-player winning prefix. Note that $I^{[e_2]}_{[e_1]}$ restricted to the synchronisation segment of e_2 is the isomorphism $I^{Sg(B,e_2)}_{Sg(B,e_1)}$.

In Fig. 3.13, Lemma 3.30 is illustrated. There, the events e_2 and e are concurrent. Since $I^{[e_2]}_{[e_1]}$ preserves concurrency, the events e_1 and $I^{[e_2]}_{[e_1]}(e)$ are also concurrent. Lemma 3.30 states in this setting that $sqc(B, e_1, e_2)$ implies $sqc(B, e, I^{[e_2]}_{[e_1]}(e))$. The setting described the exact number of tokens which participate in e_2 and e, respectively. Both events have 2 preconditions and the net is a token-partitioning net with 4 tokens.

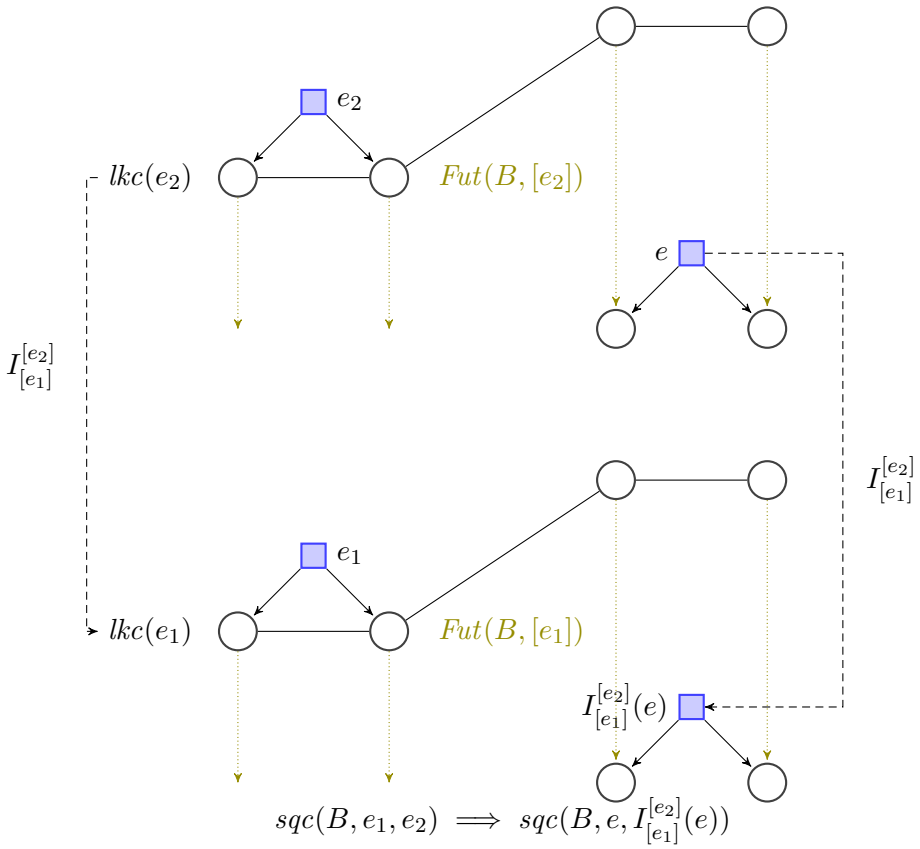


Figure 3.13: Graphical depiction of Lemma 3.30. The events e_2 and e are concurrent. In the depicted setting $sqc(B, e_1, e_2)$ implies $sqc(B, e, I^{[e_2]}_{[e_1]}(e))$.

Lemma 3.30 (Synchronisation equivalence). *Let B be a token-partitioning branching process of a strategy τ (omitting the index τ) and $e_1, e_2, e \in \mathcal{T}$ such that $sqc(B, e_1, e_2)$,*

$e \in \mathcal{T}_{Sg(B,e_2)}$, $|pre(e_2)| = |pre(e)| = 2$, $|I_P| = 4$ *and* $I^{[e]}_{I^{[e_2]}_{[e_1]}(e)}(e_2) = e_1$. *Then, it holds that* $sqc(B, e, I^{[e_2]}_{[e_1]}(e))$.

Proof. Note that we omit the index τ of the branching process $B = B_\tau$. If $prc(B, e_1, e_2)$, by Lemma 3.5 it follows that $e = I^{[e_2]}_{[e_1]}(e)$ and the statement holds because $sqc(B, e, e)$ holds.

Otherwise, we show the 3 properties of synchronisation equivalent cuts of Def. 3.24. The first property that $|pre(e)| = |pre(I^{[e_2]}_{[e_1]}(e))| \geq 2$ holds by assumption.

Since e_1 and e_2 are segment equivalent, $pre(e) \geq 2$ and $e \in \mathcal{T}_{Sg(B,e_2)}$, it holds that e and $I^{[e_2]}_{[e_1]}(e)$ are segment equivalent by definition of $sqc(B, e_1, e_2)$, property 3, (Def. 3.24). Thus, the second property of synchronisation equivalent cuts holds for e and $I^{[e_2]}_{[e_1]}(e)$.

It remains to show that for all $e'_2 \in \mathcal{T}_{Sg(B,e)}$ it holds that $|pre(e'_2)| \geq 2$ implies that e'_2 and $I^{[e]}_{I^{[e_2]}_{[e_1]}(e)}(e'_2)$ are segment equivalent.

Let e'_2 be such an event. By definition of the synchronisation segment of e, $e'_2||e$ holds. Since $|pre(e'_2)| \geq 2$, $|I_P| = 4$ and $|pre(e)| = 2$ it follows that $Ind(e_2) = Ind(e'_2) = I_P \setminus Ind(e)$. By Lemma 3.26 it follows that $lkc(e'_2) \setminus post(e'_2) = lkc(e_2) \setminus post(e_2)$.

By Lemma 3.29, $\{x \in \mathcal{T}_{Sg(B,e'_2)} \cup \mathcal{P}_{Sg(B,e'_2)} \mid x||e'_2\} = \{x \in \mathcal{T}_{Sg(B,e_2)} \cup \mathcal{P}_{Sg(B,e_2)} \mid x||e_2\}$ holds.

By Lemma 3.27, it holds that $lkc(I^{[e]}_{I^{[e_2]}_{[e_1]}(e)}(e'_2)) \setminus post(I^{[e]}_{I^{[e_2]}_{[e_1]}(e)}(e'_2)) = lkc(I^{[e]}_{I^{[e_2]}_{[e_1]}(e)}(e_2)) \setminus post(I^{[e]}_{I^{[e_2]}_{[e_1]}(e)}(e_2)) \overset{Assumption}{=} lkc(e_1) \setminus post(e_1)$.

By Lemma 3.29, $\{x \in \mathcal{T}_{Sg(B,e_1)} \cup \mathcal{P}_{Sg(B,e_1)} \mid x||e_1\} = \{x \in \mathcal{T}_{Sg(B,I^{[e]}_{I^{[e_2]}_{[e_1]}(e)}(e'_2))} \cup \mathcal{P}_{Sg(B,I^{[e]}_{I^{[e_2]}_{[e_1]}(e)}(e'_2))} \mid x||I^{[e]}_{I^{[e_2]}_{[e_1]}(e)}(e'_2)\}$ holds.

We continue with showing that e'_2 and $I^{[e]}_{I^{[e_2]}_{[e_1]}(e)}(e'_2)$ are segment equivalent. Therefore, the function $I^{Sg(B,e'_2)}_{Sg(B,I^{[e]}_{I^{[e_2]}_{[e_1]}(e)}(e'_2))}$ from $Sg(B, e)$ to $Sg(B, I^{[e]}_{I^{[e_2]}_{[e_1]}(e)}(e'_2))$ is defined as follows:

$$I^{Sg(B,e'_2)}_{Sg(B,I^{[e]}_{I^{[e_2]}_{[e_1]}(e)}(e'_2))}(x) = \begin{cases} I^{Sg(B,e_2)}_{Sg(B,e_1)}(x) & \text{if } x||e'_2 \\ p_i(post(I^{[e]}_{I^{[e_2]}_{[e_1]}(e)}(e'_2)))) & \text{if } x \in P_i \cap post(e'_2). \end{cases}$$

The function $I^{Sg(B,e'_2)}_{Sg(B,I^{[e]}_{I^{[e_2]}_{[e_1]}(e)}(e'_2))}$ is bijective since $\{x \in \mathcal{T}_{Sg(B,e'_2)} \cup \mathcal{P}_{Sg(B,e'_2)} \mid x||e'_2\} = \{x \in \mathcal{T}_{Sg(B,e_2)} \cup \mathcal{P}_{Sg(B,e_2)} \mid x||e_2\}$ and $\{x \in \mathcal{T}_{Sg(B,e_1)} \cup \mathcal{P}_{Sg(B,e_1)} \mid x||e_1\} = \{x \in \mathcal{T}_{Sg(B,I^{[e]}_{I^{[e_2]}_{[e_1]}(e)}(e'_2))} \cup \mathcal{P}_{Sg(B,I^{[e]}_{I^{[e_2]}_{[e_1]}(e)}(e'_2))} \mid x||I^{[e]}_{I^{[e_2]}_{[e_1]}(e)}(e'_2)\}$ hold and $p_i(post(I^{[e]}_{I^{[e_2]}_{[e_1]}(e)}(e'_2))))$ if $x \in P_i \cap post(e)$ defines a bijection on $post(e)$ to $post(I^{[e]}_{I^{[e_2]}_{[e_1]}(e)}(e'_2))$. By definition of

$Sg(B, e_2')$ there a no events which are causal successors of e_2' in $\mathcal{T}_{Sg(B,e_2)}$ such that there are no more nodes to consider.

Since $I^{Sg(B,e_2)}_{Sg(B,e_1)}(x)$ is a branching process homomorphism $I^{Sg(B,e_2')}_{Sg(B,I^{[e]}_{I^{[e_2]}_{[e_1]}(e)}(e_2'))}$ satisfies the branching process homomorphism properties for all events that are concurrent to e_2'.

Since $I^{Sg(B,e_2)}_{Sg(B,e_1)}(x)$ preserves the strategy τ of all conditions, which are concurrent to e_2, the synchronisation equivalence holds. □

Loosely speaking, Lemma 3.30 implies that repeating a strategy after an event e_2 from a synchronisation equivalent cut $lkc(e_1)$ ($sqc(B, e_1, e_2)$) guarantees that for any concurrent event in the synchronisation segment of e_2 that is also synchronisation equivalent to another event the strategy is still copied from the same cut, i.e. from $lkc(e_1)$ here.

The next Lemma 3.31, states that the isomorphisms of two different pairs of synchronisation equivalent cuts $sqc(B, e_1, e_2)$ and $sqc(B, e_1', e_2')$ are equal on their joint future if $I^{[e_2]}_{[e_1]}(e_2') = e_1'$ and $I^{[e_2']}_{[e_1']}(e_2) = e_1$ hold, which is the case if e_2 and e_2' are in each other's synchronisation segment as shown in the construction of a winning strategy later on.

In Fig. 3.14, Lemma 3.31 is illustrated. The events e_2 and e_2' are concurrent and every token of the underlying net either participate in e_2 or e_2'. This does not have to be 2 tokens each and overall 4 tokens. However, we illustrate it this way since this is the case it is applied on in the construction of a winning strategy.

Lemma 3.31 (Preserved joint future). *Let N be a token-partitioning Petri net and $e_1, e_2, e_1', e_2' \in \mathcal{T}_u$ such that $lkm_u(e_1) = lkm_u(e_2)$, $lkm_u(e_1') = lkm_u(e_2')$, $e_2 || e_2'$, $Ind(e_2) \cup Ind(e_2') = I_P$, $Ind(e_2) = Ind(e_1)$, $Ind(e_2') = Ind(e_1')$, $I^{[e_2]}_{[e_1]}(e_2') = e_1'$ and $I^{[e_2']}_{[e_1']}(e_2) = e_1$. Then,*

$$\forall x \in \mathcal{P}_{Fut(u, Cut([e_2]_u \cup [e_2']_u)} \cup \mathcal{T}_{Fut(u, Cut([e_2]_u \cup [e_2']_u)} : I^{[e_2]}_{[e_1]}(x) = I^{[e_2']}_{[e_1']}(x)$$

holds.

Proof by induction over the set of finite configurations of $Fut(u, Cut([e_2]_u \cup [e_2']_u)$. The set $[e_2] \cup [e_2']$ is a configuration by Lemma 2.15. Therefore, we show the statement by induction over the set of finite configurations of $Fut(u, Cut([e_2] \cup [e_2']))$.

Induction assumption: for all $x \in \mathcal{P}_{Fut(u, Cut([e_2]_u \cup [e_2']_u)} \cup \mathcal{T}_{Fut(u, Cut([e_2]_u \cup [e_2']_u)}$ it holds that $[x]_{Fut(u, Cut([e_2]_u \cup [e_2']_u)} \preceq C_n$ implies $I^{[e_2]}_{[e_1]}(x) = I^{[e_2']}_{[e_1']}(x)$.

Induction base case: Since $C_1 = \emptyset$ and $Ind(e_2) \cup Ind(e_2') = I_P$ for all conditions $b \in Cut_{Fut(u, Cut([e_2]_u \cup [e_2']_u))}(\emptyset) = In_{Fut(u, Cut([e_2]_u \cup [e_2']_u))}$ either $b \in post(e_2)$ or $b \in post(e_2')$ holds. Assume $b \in post(e_2')$. It holds that $I^{[e_2']}_{[e_1']}(post(e_2')) = post(e_1')$ since $Ind(e_2') = Ind(e_1')$. Since $I^{[e_2]}_{[e_1]}(e_2') = e_1'$ by assumption, it holds that $I^{[e_2]}_{[e_1]}(post(e_2')) = post(I^{[e_2]}_{[e_1]}(e_2')) = post(e_1')$. Since u is a branching process of a safe net it follows for all $b \in post(e_2')$ that $I^{[e_2]}_{[e_1]}(b) = I^{[e_2']}_{[e_1']}(b)$. Analogously, it follows $I^{[e_2]}_{[e_1]}(b) = I^{[e_2']}_{[e_1']}(b)$ if $b \in post(e_2)$.

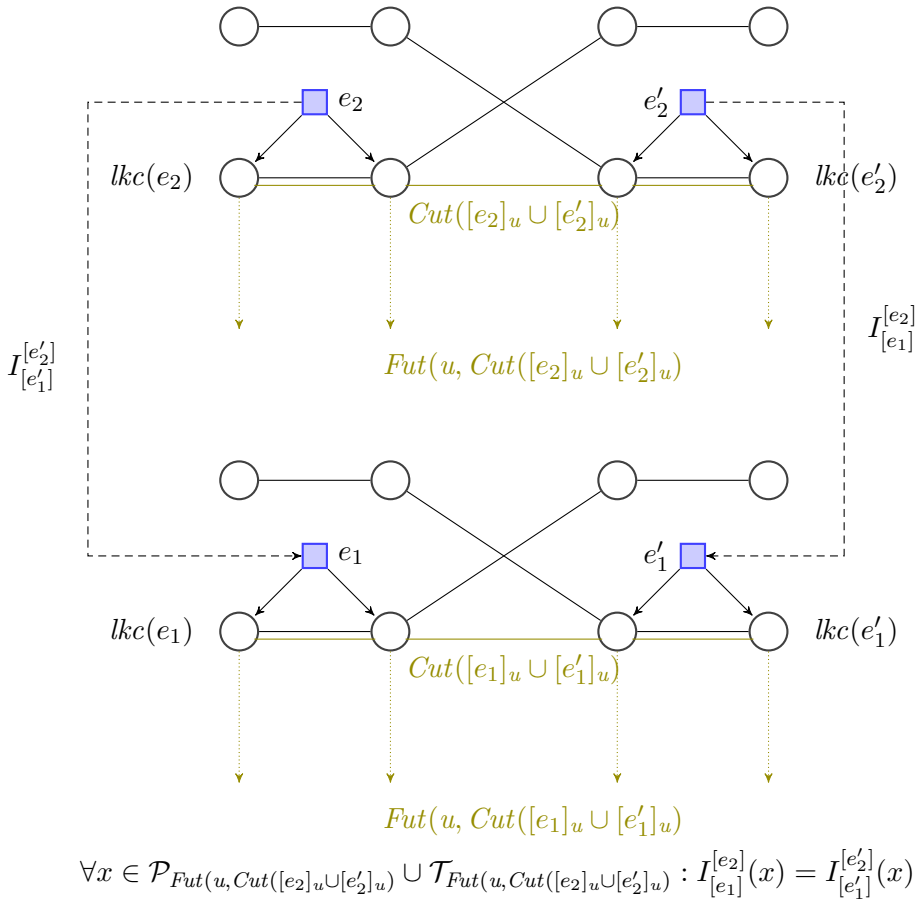


Figure 3.14: Graphical depiction of the setting in Lemma 3.31. The isomorphisms $I^{[e_2]}_{[e_1]}$ and $I^{[e'_2]}_{[e'_1]}$ are the same isomorphism restricted to the nodes in the future of $Cut([e_2]_u \cup [e'_2]_u)$.

Induction step: Let C_{n+1} be the next configuration regarding $\preceq$ and e an event in $\mathcal{T}_{Fut(u,Cut([e_2]_u \cup [e'_2]_u)}$ such that $[e]_{Fut(u,Cut([e_2]_u \cup [e'_2]_u)} = C_{n+1}$. If there exists no such e the induction property holds by induction assumption. It holds that $pre(I^{[e_2]}_{[e_1]}(e)) = I^{[e_2]}_{[e_1]}(pre(e)) \overset{I.A.}{=} I^{[e'_2]}_{[e'_1]}(pre(e)) = pre(I^{[e'_2]}_{[e'_1]}(e))$ and $I^{[e_2]}_{[e_1]}(e) = I^{[e'_2]}_{[e'_1]}(e)$ follows from the property of a branching process that there a no duplicates of events with the same label and the same preset. The induction assumption follows for all $b \in post(e)$ by the property of a branching process homomorphism which preserves the postset (and presets) of events. □

The next lemma is an implication of Lemma 3.27. We show that a partial repetition $prc(B, e_1, e_2)$ is preserved by any isomorphism $I^{[f_2]}_{[f_1]}$ ($lkm(f_2) = lkm(f_1)$ is provided)

if e_1 and e_2 are in the future of f_2. This is an important lemma that is used in the construction of a winning strategy. It is used to show that is does not matter if a strategy is repeated from a partial repetition $prc(B, e_1, e_2)$ or the synchronisation equivalence cuts $sqc(B, f_1, f_2)$ if $f_2||e_2$ holds. The setting is similar to the setting in Fig. 3.12 except that besides $lkc(e_2) \setminus post(e_2) = lkc(e_1) \setminus post(e_1)$ also $lkm(e_2) = lkm(e_1)$, i.e. $prc(B, e_1, e_2)$ holds.

Lemma 3.32 (Isomorphism preserves partial repetition). *Let N be a net, $f_1, f_2 \in \mathcal{T}_u$ and $e_1, e_2 \in \mathcal{T}_{Fut(u,[f_2]u)}$ such that $prc(u, e_1, e_2)$ and $lkm_u(f_2) = lkm_u(f_1)$ hold. Then, $prc(u, I^{[f_2]}_{[f_1]}(e_1), I^{[f_2]}_{[f_1]}(e_2))$ holds.*

Proof. By Lemma 3.27, $lkc(I^{[f_2]}_{[f_1]}(e_1)) \setminus post(I^{[f_2]}_{[f_1]}(e_1)) = lkc(I^{[f_2]}_{[f_1]}(e_2)) \setminus post(I^{[f_2]}_{[f_1]}(e_2))$ holds.

The other property of a partial repetition, $lkm_u(I^{[f_2]}_{[f_1]}(e_1)) = lkm_u(I^{[f_2]}_{[f_1]}(e_2))$, follows from $I^{[f_2]}_{[f_1]}$ being a branching process isomorphism, which preserves the labels of the conditions in $post(e_1)$ and in $post(e_2)$. Since $lkm_u(e_2) = lkm_u(e_1)$ and $lkc(I^{[f_2]}_{[f_1]}(e_1)) \setminus post(I^{[f_2]}_{[f_1]}(e_1)) = lkc(I^{[f_2]}_{[f_1]}(e_2)) \setminus post(I^{[f_2]}_{[f_1]}(e_2))$ it follows that the last known markings $lkm_u(I^{[f_2]}_{[f_1]}(e_2))$ and $lkm_u(I^{[f_2]}_{[f_1]}(e_1))$ are equal. □

The next Lemma 3.33 shows that applying the isomorphism of synchronisation equivalent cuts and the isomorphism of partial repetition cuts in the future of a synchronisation equivalent cut commute. As in the previous Lemma 3.32, this lemma is necessary to show that the choice whether the future of a partial repetition or the future of synchronisation equivalent cuts is repeated results in the same strategy.

In Fig. 3.15, a commutating diagram of the isomorphism of a partial repetition and a synchronisation equivalence is shown. Note that we do not require synchronisation equivalence but only the isomorphism $I^{[f_2]}_{[f_1]}$ from two events $f_2, f_1 \in \mathcal{T}_u$. This isomorphism exists if f_2 and f_1 are synchronisation equivalent since $lkm_u(f_2) = lkm_u(f_1)$ is a necessary condition for synchronisation equivalence. Furthermore, note that Lemma 3.32 is essential here. It states that $prc(u, e_1, e_2)$ implies $prc(u, I^{[f_2]}_{[f_1]}(e_1), I^{[f_2]}_{[f_1]}(e_2))$ in this setting.

Lemma 3.33 (Partial repetition and synchronisation equivalence commute). *Let N be a net, $f_1, f_2 \in \mathcal{T}_u$, $e_1, e_2 \in \mathcal{T}_{Fut(u,[f_2]_u)}$ and $e_2||f_2$. If $lkm_u(f_1) = lkm_u(f_2)$ and $prc(u, e_1, e_2)$ it holds that for all $x \in (\mathcal{T}_{Fut(u,Cut([e_2]_u \cup [f_2]_u)} \cup \mathcal{P}_{Fut(u,Cut([e_2]_u \cup [f_2]_u)})$: $I^{[f_2]}_{[f_1]}(I^{[e_2]}_{[e_1]}(x)) = I^{[I^{[f_2]}_{[f_1]}(e_2)]}_{[I^{[f_2]}_{[f_1]}(e_1)]}(I^{[f_2]}_{[f_1]}(x))$.*

Proof by induction over the set of finite configurations of $Fut(u, Cut([e_2]_u \cup [f_2]_u)$. Since e_2 and f_2 are concurrent $[e_2]_u \cup [f_2]_u$ is a configuration in u by Lemma 2.15 such that the proof by induction over the set of finite configurations of $Fut(u, Cut([e_2]_u \cup [f_2]_u)$ is viable.

Induction base case: For $b \in Cut([e_2]_u \cup [f_2]_u)$ the following cases are distinguished:

$$\begin{array}{ccc} Fut(u, Cut([e_2]_u \cup [f_2]_u) & \xrightarrow{I^{[e_2]}_{[e_1]}} & Fut(u, Cut([e_1]_u \cup [f_2]_u)) \\ \Big\downarrow{\scriptstyle I^{[f_2]}_{[f_1]}} & & \Big\downarrow{\scriptstyle I^{[f_2]}_{[f_1]}} \\ Fut(u, Cut([I^{[f_2]}_{[f_1]}(e_2)]_u \cup [f_1]_u)) & \xrightarrow[I^{[I^{[f_2]}_{[f_1]}(e_1)]}_{[I^{[f_2]}_{[f_1]}(e_2)]}]{} & Fut(u, Cut([e_1]_u \cup [f_1]_u)) \end{array}$$

Figure 3.15: Commutative diagram of the isomorphism of a partial repetition $prc(u, e_1, e_2)$ and the isomorphism $I^{[f_2]}_{[f_1]}$. The isomorphism $I^{[f_2]}_{[f_1]}$ exists if f_2 and f_1 are synchronisation equivalent.

Case $b||e_2$**:** By Lemma 3.5, $I^{[e_2]}_{[e_1]}(b) = b$. By Lemma 3.27, Lemma 2.18 and Lemma 3.5, $I^{I^{[f_2]}_{[f_1]}(e_2)}_{I^{[f_2]}_{[f_1]}(e_1)}(I^{[f_2]}_{[f_1]})(b)) = I^{[f_2]}_{[f_1]}(b)$ holds and the property follows.

Case $b \in post_u(e_2)$**:** Because of $prc(u, e_1, e_2)$, it holds that $I^{[e_2]}_{[e_1]}(post_u(e_2)) = post_u(e_1)$ and $I^{[I^{[f_2]}_{[f_1]}(e_2)]}_{[I^{[f_2]}_{[f_1]}(e_1)]}(post_u(I^{[f_2]}_{[f_1]}(e_2))) = post_u(I^{[f_2]}_{[f_1]}(e_1))$ by Lemma 3.9. Using the branching process homomorphism property, it follows that $I^{[f_2]}_{[f_1]}(I^{[e_2]}_{[e_1]}(post_u(e_2))) = I^{[f_2]}_{[f_1]}(post_u(e_1)) = post_u(I^{[f_2]}_{[f_1]}(e_1)) = I^{[I^{[f_2]}_{[f_1]}(e_2)]}_{[I^{[f_2]}_{[f_1]}(e_1)]}(post_u(I^{[f_2]}_{[f_1]}(e_2))) = I^{[I^{[f_2]}_{[f_1]}(e_2)]}_{[I^{[f_2]}_{[f_1]}(e_1)]}(I^{[f_2]}_{[f_1]}(post_u(e_2)))$ and the property holds.

Induction assumption: The property holds for all nodes $x \in Fut(u, Cut([e_2]_u \cup [f_2]_u)$ such that $[x]_{Fut(u,Cut([e_2]_u \cup [f_2]_u)} \preceq C_n$.

Induction step: Let C_{n+1} be the next configuration regarding $\preceq$. If there exists an $e \in C_{n+1}$ such that there is no lesser configuration that contains e we proceed as follows. Otherwise the property holds by induction assumption.

Let $e \in C_{n+1}$ such that $[e]_{Fut(u,Cut([e_2]_u \cup [f_2]_u)} = C_{n+1}$. It holds that $pre_u(I^{[f_2]}_{[f_1]}(I^{[e_2]}_{[e_1]}(e)))$ $= (I^{[f_2]}_{[f_1]}(I^{[e_2]}_{[e_1]}(pre_u(e))))$ by the property of branching process homomorphisms. The induction assumption can be applied for the conditions in $pre_u(e)$. Thus, $I^{[f_2]}_{[f_1]}(I^{[e_2]}_{[e_1]}(pre_u(e)))$ $= I^{[I^{[f_2]}_{[f_1]}(e_2)]}_{[I^{[f_2]}_{[f_1]}(e_1)]}(I^{[f_2]}_{[f_1]}(pre_u(e))) = pre_u(I^{[I^{[f_2]}_{[f_1]}(e_2)]}_{[I^{[f_2]}_{[f_1]}(e_1)]}(I^{[f_2]}_{[f_1]}(e)))$ and the property follows because of the property of a branching process (Def. 2.4) that two events with the same label and same preset must be equal. □

Having shown quite a few properties of synchronisation equivalent cuts, we can proceed with the construction of a winning strategy starting with a winning prefix.

3.3.2 Winning 4-player Prefix

Before defining a winning 4-player prefix, we define a 4-player cut-off event similar to a 2-player cut-off event. The difference is that synchronisation equivalent cuts are used as condition for the cut-off instead of partial repetition cuts.

Definition 3.34 (4-player cut-off event). Let B be a branching process. An event $e \in \mathcal{T}$ is a *4-player cut-off event* if there exists $e' \in \mathcal{T}$ such that

1. $sqc(B, e, e')$, and
2. $[e'] \prec [e]$.

$\triangleleft$

As in the 2-player case: for an event $e \in \mathcal{T}$, the smallest (w.r.t. $\preceq$) e' that satisfies the properties 1 and 2 in the definition of a 4-player cut-off event (Def. 3.34) is denoted as $R(e)$. Later on, this is used to determine how a winning strategy is constructed from a winning prefix.

The *4-player non-cut-off events* are the set of events that are not preceded by a cut-off event.

Definition 3.35 (4-player non-cut-off events). For a branching process B of a token-partitioning net with 4 tokens the set of 4-player non-cut-off events $NCO(B)$ is the set $NCO(B) = \{e \in \mathcal{T} \mid \neg\exists e' \in \mathcal{T} : e' \leq e \wedge \text{e' is a 4-player cut-off event.}\}$ $\triangleleft$

The next definition determines where the constructed winning strategy is repeated. Informally speaking, the strategy is repeated after every cut-off event e that occurs first w.r.t. the causal successor relation. Then, the strategy is copied from the corresponding synchronisation equivalent cut, i.e. the cut which is minimal w.r.t. $\preceq$ leading to e being a cut-off event which is $R(e)$.

Definition 3.36 (4-player repetitions). For a winning four-player prefix τ, the set of repetitions is defined as the set $REP(B_\tau) = \{(R(e), e) \mid e \in \mathcal{T}_\tau \wedge \neg\exists e' \in \mathcal{T}_\tau : e' < e \wedge e' \text{ is a 4-player cut-off event.}\}$. $\triangleleft$

A winning 4-player prefix τ must be winning in all configurations that are a subset of the non-cut-off events.

Definition 3.37 (Winning 4-player prefix). A *winning 4-player prefix* w.r.t. a total adequate order $\preceq$ is a strategy τ such that for the branching process B_τ holds that

- for all configurations $C \subseteq NCO(B_\tau)$ the branching process B_τ is winning in C.

$\triangleleft$

We present two examples of winning 4-player prefixes.

Example 3.38 (Winning 4-player prefix). A branching process B_τ of a winning 4-player prefix τ is shown in Fig. 3.16. The last known cuts of t^1 and t^2 are synchronisation equivalent cuts, i.e. $sqc(B_\tau, t^1, t^2)$ holds. In the synchronisation segment of t^1 the first system player allows to open its door via o_1^1 and the second system player denies to open its door d_2^1, i.e. $\tau(S_{13}^1) = \{o_1\}$ and $\tau(S_{23}^1) = \{d_2\}$ hold. Since o_1^1 and d_2^1 are not concurrent to t^1 those events are not in the synchronisation segment of t^1.

The environment players on $W_1^1, W_2^1, W_1^2, W_2^2$ allow all labels of events. Therefore, as required in the definition of synchronisation equivalent cuts for all conditions $b \in Sg(B_\tau, t^2) \backslash post_\tau(t^2)$ it holds that $\tau(b) = \tau(I_{Sg(B_\tau,t^1)}^{Sg(B_\tau,t^2)}(b))$, i.e. $\tau(W_1^2) = \tau(I_{Sg(B_\tau,t^1)}^{Sg(B_\tau,t^2)}(W_1^2)) = \tau(W_1^1)$ and $\tau(W_2^2) = \tau(I_{Sg(B_\tau,t^1)}^{Sg(B_\tau,t^2)}(W_2^2)) = \tau(W_2^1)$.

For the conditions S_{13}^2 and S_{23}^2 in the postset of t^2, it is not necessary that $\tau(b) = \tau(I_{Sg(B_\tau,t^1)}^{Sg(B_\tau,t^2)}(b))$ holds by definition of segment equivalence (Def. 3.23). This is valid since a configuration containing any cut-off event (t^2) does not need to be winning. Therefore, the prefix does not need to be continued any further. Condition 3 of Def. 3.24 is satisfied trivially since there is no event in the synchronisation segment of t^2. Note that the 2-player cut-off criterion does not suffice in this case since the last known cuts of t^1 and t^2 differ in the conditions which are not in the postsets of t^1 and t^2, respectively. ◁

Example 3.39 (Winning 4-player prefix). Another branching process of a winning 4-player prefix is shown in Fig. 3.9. There, for the branching process B_{Sg} exists a winning 4-player prefix τ_{Sg} such that $B_{Sg} = B_{\tau_{Sg}}$ since all possible extensions are preceded by cut-off events due to partial repetitions. Note that $\tau_{Sg}(S_{12}^2) = \emptyset$ and $\tau_{Sg}(S_{22}^2) = \emptyset$ such that B_{Sg} does not continue any further. B_{Sg} does not need to continue any further since any configuration containing a cut-off event r_1^2 or r_2^2 does not need to be winning. ◁

Now, the construction of a winning strategy is presented starting with a winning 4-player prefix. The induction assumption is split into 3 properties: Property 1 ensures that the constructed strategy is winning in every configuration. Property 2 ensures that the constructed strategy is isomorphic after every pair of repetitions in $REP(B_\tau)$. Property 3 states that the strategy is only altered after cut-off events ensuring that the winning properties are preserved for any configuration containing no cut-off event.

The copying process is similar to the copying process in the 2-player case. For each condition b, which is causally preceded by a cut-off event e the strategy is copied from $I_{[R(e)]}^{[e]}(b)$. If b is causally preceded by more than one cut-off event it remains to show that the choice from which cut-off event the strategy is copied always leads to the same strategy. This is a tedious case analysis for concurrent 4-player cut-off events. The Lemma 3.33 is used if exactly one of the cut-off events is due to a partial repetition. If both cut-off events are due to partial repetitions Lemma 3.8 is used. Lemma 3.30 is used if neither of the two concurrent events are cut-off events due to partial repetition. Note that this case only occurs if both events are in each other's synchronisation segment. Here, two of the four tokens participate in one event and the other two tokens participate in the other event. Then, if the events are concurrent and one is not in the other's synchronisation segment the event which is not in the other's synchronisation segment

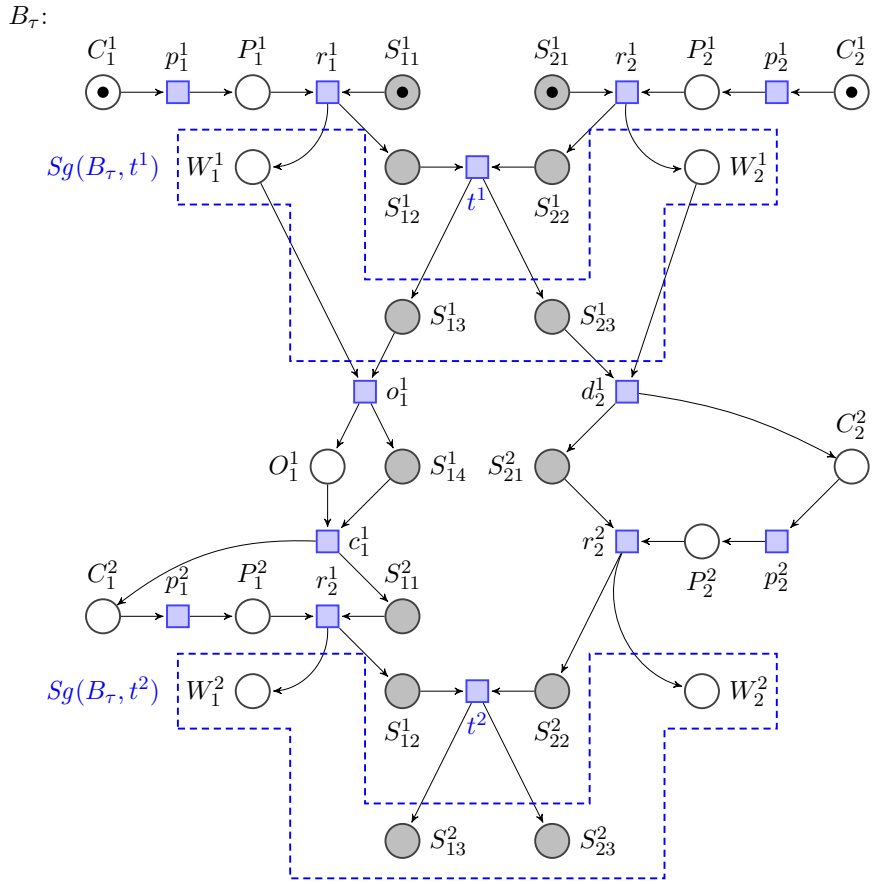


Figure 3.16: A branching process B_τ of a winning 4-player prefix τ of the Petri game G_0 of Fig. 3.1 is shown.

is preceded by a partial repetition by definition of a synchronisation segment (Def. 3.20. Therefore, the preceding partial repetition is used to copy the strategy. Then, this case belongs to the case where one repetition is due to partial repetition which uses Lemma 3.33.

Lemma 3.40 (Continuation of winning 4-player prefix). *If there exists a winning 4-player prefix τ in a 4-player Petri game G there exists a winning strategy in G.*

Proof by induction over the set of finite configurations of u. We construct a winning strategy starting with a winning 4-player prefix τ by induction over the set of finite configurations of the unfolding.

Induction assumption:

1. B_{τ_n} satisfies the winning properties for all configurations C_i, $i \leq n$,
2. for all $(e_1, e_2) \in REP(B_\tau) : \forall b \in \mathcal{P}_{Fut(B_{\tau_n}, [e_2]_{\tau_n})} : ([b]_u \preceq C_n \implies \tau_n(b) = \tau_n(I^{[e_2]}_{[e_1]}(b))$.

3. For all $b \in In_{B_\tau} \cup pre(NCO(B_\tau))$ holds that $\tau_n(b) = \tau(b)$.

Induction base case: The construction starts with $\tau_1 = \tau$. The winning properties hold in $C_1 = \emptyset$, because the cut of C_1 is the initial marking and by definition of a winning 4-player prefix the winning properties hold in the initial marking. The second property holds since for all pairs $(e_1, e_2) \in REP(B_\tau)$ the isomorphism $I^{[e_2]}_{[e_1]}$ maps each condition in the inital marking to itself if that condition is in $\mathcal{P}_{Fut(u,[e_2]_u)}$: if $prc(u, e_1, e_2)$ holds this follows from Lemma 3.5. Otherwise that condition of the initial marking is in the synchronisation segment of e_2 and concurrent to e_2 and the strategy is preserved for to e_2 concurrent conditions under the isomorphism of the synchronisation segment. The third property holds since $\tau_1 = \tau$.

Induction step: Let C_{n+1} be the next configuration of the unfolding regarding $\preceq$.

Case $C_{n+1} \not\subseteq \mathcal{T}_{B_{\tau_n}}$: Here, the configuration C_{n+1} is not a configuration of B_{τ_n}. Therefore, we define $\tau_{n+1} = \tau_n$ and the induction properties hold by induction assumption.

Case $C_{n+1} \subseteq NCO(B_{\tau_n})$: Here, the configuration C_{n+1} does not contain any 4-player cut-off event. Therefore, C_{n+1} is a configuration of the starting prefix B_τ. We define $\tau_{n+1} = \tau_n$ and the first induction property holds by definition of a winning 4-player prefix and the third induction property holds by induction assumption.

For all conditions b with $[b]_u = C_{n+1}$ and all pairs $(e_1, e_2) \in REP(B_\tau)$ such that $prc(B_\tau, e_1, e_2)$: if $b \in Fut(B_\tau, [e_2]_\tau)$ it holds that $b||e_2$. (Otherwise, b is preceded by a cut-off event which contradicts the assumption.) Then, property 2 follows by Lemma 3.5. If $b \in Fut(B_\tau, [e_2]_\tau)$ for a pair $(e_1, e_2) \in REP(B_\tau)$ such that the properties 1-3 of Def. 3.24 hold induction property 2 follows since the strategy is preserved for all to e_2 concurrent conditions by the isomorphism on the synchronisation segments. (Otherwise, b is preceded by a cut-off event which contradicts the assumption.) For all other conditions, the induction properties 2 and 3 hold by induction assumption.

Case $C_{n+1} \subseteq \bigcup_{1,\ldots,n} C_n$: All events of the configuration C_{n+1} are in a previous configuration. We define $\tau_{n+1} = \tau_n$ and the induction properties 2 and 3 hold by induction assumption. Induction property 1 holds since either $C_{n+1} \subseteq NCO(B_{\tau_n})$ holds (as in the previous case) or $Cut(C_{n+1})$ is a cut in the future of an event e_2 such that $(R(e_2), e_2) \in REP(B_\tau)$. Then, $I^{[e_2]}_{[R(e_2)]}(C_{n+1})$ is winning by induction assumption.

Case $C_{n+1} \subseteq \mathcal{T}_{B_{\tau_n}} \wedge \exists e \in C_{n+1} \setminus \bigcup_{1,\ldots,n} C_n : \exists (e_1, e_2) \in REP(B_\tau) : e_2 \leq e$: Let $(e_1, e_2) \in REP(B_\tau)$ be such a pair and e the event of the configuration C_{n+1} that is in no lesser configuration. (This event is unambiguous.) For all $b \in post_u(e)$, the strategy τ_{n+1} is defined as $\tau_{n+1}(b) = \tau_n(I^{[e_2]}_{[e_1]}(b))$ and $\tau_{n+1}(b) = \tau_n(b)$ otherwise.

The second induction property holds for the pair (e_1, e_2) for all $b \in post_u(e)$ by definition of τ_{n+1} and by induction assumption for all other conditions.

Since $[e_1] \prec [e_2]$ and by induction property 2 the cut $I^{[e_2]}_{[e_1]}(Cut(C_{n+1}))$ is winning by induction assumption's property 1 Induction property 3 holds by induction assumption since e is preceded by a cut-off event

It remains to show that the second induction property holds for all other pairs $(e'_1, e'_2) \in REP(B_\tau)$.

Let $(e_1', e_2') \in REP(B_\tau)$ be another pair such that there exists a condition $b \in post_{\tau_n}(e)$ that is a condition in $\mathcal{P}_{Fut(B_{\tau_{n+1}}, [e_2']_{\tau_{n+1}})}$. If there does not exist such a condition b the induction property 2 holds by induction assumption for the pair (e_1', e_2'). The following subcases are distinguished:

Subcase $e_2 \# e_2'$: Since $e_2 \leq b$ it follows that $e_2' \# b$. This is a contradiction to $b \in Fut(B_{\tau_{n+1}}, [e_2']_{\tau_{n+1}})$. Therefore, this case does not occur and the induction property 2 holds for (e_1', e_2') by induction assumption.

Subcase $e_2 || e_2'$, $prc(B_\tau, e_1, e_2)$ and $prc(B_\tau, e_1', e_2')$: By Lemma 3.8, it holds for all conditions $b \in post_u(e)$ that $I^{[e_2]}_{[e_1]}(I^{[e_2']}_{[e_1']}(b)) = I^{[e_2']}_{[e_1']}(I^{[e_2]}_{[e_1]}(b))$. Since $I^{[e_2]}_{[e_1]}(b)$ is an isomorphism and by Lemma 3.7 it follows that ${I^{[e_2]}_{[e_1]}}^{-1}(I^{[e_2']}_{[e_1']}(I^{[e_2]}_{[e_1]}(b))) = I^{[e_2']}_{[e_1']}(b)$. We show that $\tau_{n+1}(b) = \tau_{n+1}(I^{[e_2']}_{[e_1']}(b))$ holds. Therefore, we show for the left side of the equation that $\tau_{n+1}(b) = \tau_{n+1}({I^{[e_2]}_{[e_1]}}^{-1}(I^{[e_2']}_{[e_1']}(I^{[e_2]}_{[e_1]}(b))))$ holds. As defined in the induction step, $\tau_{n+1}(b) = \tau_{n+1}(I^{[e_2]}_{[e_1]}(b))$ holds. By Lemma 3.7, $I^{[e_2']}_{[e_1']}$ can be applied on $I^{[e_2]}_{[e_1]}(b)$. The equality still holds after applying $I^{[e_2']}_{[e_1']}$ by induction assumption. Afterwards, the inverse ${I^{[e_2]}_{[e_1]}}^{-1}$ can be applied by Lemma 3.7 since $e_1 || e_2'$ holds by Lemma 3.6. Then, applying the inverse of $I^{[e_2]}_{[e_1]}$ also preserves the equality by induction assumption since the local configuration of $I^{[e_2']}_{[e_1']}(I^{[e_2]}_{[e_1]}(b))$ is less than C_{n+1} w.r.t. $\preceq$.

Subcase $e_2 || e_2'$, $\neg prc(B_\tau, e_1, e_2)$ and $prc(B_\tau, e_1', e_2')$: The difference to the previous subcase is that $prc(B_\tau, e_1, e_2)$ does not hold. This means that the conditions 1-3 of Def. 3.24 hold for e_1 and e_2.

By Lemma 3.6, $e_2 || e_1'$. By Lemma 3.33, it holds for all conditions $b \in post_u(e)$ that $I^{[e_2]}_{[e_1]}(I^{[e_2']}_{[e_1']}(b)) = I^{I^{[e_2]}_{[e_1]}(e_2')}_{I^{[e_2]}_{[e_1]}(e_1')}(I^{[e_2]}_{[e_1]}(b))$. Since $I^{[e_2]}_{[e_1]}(b)$ is an isomorphism and by Lemma 3.7, it follows that ${I^{[e_2]}_{[e_1]}}^{-1}(I^{I^{[e_2]}_{[e_1]}(e_2')}_{I^{[e_2]}_{[e_1]}(e_1')}(I^{[e_2]}_{[e_1]}(b))) = I^{[e_2']}_{[e_1']}(b)$. We show that $\tau_{n+1}(b) = \tau_{n+1}(I^{[e_2']}_{[e_1']}(b))$ holds. Therefore, we show for the left side of the equation that the strategy τ_{n+1} is preserved, i.e. that $\tau_{n+1}(b) = \tau_{n+1}({I^{[e_2]}_{[e_1]}}^{-1}(I^{I^{[e_2]}_{[e_1]}(e_2')}_{I^{[e_2]}_{[e_1]}(e_1')}(I^{[e_2]}_{[e_1]}(b))))$ holds. By definition of τ_{n+1} it holds that $\tau_{n+1}(b) = \tau_{n+1}(I^{[e_2]}_{[e_1]}(b))$. The equality still holds after applying $I^{I^{[e_2]}_{[e_1]}(e_2')}_{I^{[e_2]}_{[e_1]}(e_1')}$: By Lemma 3.27, it holds that $prc(B_{\tau_{n+1}}, I^{[e_2]}_{[e_1]}(e_1'), I^{[e_2]}_{[e_1]}(e_2'))$. Since $e_1' \in \mathcal{T}_{Sg(B_\tau, e_2)}$, it holds that $I^{[e_2]}_{[e_1]}(e_1') = I^{Sg(B_\tau, e_2)}_{Sg(B_\tau, e_1)}(e_1') \in \mathcal{T}_{Sg(B_\tau, e_1)}$. Therefore, $I^{[e_2]}_{[e_1]}(e_1')$ is not preceded by a partial repetition such that $(I^{[e_2]}_{[e_1]}(e_1'), I^{[e_2]}_{[e_1]}(e_2')) \in REP(B_\tau)$ follows. Thus, the equality holds after applying $I^{I^{[e_2]}_{[e_1]}(e_2')}_{I^{[e_2]}_{[e_1]}(e_1')}$ by induction assumption. Then, applying the inverse of $I^{[e_2]}_{[e_1]}(b)$ also preserves the equality by induction assumption since the local configuration

of $I^{[e'_2]}_{[e'_1]}(b)$ is less than C_{n+1} w.r.t. $\preceq$.

Subcase $e_2||e'_2$, $prc(B_\tau, e_1, e_2)$ and $\neg prc(B_\tau, e'_1, e'_2)$: The proof works similar as in the previous subcase, but τ_{n+1} was defined using $prc(B_\tau, e_1, e_2)$ and it remains to show that the equality $\tau_{n+1}(b) = \tau_{n+1}(I^{[e'_2]}_{[e'_1]}(b))$ holds. By Def. 3.24, since $\neg prc(B_\tau, e'_1, e'_2)$ the conditions 1-3 of that definition hold for the pair (e_1, e_2).

By Lemma 3.6, $e'_2||e_1$ holds. By Lemma 3.33, it holds for all conditions $b \in post_u(e)$ that $I^{[e'_2]}_{[e'_1]}(I^{[e_2]}_{[e_1]}(b)) = I^{I^{[e'_2]}_{[e'_1]}(e_2)}_{I^{[e'_2]}_{[e'_1]}(e_1)}(I^{[e'_2]}_{[e'_1]}(b))$. Since $I^{I^{[e'_2]}_{[e'_1]}(e_2)}_{I^{[e'_2]}_{[e'_1]}(e_1)}$ is an isomorphism and by Lemma 3.7, it follows that $I^{I^{[e'_2]}_{[e'_1]}(e_2)}_{I^{[e'_2]}_{[e'_1]}(e_1)}{}^{-1}(I^{[e'_2]}_{[e'_1]}(I^{[e_2]}_{[e_1]}(b))) = I^{[e'_2]}_{[e'_1]}(b)$. We show that $\tau_{n+1}(b) = \tau_{n+1}(I^{[e'_2]}_{[e'_1]}(b))$ holds. Therefore we show for the left side of the equation that $\tau_{n+1}(b)$ is preserved, i.e. that $\tau_{n+1}(b) = \tau_{n+1}(I^{I^{[e'_2]}_{[e'_1]}(e_2)}_{I^{[e'_2]}_{[e'_1]}(e_1)}{}^{-1}(I^{[e'_2]}_{[e'_1]}(I^{[e_2]}_{[e_1]}(b))))$ holds.

As defined before $\tau_{n+1}(b) = \tau_{n+1}(I^{[e_2]}_{[e_1]}(b))$. The equality still holds after applying $I^{[e'_2]}_{[e'_1]}$ by induction assumption. By Lemma 3.27, $prc(B_{\tau_{n+1}}, I^{[e_2]}_{[e_1]}(e'_1), I^{[e_2]}_{[e_1]}(e'_2))$ holds. By definition of $REP(B_\tau)$, e_1 is minimal w.r.t. $\preceq$ such that $e_1 \in \mathcal{T}_{Sg(B_\tau, e'_2)}$ follows from Lemma 3.6. It follows that $I^{[e'_2]}_{[e'_1]}(e_1) = I^{Sg(B_\tau, e'_2)}_{Sg(B_\tau, e'_1)}(e_1) \in \mathcal{T}_{Sg(B_\tau, e'_1)}$. Therefore, $I^{[e'_2]}_{[e'_1]}(e_1)$ is not preceded by a partial repetition such that $(I^{[e'_2]}_{[e'_1]}(e_1), I^{[e'_2]}_{[e'_1]}(e_2)) \in REP(B_\tau)$ follows. Then, applying the inverse of $I^{I^{[e'_2]}_{[e'_1]}(e_2)}_{I^{[e'_2]}_{[e'_1]}(e_1)}$ also preserves the equality by induction assumption since the local configuration of $I^{[e'_2]}_{[e'_1]}(b)$ is less than C_{n+1} w.r.t. $\preceq$.

Subcase $e_2||e'_2$, $\neg prc(B_\tau, e_1, e_2)$ and $\neg prc(B_\tau, e'_1, e'_2)$: Since neither $prc(B_\tau, e_1, e_2)$ nor $prc(B_\tau, e'_1, e'_2)$ for both pairs the conditions 1-3 of Def. 3.24 holds. Thus, $|pre(e_2)| = |pre(e_1)| = 2$.

We show that $I^{[e'_2]}_{I^{[e_2]}_{[e_1]}(e'_2)}(e_2) = e_1$ such that Lemma 3.30 is applicable.

It holds that $e_2 \in \mathcal{T}_{Sg(B_\tau, e'_2)}$ since $e_2||e'_2$ and $(e_1, e_2) \in REP(B_\tau)$, i.e. e_2 is not causally preceded by a cut-off event. (In particular, it is not preceded by a partial repetition.)

It follows that $I^{[e'_2]}_{I^{[e_2]}_{[e_1]}(e'_2)}(e_2) = I^{Sg(B_\tau, e'_2)}_{Sg(B_\tau, I^{[e_2]}_{[e_1]}(e'_2))}(e_2) \in \mathcal{T}_{Sg(B_\tau, I^{[e_2]}_{[e_1]}(e'_2))}$, since e'_2 and $I^{[e_2]}_{[e_1]}(e'_2))$ are synchronisation equivalent by definition of $sqc(B_\tau, e_1, e_2)$.

Now, we show that $e_1 \in \mathcal{T}_{Sg(B_\tau, I^{[e_2]}_{[e_1]}(e'_2))}$. By Lemma 2.18, the isomorphism $I^{[e_2]}_{[e_1]}$ preserves concurrency. Since $e_2||e'_2$ it holds that e'_2 is concurrent to all conditions in $post_\tau(e_2)$. Thus, $I^{[e_2]}_{[e_1]}(e'_2)$ is concurrent to all conditions in $post_\tau(e_1)$ since it holds that $I^{[e_2]}_{[e_1]}(post_\tau(e_2)) = post_\tau(e_1)$. Therefore, $I^{[e_2]}_{[e_1]}(e'_2)||e_1$ holds.

By definition of $REP(B_\tau)$, e_1 is not preceded by a cut-off event, i.e. it is not preceded

by a partial repetition in particular. Thus, $e_1 \in \mathcal{T}_{Sg(B_\tau, I^{[e_2]}_{[e_1]}(e'_2))}$ holds.

Since e_2 and e_1 have the same labels in their postset and $I^{[e'_2]}_{I^{[e_2]}_{[e_1]}(e'_2)}(e_2)$ and e_2 have the same labels in their postsets it follows that e_1 and $I^{[e'_2]}_{I^{[e_2]}_{[e_1]}(e'_2)}(e_2)$ have the same labels in their postsets. By Lemma 3.26, $lkc_\tau(e_1) \setminus post_\tau(e_1) = lkc_\tau(I^{[e'_2]}_{I^{[e_2]}_{[e_1]}(e'_2)}(e_2)) \setminus post_\tau(I^{[e'_2]}_{I^{[e_2]}_{[e_1]}(e'_2)}(e_2))$. It follows that $prc(B_\tau, e_1, I^{[e'_2]}_{I^{[e_2]}_{[e_1]}(e'_2)}(e_2))$. Now, $e_1 \neq I^{[e'_2]}_{I^{[e_2]}_{[e_1]}(e'_2)}(e_2)$ implies that either e_1 or $I^{[e'_2]}_{I^{[e_2]}_{[e_1]}(e'_2)}(e_2)$ is not in $\mathcal{T}_{Sg(B_\tau, I^{[e_2]}_{[e_1]}(e'_2))}$ which is a contradiction to $e_1 \in \mathcal{T}_{Sg(B_\tau, I^{[e_2]}_{[e_1]}(e'_2))}$ and $I^{[e'_2]}_{I^{[e_2]}_{[e_1]}(e'_2)}(e_2) \in \mathcal{T}_{Sg(B_\tau, I^{[e_2]}_{[e_1]}(e'_2))}$. Thus, $I^{[e'_2]}_{I^{[e_2]}_{[e_1]}(e'_2)}(e_2) = e_1$.

By Lemma 3.30, $sqc(B_\tau, e'_2, I^{[e_2]}_{[e_1]}(e'_2)))$ holds. We show that $I^{[e_2]}_{[e_1]}(e'_2) = e'_1$ to apply Lemma 3.31.

Since synchronisation equivalence is transitive it holds that $sqc(B_\tau, I^{[e_2]}_{[e_1]}(e'_2), e'_1)$. If $I^{[e_2]}_{[e_1]}(e'_2)) \neq e'_1$ we show that $sqc(B_\tau, e_1, I^{I^{[e_2]}_{[e_1]}(e'_2))}_{e'_1}(e_1))$. The proof works analogously to the proof that $sqc(B_\tau, e'_2, I^{[e_2]}_{[e_1]}(e'_2)))$ holds substituting e'_2 with e_1 and $I^{[e_2]}_{[e_1]}(e'_2)$ with $I^{I^{[e_2]}_{[e_1]}(e'_2))}_{e'_1}(e_1)$. For this, we compare if all prerequisites used to show $sqc(B_\tau, e'_2, I^{[e_2]}_{[e_1]}(e'_2)))$ also hold for e_1 and $I^{I^{[e_2]}_{[e_1]}(e'_2))}_{e'_1}(e_1)$: One prerequisite used is that $sqc(B_\tau, e_1, e_2)$. This holds analogously since $sqc(B_\tau, e'_1, I^{[e_2]}_{[e_1]}(e'_2))$. Another prerequisite used is that $(e_1, e_2) \in REP(B_\tau)$. Since $(e'_1, e'_2) \in REP(B_\tau)$ it holds that e'_1 is the minimal event in its synchronisation equivalence class, i.e. analogously $(e'_1, I^{[e_2]}_{[e_1]}(e'_2)) \in REP(B_\tau)$ holds since $I^{[e_2]}_{[e_1]}(e'_2)$ is not preceded by a partial repetition because it is in the synchronisation segment of e_1. We still need to show that $I^{[e_2]}_{[e_1]}(e'_2)$ is not preceded by a 4-player cut-off event because $I^{[e_2]}_{[e_1]}(e'_2) \in \mathcal{T}_\tau$ must hold to get $(e'_1, I^{[e_2]}_{[e_1]}(e'_2))REP(B_\tau)$. If $I^{[e_2]}_{[e_1]}(e'_2)$ is preceded by a 4-player cut-off event f it holds that f is in the synchronisation segment of e_1 since e_1 is not preceded by a 4-player cut-off event. Therefore, ${I^{[e_2]}_{[e_1]}}^{-1}(f)$ precedes e'_2. By Lemma 3.30 it follows that f and ${I^{[e_2]}_{[e_1]}}^{-1}(f)$ are synchronisation equivalent. Thus, e'_2 is also preceded by a 4-player cut-off event which is a contradiction to $e'_2 \in \mathcal{T}_\tau$ since $(e'_1, e'_2) \in REP(B_\tau)$. Therefore $I^{[e_2]}_{[e_1]}(e'_2) \in \mathcal{T}_\tau$ holds and $(e'_1, I^{[e_2]}_{[e_1]}(e'_2))$ follows. The remaining prerequisites are the following. The cardinality of the presets of e'_2, e_1, $I^{[e_2]}_{[e_1]}(e'_2)$ and $I^{I^{[e_2]}_{[e_1]}(e'_2))}_{e'_1}(e_1)$ is always 2. The prerequisite that e_2 and e'_2 are concurrent holds analogously for e_1 and $I^{[e_2]}_{[e_1]}(e'_2)$ by Lemma 2.18 since $I^{[e_2]}_{[e_1]}(post(e_1)) = post(e_2)$. Therefore, all prerequisite are satisfied analogously.

Since e'_1 is the event which is minimal w.r.t. $\preceq$ in its synchronisation equivalence class,

especially lesser than $I^{[e_2]}_{[e_1]}(e'_2)$, it follows that $I^{I^{[e_2]}_{[e_1]}(e'_2))}_{e'_1}(e_1) \prec e_1$. This is a contradiction to e_1 being the event which is minimal w.r.t. $\preceq$ in its synchronisation equivalence class. Thus, $I^{[e_2]}_{[e_1]}(e'_2)) = e'_1$.

By Lemma 3.31, $I^{[e_2]}_{[e_1]}(b) = I^{[e'_2]}_{[e'_1]}(b)$ and $\tau_{n+1}(b) = \tau_{n+1}(I^{[e'_2]}_{[e'_1]}(b))$ follows and induction assumption 2 holds for the pair (e'_1, e'_2). □

3.3.3 Solving 4-player Petri Games

In the following, the nondeterministic algorithm that constructs a winning 4-player prefix is presented. Afterwards, the complexity of the algorithm is investigated. The algorithm is similar to Algorithm 1 constructing a winning 2-player prefix. Again, the algorithm is similar to the algorithm constructing a complete finite prefix (see. [ERV02]). However, we cut off 4-player cut-off events (Def. 3.34) while in [ERV02] an event is a cut-off event if there exists a lesser event (w.r.t. $\preceq$) with the same last known marking. Completeness is in the sense of reachability of a net, i.e. for every reachable marking in the net there exists a cut in the complete prefix such that the marking of the cut is the reachable marking of the net. A finite prefix is guessed until all possible extensions (Def. 3.17) are preceded by 4-player cut-off events (Def. 3.34) or are not allowed by the guessed strategy.

Keeping the check whether an event e is synchronisation equivalent to another event consistent requires the algorithm to define the whole synchronisation segment of e as soon as e is added to the prefix if it has at least two preconditions. Also, the synchronisation segments of all events with at least two preconditions in the synchronisation segment of e must be added when e is added. The synchronisation segments must not be altered afterwards. Therefore, the strategy carries a flag for every condition stating if the strategy has already been guessed for that condition. Thus, the strategy τ is a mapping $\mathcal{P}_u \to 2^{\mathcal{T}} \times \{0, 1\}$ in Algorithm 3 where the second component states that the strategy has not been guessed yet if it is 0. The instruction *guess* $\tau(b)$ assigns a pair $(S, 1)$ to $\tau(b)$ where S is a random subset of $h_u(post_u(b))$ if the second component of $\tau(b)$ is 0 beforehand. Otherwise *guess* $\tau(b)$ does not change $\tau(b)$. If b is an environment condition the subset S of $h_u(post_u(b))$ is the whole set $h_u(post_u(b))$.

The subroutine guessing a synchronisation equivalence class is shown in Subroutine 2. There, a procedure *GuessSegment* and a procedure *GuessSynchronisation* is defined. The procedure *GuessSegment* constructs a synchronisation segment of an event e' similar to the unfolding algorithm: starting with the last known cut of e' events are added to the synchronisation segment until the conditions in Def. 3.20 are met. In particular, the cut-off criterion used in line 10 is the 2-player cut-off criterion which is partial repetition. Note that the last condition checked in line 8 ensures that only events are added which are concurrent to e'.

After adding an event e in line 12 of Algorithm 3, its synchronisation equivalence class is determined by the function call *GuessSynchronisation(τ, e)* in line 13. The function call *GuessSynchronisation(τ, e)* guesses the strategy for all conditions in the synchronisation

Subroutine 2: Subroutine guessing a synchronisation equivalence class.

```
1  procedure GuessSegment(τ, e′):
2      B_segment := (lkc_u(e′), ∅, ∅, ∅, lkc_u(e′));
3      PE := PE(B_segment), cutoff := ∅, forbidden := ∅;
4      forall b ∈ post(e′) do
5          guess (τ(b));
6      while PE ≠ ∅ do
7          choose event e ∈ PE which is mimimal with respect to ⪯;
8          if [e]_u ∩ cutoff = ∅ ∧ ∀b ∈ pre(e) : h_u(e) ∈ τ(b) ∧ e||e′ then
9              addEvent (B_segment, e);
10             if e is 2-player cut-off event in B_τ then
11                 cutoff := cutoff ∪ {e};
               else
12                 forall b ∈ post(e) do
13                     guess (τ(b));
                   /* Generating possible extensions for new B_segment.     */
14                 PE := PE(B_segment) \ forbidden;
           else
15             PE := PE \ {e};
16             forbidden := forbidden ∪ {e};
17     return;
18 procedure GuessSynchronisation(τ, e′):
19     if |pre(e′)| ≥ 2 then
20         GuessSegment(τ, e′);
21         if ∃e ∈ Sg(B_τ, e′) : |pre(e)| = 2 then
22             GuessSegment(τ, e);
       else
23         forall b ∈ post(e′) do
24             guess (τ(b));
25     return;
```

Algorithm 3: NEXP-algorithm constructing a winning 4-player prefix

Input: Token-partitioning 4-player Petri game G_0
Output: Strategy τ, and "*yes*" or "*no*" stating if τ is a winning 4-player prefix

forall $b \in \mathcal{P}_u$ **do**
  $\tau(b) := (\emptyset, 0)$
forall $b \in In_u^S$ **do**
  `guess` $(\tau(b))$
forall $b \in In_u^E$ **do**
  $\tau(b) := (h_u(post_u(b)), 1)$
$B_{prefix} := (In_u, \emptyset, \emptyset, \emptyset, In_u)$;
$PE := PE(B_{prefix})$, $cutoff := \emptyset$, $forbidden := \emptyset$;
while $PE \neq \emptyset$ **do**
  choose event $e \in PE$ which is mimimal with respect to $\preceq$;
  if $[e]_u \cap cutoff = \emptyset \wedge \forall b \in pre(e) : h_u(e) \in \tau(b)$ **then**
    `addEvent` (B_{prefix}, e);
    `/* Guessing synchronisation class of added event. */`
    `GuessSynchronisation` (τ, e);
    if *e is 4-player cut-off event in* B_τ **then**
      $cutoff := cutoff \cup \{e\}$;
    else
      `/* Generating possible extensions for new` B_{prefix}`. */`
      $PE := PE(B_{prefix}) \setminus forbidden$;
  else
    $PE := PE \setminus \{e\}$;
    $forbidden := forbidden \cup \{e\}$;
`/* Check if guessed prefix is a winning 4-player prefix. */`
output := τ, "*yes*"
forall *For all configurations* $C \subseteq \mathcal{T}_\tau \setminus cutoff$ **do**
  if *A winning condition does not hold in* C **then**
    output := τ, "*no*";

segment of the added event. If an event has only one precondition the strategy is guessed only for the conditions in the postset of that event (line 23-24 Subroutine 2). If there is an event with at least two preconditions in the synchronisation segment of the added event its synchronisation segment is also guessed (line 21-22 Subroutine 2). Remember that the strategy once guessed cannot be altered afterwards since the strategy is extended by a flag for every condition if the strategy is already defined. This way, the output of Algorithm 3 is a winning 4-player prefix if the check in lines 19-21 succeeds.

Lemma 3.41 (Correctness Algorithm 3). *If Algorithm 3 outputs a strategy τ and "yes", τ is a winning 4-player prefix given a token-partitioning 4-player Petri game G_0.*

Proof. Lines 19-21 of Algorithm 3 check if all configurations $C \subseteq NCO(B_\tau)$ are winning. So, if this check succeeds for all configurations the property of Def. 3.37 is satisfied. □

If there exists a winning strategy in a token-partitioning 4-player Petri game we show that there exists an execution of Algorithm 3 that outputs a winning prefix.

Lemma 3.42 (Completeness Algorithm 3). *Given a token-partitioning 4-player Petri game G_0, if there exists a winning strategy in G_0 there exists an execution of Algorithm 3 that outputs a winning prefix.*

Proof. Algorithm 3 guesses the strategy for the condition in the initial marking and all postconditions of events until no more events are allowed by the guessed strategy or they are preceded by 4-player cut-off events. So, the execution of Algorithm 3 where the guessed prefix coincides with the given winning strategy for all conditions which are not preceded by a 4-player cut-off event is a winning prefix. □

Now, the complexity of Algorithm 3 is investigated. Therefore, the maximal size of a synchronisation segment is determined. Afterwards, it is shown that there are at most twice as many synchronisation equivalence classes in a token-partitioning 4-player Petri game. We consider the size of any net N to be $|\mathcal{T} \cup \mathcal{P}|$.

Lemma 3.43 (Size of synchronisation segment). *Let N_0 be a token-partitioning net with at most 4 tokens, B a branching process of N_0 and $e \in \mathcal{T}$ such that $|pre(e)| \geq 2$. The size of the synchronisation segment $Sg(B, e)$ is quadratic in the size of the net N_0.*

Proof. The synchronisation segment $Sg(B, e)$ consists of its initial marking, which is the last known cut of the event. All events in $\mathcal{T}_{Sg(B,e)}$ are concurrent to e. Thus, there is no event which is a causal successor of e, i.e. the preset of every event in the synchronisation segment is a subset of conditions of tokens which are not participating in e. Those are at most two tokens since there are at most 4 tokens and $|pre(e)| \geq 2$.

From Lemma 3.26 it follows that, if those two tokens do not take any joint event after at most $|\mathcal{T}_0|$-many events both tokens reach partial repetition cuts, i.e. the synchronisation segment ends. Otherwise, the two tokens take a joint event before that bound. Then again, after at most $|\mathcal{T}_0|$-many the two tokens take a joint event or the synchronisation segments ends due to partial repetitions. After at most $|\mathcal{T}_0|$-many joint events, a

label of a joint event repeats. From Lemma 3.26 it follows that the joint event with the repeated label is a partial repetition. Thus, a synchronisation segments contains at most $|\mathcal{T}_0|^2$-many events. Thus, the number of conditions is also quadratically bounded which results in a quadratic size of the underlying net N_0. □

We continue to show that there are at most quadratically many synchronisation equivalence classes as there are synchronisation segments.

Lemma 3.44 (Number of synchronisation equivalence classes). *Let N_0 be a token-partitioning net with partition P such that $|I_P| = 4$ of a game G_0, τ a strategy in G_0 and $e \in \mathcal{T}_\tau$ such that $|pre(e)| \geq 2$. Then, there are at most exponentially many synchronisation equivalence classes for e.*

Proof. By Lemma 3.43, the number of events of a synchronisation segment is quadratically bounded in the number of transitions of the underlying net N_0. By definition of an adequate order (Def. 2.12), $\preceq$ is preserved by finite extension. This means in particular, that for two events that have the same last known marking the adequate order is preserved by finite extensions.

In the following, let $e, e' \in \mathcal{T}_\tau$ such that $|pre(e)| = 2$, $|pre(e')| = 2$. We refer to the two tokens that do not participate in e and e' as token i and token j. Let $M_i(e)$ denote the last known marking of the condition of token i in the last known cut of e and let $M_j(e)$ denote analogously the last known marking of token j of the condition of token j in the last known cut of e.

We show that a mapping $\sigma_{(lkm_\tau(e),M_i(e),M_j(e))} : \mathcal{P}_{Sg(B_\tau,e)} \cap (P_i \cup P_j) \to 2^{\mathcal{T}_0}$ dependent on $lkm_\tau(e)$, $M_i(e)$ and $M_j(e)$ describes the synchronisation segment $Sg(B_\tau, e)$ unambiguously. For all $p \in \mathcal{P}_i \cup \mathcal{P}_j$ we define $\sigma_{(lkm_\tau(e),M_i(e),M_j(e))}(p) = \tau(b)$ if $b \in \mathcal{P}_{Sg(B_\tau,e)}$ and $h_\tau(b) = p$. If there exists no such $b \in \mathcal{P}_{Sg(B_\tau,e)}$ we define $\sigma_{(lkm_\tau(e),M_i(e),M_j(e))}(b) = \emptyset$.

We continue by showing that $e, e' \in \mathcal{T}_\tau$ have equivalent synchronisation segments if $\sigma_{(lkm_\tau(e),M_i(e),M_j(e))} = \sigma_{(lkm_\tau(e'),M_i(e),M_j(e))}$ holds and for the triple of last known markings $(lkm_\tau(e), M_i(e), M_j(e)) = (lkm_\tau(e'), M_i(e'), M_j(e'))$ holds.

Since the strategies $\sigma_{(lkm_\tau(e),M_i(e),M_j(e))} = \sigma_{(lkm_\tau(e'),M_i(e),M_j(e))}$ are the same and the adequate order is preserved by finite extension for conditions with the same last known marking the synchronisation segments of e e' are restricted by partial repetitions isomorphically. Therefore, the synchronisation segments of e and e' are equivalent. Since the mapping $\sigma_{(lkm_\tau(e),M_i(e),M_j(e))}$ together with the last known markings $lkm_\tau(e)$, $M_i(e)$ and $M_j(e)$ has polynomial size, there are at most exponentially different segment equivalence classes.

Now, let $e, e' \in \mathcal{T}_\tau$ be two events such that $sqc(B, e, e')$ holds. For the synchronisation equivalence class, another segment equivalence class must coincide, namely the segment equivalence class of joint events in the synchronisation segment e to satisfy property 3 of Def. 3.24. From Lemma 3.26 and Lemma 3.28 follows that any two event $e_1, e_2 \in \mathcal{T}_{Sg(B,e)}$ have the same set of events in their synchronisation segment. Therefore, the synchronisation segments of e_1 and e_2 only differ in their postsets. Thus, if $lkm_\tau(e_2) = lkm(e_1)$ the events e_1 and e_2 are segment equivalent. Analogous holds for $I^{[e]}_{[e']}(e_2)$ and

$I^{[e]}_{[e']}(e_1)$. Thus, segment equivalence of e_1 and $I^{[e]}_{[e']}(e_1)$ implies segment equivalence of e_2 and $I^{[e]}_{[e']}(e_2)$. Therefore, there are quadratically many synchronisation equivalence classes as there are synchronisation segment equivalence classes.

Overall, there are quadratically many synchronisation equivalence classes as there are synchronisation segments. Thus, there are exponentially many synchronisation equivalence classes. □

The size of a winning prefix τ is the number of nodes that are not preceded by a 4-player cut-off event. We show that a winning prefix contains at most exponentially many nodes.

Lemma 3.45 (Size of winning 4-player prefix). *The number of non-cut-off events of a winning 4-player prefix is exponentially bounded.*

Proof. Each player reaches a partial repetition after at most $|\mathcal{T}_0|$-many events without a joint event. So, after at most $4 \cdot |\mathcal{T}_0|$-many events of the winning prefix a joint event occurs or all further events are cut-off events. This joint event has at least two preconditions and by Lemma 3.44 there are at most exponentially many different synchronisation equivalence classes. Thus, after at most as many joint events as there are synchronisation equivalence classes which are non-cut-off events every joint event is a 4-player cut-off event. Hence, the number of non-cut-off events of a winning 4-player prefix is exponentially bounded. □

This section concludes with the theorem that token-partitioning 4-player Petri games can be solved within nondeterministic exponential time (NEXP).

Theorem 3.46 (Solving 4-player Petri games). *The existence of a winning strategy of a token-partitioning 4-player Petri game G_0 is decidable in NEXP.*

Proof. If there exists a winning strategy τ in G_0 there exists a winning 4-player prefix in G_0 since τ is a winning 4-player prefix by definition.

By Lemma 3.40, a winning strategy exists if a winning 4-player prefix exists.

By Lemma 3.45, the winning 4-player prefix yielded by Algorithm 3 has at most exponential size in the size of the underlying net N_0. Since there are 4 tokens in G_0 there are polynomially many configurations in the size of the prefix that are checked if they are winning. Thus, the time complexity remains single exponential. Since Algorithm 3 is nondeterministic the resulting complexity is NEXP. □

3.4 Related Work

The presented decidability results are an adaptation of the construction of a complete finite prefix of a Petri net [ERV02]. We adjusted the cut-off criterion such that the new cut-off criteria for 2- and 4-player Petri games are not only sufficient for completeness in the sense of reachability as in [ERV02] but also complete in a sense of the behaviour of a winning strategy.

The decidability result of the synthesis problem for 4-player Petri games is related to the decidability result of control games with 4 processes equipped with a local termination condition [Gim17]. According to [BFH19], control games can be translated to Petri games and vice versa, e.g. the decidability result for control games [Gim17] is transferable to Petri games. The translation comes with an exponential overhead in the size of the game. However, the number of processes in the control game coincides with the number of tokens in the Petri game. This equivalence is kept in both directions of the translation. In [Gim17], no complexity bound is given. The translation to Petri games with a local termination condition already requires an exponential overhead in the number of nodes, i.e. states of the processes and actions. Furthermore, a global winning condition is expected to increase the complexity of the synthesis problem. For example, generalizing the decidability result for a local safety condition with a single environment player and a bounded number of system players [FO17] to a global safety condition [FGHO22] raises the complexity from EXPTIME-complete to 2-EXP. However, in [FGHO22] no matching lower bound was found.

While being already undecidable in simple concurrent architectures for synchronous distributed systems [PR90] there have been a few positive decidability results originally achieved in the setting of control games when restricting the architecture [GGMW13, GSZ09, PR90] or restricting the concurrent behaviour [MT02, MTY05]. The complexity solving the synthesis problem for pipeline architectures with a temporal winning condition has non-elementary complexity [PR90]. In [GGMW13], the complexity is a tower of exponential function with the same height as the height of the tree architecture. In [MT02], the synthesis problem is shown to be decidable if the processes have an acyclic communication graph. There, the winning condition is expressed in LTL. The synthesis problem has non-elementary complexity in that setting. In [MTY05], the concurrent behaviour is restricted in a way such that every process communicates with every other process within a bound n or never communicates with that process again. The winning condition is expressed in monadic second order logic over the unfolding. The complexity is also non-elementary. Overall, many of the previous results have non-elementary complexity while the presented result has a far lesser complexity, namely NEXP complexity.

Another line of work is presented in [ABP21] involving an uncontrollable environment player and a controllable system player. Unlike Petri games, where each token represents an individual player, these games define the two players through disjoint sets of transitions—uncontrollable for the environment and controllable for the system. Both players control multiple tokens to allow their respective transitions. As for Petri games, the unfolding of the Petri net is used to define a strategy for the system player. However, this setting does not stick to a causal memory model which is characteristic for Petri games as presented here. In this setting, a strategy is defined as a mapping from cuts in the unfolding to sets of transitions that the system can choose from for its next step. Thus, the information based on which the system makes is decision may also use information which is not from a causal source. Similar to the presented decidability results, the algorithm solving these games also adapts the construction of a complete finite prefix in [ERV02]. The winning condition is a local safety condition, i.e. certain bad places

must be avoided.

3.5 Outlook

In this chapter, we have shown that token-partitioning 4-player Petri Games with a global safety condition can be solved within nondeterministic exponential time and 2-player Petri games belong to the class of NP-complete problems. An efficient deterministic algorithm solving 4-player (and 2-player) Petri games remains future work. Looking at the parametrized benchmark families of Petri games in Chapter 5, this decidability result extends the set of Petri games for which the existence of a winning strategy can be determined. However, for the benchmark families this affects only the first Petri game of the *Production Line* benchmark family. Finding more applications for 4-player Petri games equipped with the presented global safety condition is desirable.

Looking at Petri games with different winning conditions, the decidability result presented here does not have many implications. Obviously, a local safety condition is subsumed. However, the decidability result for 4-player Petri games under a global safety condition is not transferable to Petri games with a global termination condition, which seems like the dual case referring to the duality of reachability and safety 2-player games on finite graphs. Informally speaking, the duality relation does not hold for Petri games because the winning conditions besides safety are asymmetric regarding the system and environment players whereas for games on finite graphs there is a symmetry. We think that it is more likely that 4-player Petri games with a global termination condition are decidable than undecidable. The results in [BFH19] and [Gim17] imply that 4-player Petri games with a local termination condition are decidable. There are no implications for more complex winning conditions such as parity conditions or conditions expressed in LTL.

The synthesis Problem for 5-player Petri games with the global safety condition remains open. The main reason the presented approach does not work for 5-player Petri games is that the size of a synchronisation segment of an event with two participating players is not bounded in the 5-player case. The remaining other three players might never reach partial repetitions or get to know of that event.

In [HO22], we have shown that the synthesis Problem for Petri games with an arbitrary number of system and environment players and where the underlying net satisfies a synchronisation condition is decidable within single exponential time. That result is largely subsumed by the results presented here. The synchronisation condition in [HO22] states that for every node the number of events which are concurrent to that node is bounded by a constant n. This implies that the size of a synchronisation segment as presented here (Def. 3.20) of such a net is linearly bounded by n. The reduction in [HO22] to a 2-player safety game on a finite graphs yields a complexity bound in deterministic single exponential time opposed to nondeterministic single exponential time if we adapt the construction presented here.

A 2-player safety graph game is a game played on a finite directed graph where the set of vertices is disjunct into a set of system and a set of environment places. A token

starts on the initial vertex. On a system vertex, the system player chooses an outgoing edge to which vertex the token moves and on an environment vertex, the environment player chooses the next vertex via an outgoing edge. It is required that at least one outgoing edge exists in every vertex. Both player have complete information about the game state which is solely dependent on the current vertex.

We suspect that a reduction of 4-player Petri games to 2-player safety games on finite graphs might yield a deterministic single exponential time complexity. The states in such a graph game of a Petri game would refer to the synchronisation equivalence classes of the Petri game. However, the reduction is not trivial since the players in the graph game might have more information about which concurrent events have been fired than they would have in the Petri game.

In the next chapter, Chapter 4, it is shown that 6-player Petri games with the global safety condition are undecidable.

Chapter 4

Undecidability Results

In this chapter, we show that the synthesis problem for 6-player Petri games equipped with a global safety condition is undecidable. To this end, a suitable tiling problem is reduced to the synthesis problem of a 6-player Petri game. The tiling problem used here considers an infinite plane where a colour must be assigned to every tile such that neighbouring tiles satisfy given colouring constraints. We first show that the tiling problem is undecidable by a reduction to the ω-Post correspondence problem (ω-PCP) which is undecidable [Fin15b, Gir86, Ruo85].

Later, it can be seen that 6-player Petri games with a *local* safety winning condition are also undecidable. This required only a slight modification of the reduction to the ω-PCP of 6-player Petri games with a global safety winning condition.

The reduction is similar to the work in [Gim22], where the (normal) Post correspondence problem is reduced to a colouring problem followed by a reduction to the synthesis problem of 6-process control games with local termination as a winning condition.

Employing the translation of control games into Petri games in [BFH19], the 6-process control games yield 6-player Petri games (with exponentially many additional nodes) for local termination as a winning condition. In the obtained Petri game, all players alternate in their roles as environment and system players. Instead, we present a direct construction of 6-player Petri games with at most 3 simultaneous system players. Later, we reason that the number of system players can be even further reduced to 2. However, by doing so the number of total player increases to 8.

4.1 ω-3-Directional Tiling Problem

In this section, we introduce the ω-PCP and an infinite tiling problem. We reduce the undecidable ω-PCP to the tiling Problem. This means, that the tiling problem is also undecidable. In the next section, we reduce 6-player Petri games equipped with a global safety condition to that tiling problem showing that such Petri games are undecidable.

Definition 4.1 (ω-Post correspondence problem (ω-PCP)). Let Σ be an alphabet, and let $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$ be n-tuples of non-empty words in Σ^*. The ω-

PCP asks whether there exists an infinite sequence of indices $i_1, i_2, \ldots$, $(\forall j \in \mathbb{N} : i_j \in \{1, \ldots, n\})$, such that the infinite words $x_{i_1}x_{i_2}\ldots$ and $y_{i_1}y_{i_2}\ldots$ over Σ are identical, i.e. $x_{i_1}x_{i_2}\ldots = y_{i_1}y_{i_2}\ldots$. We say the subwords x_{i_j} and y_{i_k} belong to the *same pair* if $j = k$. ◁

In [Fin15b, Gir86, Ruo85], it is shown that the ω-PCP is undecidable. Now, we introduce a tiling problem for an infinite plain where the plain is represented by the set $\mathbb{N}^2$ ($\mathbb{N} = \{1, 2, \ldots\}$) and every pair $(x, y) \in \mathbb{N}^2$ must be assigned a colour from a finite set of colours. The problem is called the *ω-3-directional tiling problem* because the colouring constraints affect three directions: horizontal, vertical and diagonal.

Definition 4.2 (ω-tiling). Let Cl be a finite set of colours. An *ω-tiling* is a mapping $f : \mathbb{N}^2 \mapsto Cl$. The initial colour is $f(1, 1)$. ◁

Before defining the colouring constraints we define the patterns which are subject to the colouring constraints.

Definition 4.3 (Horizontal, vertical and diagonal patterns). Let f be an ω-tiling, $f : \mathbb{N}^2 \mapsto Cl$. We define three subsets of Cl^2 called the *patterns* induced by f:

- the *diagonal patterns* of f are all pairs $\{(f(x, y), f(x + 1, y + 1)) \mid (x, y) \in \mathbb{N}^2)\}$
- the *horizontal patterns* of f are all pairs $\{(f(x, y), f(x + 1, y)) \mid (x, y) \in \mathbb{N}^2)\}$
- the *vertical patterns* of f are all pairs $\{(f(x, y), f(x, y + 1)) \mid (x, y) \in \mathbb{N}^2)\}$

◁

The definition of the colouring constraints follows.

Definition 4.4 (ω-3-Directional colouring constraint). For a finite set of colours Cl, let f be an ω-tiling, $f : \mathbb{N}^2 \mapsto Cl$. A *colouring constraint* is a 4-tuple (Cl_i, DP, HP, VP), where $Cl_i \subseteq Cl$ is the set of *initial* colours, $DP \subseteq Cl^2$ a set of diagonal constraints, $VP \subseteq Cl^2$ a set of vertical constraints and $HP \subseteq Cl^2$ a set of horizontal constraints. DP, VP and HP are called *forbidden patterns*. ◁

The ω-3-directional tiling problem asks for a given ω-3-directional colouring constraints if there exists an ω-tiling such that no forbidden pattern occurs.

Definition 4.5 (ω-3-directional tiling problem (ω-3-DTP)). Given a finite set of colours Cl and an ω-3-directional colouring constraint (Cl_i, DP, HP, VP) based on Cl, the ω-3-DTP asks whether there exists a an ω-tiling f such that its initial colour is in Cl_i, no diagonal pattern of f is in DP, no vertical pattern of f is in VP and no horizontal pattern of f is in HP. An *instance* TP of the ω-3-DTP is denoted as a pair $(Cl, (Cl_i, DP, HP, VP))$. ◁

We reduce the ω-PCP to the ω-3-DTP which shows that the ω-3-DTP is undecidable.

Theorem 4.6 (ω-3-DTP is undecidable). *There exists a reduction of the ω-PCP to the ω-3-DTP.*

Proof. Characterization of an ω-PCP solution. Let $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$ be an instance of the ω-PCP with the alphabet Σ. There exists an infinite sequence of indices $i_1, i_2, \ldots$ with $x = x_{i_1}x_{i_2}\ldots = y_{i_1}y_{i_2}\ldots = y$ if and only if there exists an ω-tiling which satisfies the following colouring constraints.

The set of colours and the colouring constraints are defined as follows: Let Q_x be the finite subset of $\Sigma \times \{1, \ldots, n\} \times \mathbb{N}$ containing all (a, i, m) where $m \leq |x_i|$ and a is the m-th letter of x_i. Here, $|x_i|$ denotes the length of x_i, i.e. the number letters in x_i. For any word $w \in \sigma^*$, we use $|w|$ to denote the length of w. The letter a is called a first letter if $m = 1$ and called maximal if $m = |x_i|$. We define Q_y in the same way replacing the x_i with y_i. The set of colours Cl is defined as $Q_x \times Q_y \times \{\mathit{SameLetter}, -\} \times \{\mathit{SamePair}, -\}$, where all colours are removed that have different letters or different tile indices if the *SameLetter* flag or the *SamePair* flag is set, respectively.

The set of initial colours Cl_i is the set of colours where both letters are first letters and the flags *SameLetter* and *SamePair* are set. In the following we use the symbol $*$ as a placeholder for any other valid symbol.

1. The **set of diagonal constraints** contains all $((*, *, \mathit{SameLetter}, *), (*, *, -, *))$,
2. and all $(((a_x, i, m_x), (a_y, i, m_y), *, \mathit{SamePair}), ((a'_x, j, m'_x), (a'_y, k, m'_y), *, -))$, where a_x and a_y are maximal letters,
3. and all $(((a_x, i, m_x), (a_y, j, m_y), *, *), ((a'_x, i', m'_x), (a'_y, j', m'_y), *, *))$, where ($a_x$ is not maximal and ($m'_x \neq m_x + 1$ or $i \neq i'$)) or (a_y is not maximal and ($m'_y \neq m_y + 1$ or $j \neq j'$)) or (a_x is maximal and a'_x is not a first letter) or (a_y is maximal and a'_y is not a first letter).

Explanation: The first set of diagonal constraints ensures that there are always the same letters on the origin diagonal. This ensures that the two infinite words $x_{i_1}x_{i_2}\ldots$ and $y_{i_1}y_{i_2}\ldots$ of a solution of an ω-3-DTP are identical since any diagonal pattern that gets rid of the *SameLetter* flag is forbidden. In the initial colour, the *SameLetter* flag must be set. There are less constraints to satisfy if the *SameLetter* flag is only set on the origin diagonal. So, the *SameLetter* flag is not set if it can be avoided.

The second set of diagonal constraints ensures that a colour with the *SamePair* flag set must be chosen after both letters are maximal letters. This way, a new pair has to start diagonally after the subwords which belong to the same pair ended. Note that the subwords of the same pair might end in any coordinate of the tiling, in particular it does not have to be on the origin diagonal. Together with the third set of diagonal constraints it is guaranteed that the two subwords of the next pair start with their initial letter. As for the *SameLetter* flag, this flag must be set in the initial colour. From there on, it is desirable to avoid the *SamePair* flag whenever it can be avoided since there are more constraints to satisfy if it is set.

The third set of diagonal constraints ensures that each word is depicted letter by letter and that new subwords must start with their initial letter.

1. The **set of horizontal constraints** contains all $((*, q_y, *, *), (*, q'_y, *, *))$, where $q_y \neq q'_y$,

2. and all $(((a_x, i, m_x), *, *, *), ((a'_x, i', m'_x), *, *, *))$, where ($a_x$ is not maximal and ($m'_x \neq m_x + 1$ or $i \neq i'$)) or (a_x is maximal and a'_x is not a first letter),
3. and all $(((a_x, i, m_x), *, *, \mathit{SamePair}), ((a'_x, i', m'_x), *, *, -))$, where a_x is not maximal.

Explanation: The first set of horizontal constraints ensures that the colour q_y cannot change within a row. In particular, the index and letter of y remain the same within a row. The second set ensures that a word of x is depicted letter by letter in a row and that a new word of x starts with its first letter. The third set prevents the colouring from omitting the *SamePair* flag at any point except after a maximal letter of a word in x within a row. Note that a colour with the *SamePair* flag set has the same constraints as the same colour but without the *SamePair and* the extra constraints coming with the *SamePair* flag. Therefore, avoiding the *SamePair* flags always leaves strictly more valid colouring options. The same goes for the *SameLetter* flag. This flag must be set only on the origin diagonal and should not be set anywhere else since it adds unnecessary constraints if set somewhere besides the origin diagonal.

The set of vertical constraints is defined symmetrically, so that the index and letter of x remain the same within a column and the words of y are depicted letter by letter in a column. Together with the sets of diagonal constraints, this ensures that if a tile colour has both maximal letters in its subwords of x and y, and the *SamePair* flag is set, the diagonal colour starts new subwords of x and y and must also have the *SamePair* flag set, i.e. the next pair starts diagonally.

1. The set of **vertical constraints** contains all $((q_x, *, *, *), (q'_x, *, *, *))$, where $q_x \neq q'_x$,
2. and all $(*, ((a_y, i, m_y), *, *), (*, (a'_y, i', m'_y), *, *, *)$, where ($a_y$ is not maximal and $m'_y \neq m_y + 1$ and $i \neq i'$) or (a_y is maximal and a'_y is not a first letter),
3. and all $(*, ((a_y, i, m_y), *, \mathit{SamePair}), (*, (a'_y, i', m'_y), *, *, -)$, where a_y is not maximal.

In Fig. 4.1, it is shown how the *SameLetter* and *SamePair* are carried through an ω-tiling. On the left hand side in Fig. 4.1: Since the initial colour has the same letter flag set, the same letter flag is kept in all diagonal patterns. On the right hand side: An example of the same tile flags for a sequence of indices $i_1 i_2 i_3 i_4 \ldots$. Here we can see the length of the words x_{i_j} on the horizontal axis: $|x_{i_1}| = 2$, $|x_{i_2}| = 2$, $|x_{i_3}| = 4$, $|x_{i_4}| = 1$ and the length of the words y_{i_j} on the vertical axis: $|y_{i_1}| = 1$, $|y_{i_2}| = 3$, $|y_{i_3}| = 2$, $|y_{i_4}| = 3$.

Now, we prove that this is a reduction of the ω-PCP to the ω-3-DCP defined as above. First, we show that there exists an infinite sequence of indices $i_1, i_2, \ldots$ with $x = x_{i_1} x_{i_2} \ldots = y_{j_1} y_{j_2} \ldots = y$ if such an ω-tiling f exists.

The ω-tiling f satisfies the colouring constraints. So the initial colour has the *SameLetter* flag set. Since we forbid all diagonal patterns where the *SameLetter* flag is removed, all colours on the diagonal have the *SameLetter* flag set, which is shown in

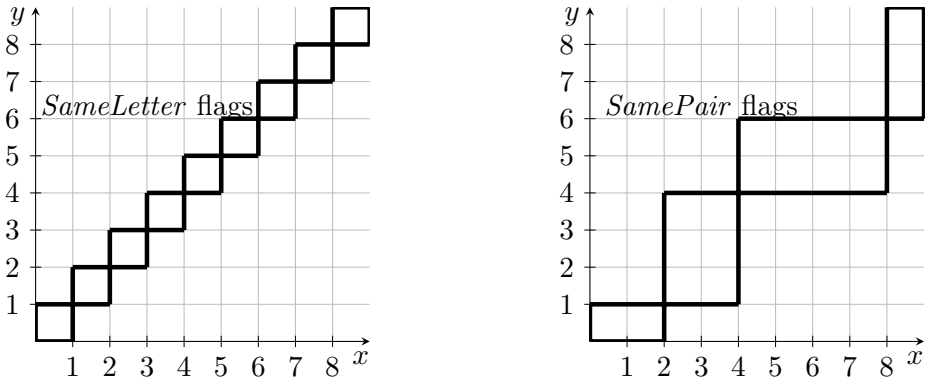


Figure 4.1: The *SameLetter* and *SamePair* flags in the colouring of an ω-PCP solution.

Fig. 4.1. Since all colours with the *SameLetter* flag have the same letter in their colours, $x = y$ holds.

Now, we turn to the sequences of indices. The initial colour has the *SamePair* flag set. The horizontal and vertical colouring constraints guarantee that the first component q_x cannot change within the same column and the second component q_y cannot change within the same row. We also guarantee that the colouring respects that for any chosen index the whole word is depicted letter by letter. We can only remove the *SamePair* flag after maximal letters. After a maximal letter in q_x the next colour in the same row gets rid of the *SamePair* flag and symmetrically for q_y. The crucial case is when both letters of a colour are maximal letters. Then, the diagonal constraints ensure that the *SamePair* flag is kept diagonally, which ensures that the sequence of indices is the same since all colours with the *SamePair* flag set have the same index.

Second, if the ω-PCP has a solution $i_1, i_2, \ldots$, $\forall j \in \mathbb{N} : i_j \in \{1, \ldots, n\}$ such that $x = x_{i_1} x_{i_2} \ldots = y_{i_1} y_{i_2} \ldots = y$, we choose the ω-tiling according to the given sequence of indices, i.e. we choose the colour accordingly to the indices and letters of the solution. We choose the initial colour as $((a, j, 1), (a, j, 1), \mathit{SameLetter}, \mathit{SamePair})$, where a is the first letter of x (also y), j is the index of the first pair, $j = i_1$, and both flags are set. The *SameLetter* flag is only kept on the origin diagonal and is not set anywhere else. This is possible since it must be set only in the initial colour and from there on it must be kept only diagonally. The *SamePair* flag is removed horizontally after the component of Q_x of the colour has a maximal letter. The *SamePair* flag is removed vertically after the component of Q_y of the colour has a maximal letter. Only if a colour has both maximal letters in its components of Q_x and Q_y, the *SamePair* flag is kept diagonally. This way, the second set of forbidden diagonal patterns is avoided. The third set of forbidden diagonal patterns is avoided since the solution of the ω-PCP is converted letter by letter into an ω-tiling.

The first set of forbidden horizontal patterns is avoided since we do not change the component Q_x within a column. The second set of forbidden horizontal patterns is avoided since the ω-PCP solution is converted into an ω-tiling letter by letter. The third

set of forbidden horizontal patterns is avoided since the *SamePair* flag is only removed after maximal letters and not within a subword.

The set of forbidden vertical constraints are avoided analogously to the forbidden horizontal patterns since their definition is symmetrically. Then we satisfy all the constraints of the colouring problem. □

In the following an example of the reduction of an instance of the ω-PCP to an ω-3-DTP is presented.

Example 4.7 (ω-PCP to ω-3-DTP). Let (b, b, aba) and (ba, aba, b) be an instance of an ω-PCP with alphabet $\Sigma = \{a, b\}$. Then, a solution of this ω-PCP is given by the sequence of indices $13232\ldots$, i.e. after starting with the first pair, the third and second pair alternate infinitely. The words $x = x_1x_3x_2x_3x_2\ldots = b \cdot aba \cdot b \cdot aba \cdot b\ldots = ba \cdot b \cdot aba \cdot b \cdot aba\ldots = y$ are identical.

The finite set of colours $Cl \subseteq Q_x \times Q_y \times \{\mathit{SameLetter}, -\} \times \{\mathit{SamePair}, -\}$ of the corresponding ω-3-DTP is given as follows: $Q_x = \{(b, 1, 1), (b, 2, 1), (a, 3, 1), (b, 3, 2), (a, 3, 3)\}$. Reminder: the format of the elements $(l, i, m) \in Q_x \cup Q_y$ consists of a letter $l \in \Sigma$, the index of the pair i and the position of the letter m in the subword of the pair of i. Analogously, the set Q_y is $\{(b, 1, 1), (a, 1, 2), (a, 2, 1), (b, 2, 2), (a, 2, 3), (b, 3, 1)\}$. The set of colours Cl contains no 4-tuples where the same letter flag is set but the letters of the components from Q_x and Q_y differ. It also contains no 4-tuples where the same pair flag is set but the indices of the pairs of the components from Q_x and Q_y differ. The only initial colour here is $((b, 1, 1), (b, 1, 1), \mathit{SameLetter}, \mathit{SamePair})$.

In Fig. 4.2, the ω-tiling of the corresponding ω-3-DTP which satisfies the colouring constraints is shown. The *SameLetter* and *SamePair* flags are abbreviated as *SL* and *SP*, respectively. All colours, which have the *SamePair* flag set, are outlined. On the origin diagonal the *SameLetter* flag is set. The *SameLetter* flag imposes strictly more constraints on the ω-tiling than the same colour without that flag. The same holds for the *SamePair* flag. Therefore, it is desirable to avoid setting those flags unnecessarily when searching for an ω-tiling. Note that the component of Q_x of a colour does not change within a column and the component of Q_y of a colour does not change within a row according to the colouring constraints. All cells that have neither flag set are left empty. These colours can be derived from the pictured colours since the first component does not change within a column and the second component does not change within a row. ◁

4.2 ω-3-DTP to 6-player Petri games

In the following we define a Petri game for which a winning strategy exists if and only if a given ω-3-DTP has a solution. The constructed Petri game depends on the given ω-3-DTP. The template of such a Petri game is shown in Fig. 4.3. There are two structurally identical parts consisting of 3 environment players, a top part and a bottom part. A place e_{ij}^T, $i = 0, 1, 2$, $j = 0, \ldots, 5$ is the j-th place of player i in the top part and a place e_{ij}^B, $i = 0, 1, 2$, $j = 0, \ldots, 5$ is the j-th place of player i in the bottom part. At each

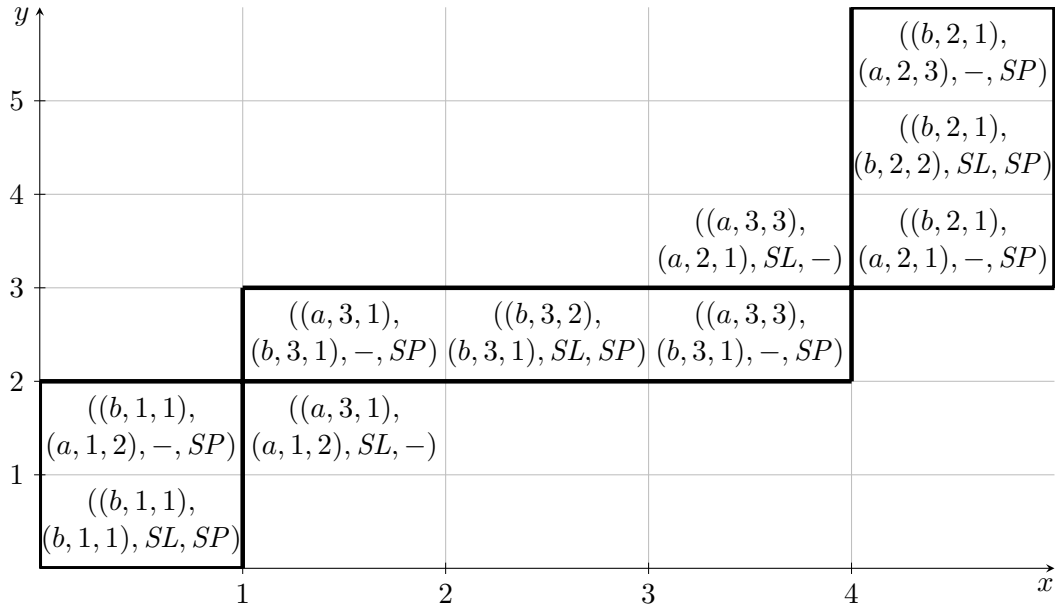


Figure 4.2: ω-tiling of a solution of an ω-PCP.

transition, two players from the same part synchronize. After firing 3 such transitions, each player has synchronised with the other two players in its part, e.g. after the first 3 transitions in the top part its players are on the places e^T_{02}, e^T_{12} and e^T_{22}. We define such a block of 3 transitions as a *round* in the net. The bottom and the top part have 3 rounds each. After the third round the players return to the places they were at the start of the second round, e.g. those are the places e^T_{02}, e^T_{12} and e^T_{22} in the top part. The dashed arrows indicate that the places with the same label are actually the same places. In the unfolding, the places e^T_{02}, e^T_{12} and e^T_{22} are copied to unfold the net, i.e. the dashed arrows lead to new conditions with the same label. We extend the notion of a round to the unfolding. Three events with the labels of a round in the net are also referred to as a round (in the unfolding).

Looking at the 3 rounds, we have an initial round and then the players alternate between the second and third round. As the game progresses, the nodes in the unfolding have exact information about how many rounds they have played in the top and bottom part, respectively, i.e. in the unfolding the second and third round are repeated infinitely and a node has information of how many rounds have been played in the top and bottom part. So, the players count the number of rounds for both part separately. For example, if the first round is played and the second and third round have been repeated both four times in the top part, the tenth round starts in the top part, while the bottom part could be still in the initial round. The labels of the tenth round in the unfolding are the places of the second round in the net.

Besides the structurally identical top and bottom parts there are check transitions in the net which are shown between the top and bottom part. In this middle part,

the environment players can take a *check transition* t_{aj-ak}, where $a \in \{0,1,2\}$ and $j,k \in \{0,\dots,5\}$ and $pre(t_{aj-ak}) = \{e^T_{aj}, e^B_{ak}\}$. In the unfolding, a *check event* is an event where its label is a check transition. We refer to the number of rounds of a check event as the number of rounds played in the *unfolding* counting the number of rounds played in the top part and bottom part separately. Here, a is the index of the player from the bottom and the top part, i.e. a check transition takes the players from the bottom and top part with the same index. The indices j and k are arbitrarily between 0 and 5, i.e. a check transition may be taken any time for the players with the same index a no matter on which places they are, i.e. no matter which labels their conditions have in the unfolding.

Informally speaking, a winning strategy can be obtained by deleting parts of the unfolding. So, for the idea how a winning strategy looks like we look at the unfolding of the Petri game in Fig. 4.3. The idea is the following: After a check event, the system player on the condition with label s_a has to choose a colour $c \in \{c_1,\dots,c_m\}$. Then, based on the number of rounds x and y played in the unfolding in the bottom part and top part, respectively, the system player has to determine the colour $f(x,y)$ of the given ω-3-DTP. Each colour choice requires one player from each part of the Petri game, so there is a maximum of 3 colour choices in one transition sequence of the Petri game.

If the colours chosen refer to a forbidden pattern a bad marking is reached. As the bad markings have to be defined on the Petri net and not in the unfolding, we define in the net when two players of the same part are in the same round or when a player is one round ahead or one round behind to be able to check for forbidden patterns. This is determined by looking at the places e_{aj-ak} in the postset of the check transitions t_{aj-ak}, i.e. for two check events with labels t_{aj-ak} and $t_{bj'-bk'}$ executed in the same event sequence the labels e_{aj-ak} and $e_{bj'-bk'}$ encode whether a horizontal, vertical or diagonal pattern is checked. If the colour choices are a forbidden pattern of that kind, a bad marking is reached. The rounds used to encode which pattern is checked are formally defined in Def. 4.9.

We start with the formal definition of the Petri net shown in Fig. 4.3. Afterwards, the definition of when a player is a round behind or ahead follows. Then, we conclude with the definition of the Petri game by adding a set of bad markings to the net.

Definition 4.8 (Petri net of an ω-3-DTP). Let $TP = (Cl, (Cl_i, DP, VP, HP))$ be an instance of the ω-3-DTP. We define the Petri net N_{TP} as follows:

$$\begin{aligned}\mathcal{P}_{TP} &= \{e^X_{aj} \mid X \in \{T,B\}, a \in \{0,1,2\}, j \in \{0,\dots,5\}\}\\ &\cup \{e_{aj-ak} \mid a \in \{0,1,2\}, j,k \in \{0,\dots,5\}\}\\ &\cup Cl \times \{0,1,2\} \cup \{s_0,s_1,s_2\}\end{aligned}$$

$$\begin{aligned}\mathcal{T}_{TP} &= \{t_{aj-ak} \mid a \in \{0,1,2\}, j,k \in \{0,\dots,5\}\}\\ &\cup \{t^X_{01-r} \mid X \in \{T,B\}, r \in \{0,1,2\}\}\\ &\cup \{t^X_{12-r} \mid X \in \{T,B\}, r \in \{0,1,2\}\}\\ &\cup \{t^X_{02-r} \mid X \in \{T,B\}, r \in \{0,1,2\}\}\\ &\cup \{t_{ac} \mid a \in \{0,1,2\}, c \in Cl\}\end{aligned}$$

$$
\begin{aligned}
&pre_{TP}(t_{aj-ak}) = \{e^T_{aj}, e^B_{ak}\},\\
&pre_{TP}(t^X_{01-r}) = \{e^X_{0j}, e^X_{1k} \mid j = 2\cdot r, k = 2\cdot r\} \text{ for } X \in \{T, B\},\\
&pre_{TP}(t^X_{12-r}) = \{e^X_{1j}, e^X_{2k} \mid j = 2\cdot r+1, k = 2\cdot r\} \text{ for } X \in \{T, B\},\\
&pre_{TP}(t^X_{02-r}) = \{e^X_{0j}, e^X_{2k} \mid j = 2\cdot r+1, k = 2\cdot r+1\} \text{ for } X \in \{T, B\},\\
&pre_{TP}(t_{ac}) = \{s_a\},
\end{aligned}
$$

$$
\begin{aligned}
&post_{TP}(t_{aj-ak}) = \{e_{aj-ak}, s_a\}\\
&post_{TP}(t^X_{01-r}) = \{e^X_{0j}, e^X_{1k} \mid j = 2\cdot r+1, k = 2\cdot r+1\} \text{ for } X \in \{T, B\},\\
&post_{TP}(t^X_{12-r}) = \begin{cases} \{e^X_{1j}, e^X_{2k} \mid j = 2\cdot r+2, k = 2\cdot r+1\} & r \in \{0,1\} \\ \{e^X_{1j}, e^X_{2k} \mid j = 2, k = 2\cdot r+1\} & r = 2 \end{cases} \text{for } X \in \{T, B\},\\
&post_{TP}(t^X_{02-r}) = \begin{cases} \{e^X_{0j}, e^X_{2k} \mid j = 2\cdot r+2, k = 2\cdot r+2\} & r \in \{0,1\} \\ \{e^X_{0j}, e^X_{2k} \mid j = 2, k = 2\} & r = 2 \end{cases} \text{for } X \in \{T, B\},\\
&post_{TP}(t_{ac}) = \{(c, a)\}
\end{aligned}
$$

$$
In_{TP} = \{e^X_{a0} \mid X \in \{T, B\}, a \in \{0, 1, 2\}\}
$$

◁

Note that the depiction of the net N_{TP} for a given tiling problem TP is easier to understand in Fig. 4.3 than in Def. 4.8. The transitions of the set $\{t^X_{01-r} \mid X \in \{T, B\}, r \in \{0, 1, 2\}\} \cup \{t^X_{12-r} \mid X \in \{T, B\}, r \in \{0, 1, 2\}\} \cup \{t^X_{02-r} \mid X \in \{T, B\}, r \in \{0, 1, 2\}\}$ are not named explicitly in Fig. 4.3. These transitions are the transitions in the top and bottom part. For example, the first transition in the top left corner of the top part is t^T_{01-0}. The T indicates that it is a transition of the top part. The first part of the index, 01 indicates that player 0 and player 1 of the top part participate, i.e. the preset contains $e^T_{\mathbf{0}0}$ and $e^T_{\mathbf{1}0}$. At last, the -0 of the index in t^T_{01-0} indicates the number of the round in the *net* (not the unfolding) of the transition (offset by 1).

Now, we define for two players of the same part (top or bottom) when one player is a round ahead (or behind) the other player. This is defined considering the places of the net. Then in the unfolding, we can determine if a condition is a round ahead (or behind) solely by looking at the labels of the conditions. Informally speaking, a player on the place at the end of a round in the net, e.g. e^B_{24} is the place of the bottom player 2 at the end of round 2, is a round behind the places at the start of the next round, e.g. the place e^B_{05} of the bottom player 0. Note that the marking $\{e^B_{05}, e^B_{15}, e^B_{24}\}$ is a reachable marking in N_{TP}. If the bottom player 2 takes a check event from a condition with label e^B_{24} while the bottom player 0 takes a check event from a condition with label e^B_{05} the number of rounds played in the bottom part for the check after e^B_{05} is one larger than in the check after e^B_{24}. Thus, the choices of colours refer to a horizontal (or diagonal) pattern, i.e. for the horizontal constraint check the colours $f(x, y)$ and $f(x + 1, y)$ are chosen or for the diagonal check the colours $f(x, y)$ and $f(x + 1, y + 1)$ are chosen. The diagonal check occurs if the label of the condition of the check of the top player 0 is also a round ahead of the label of the condition of the check of the top player 2, e.g. the label

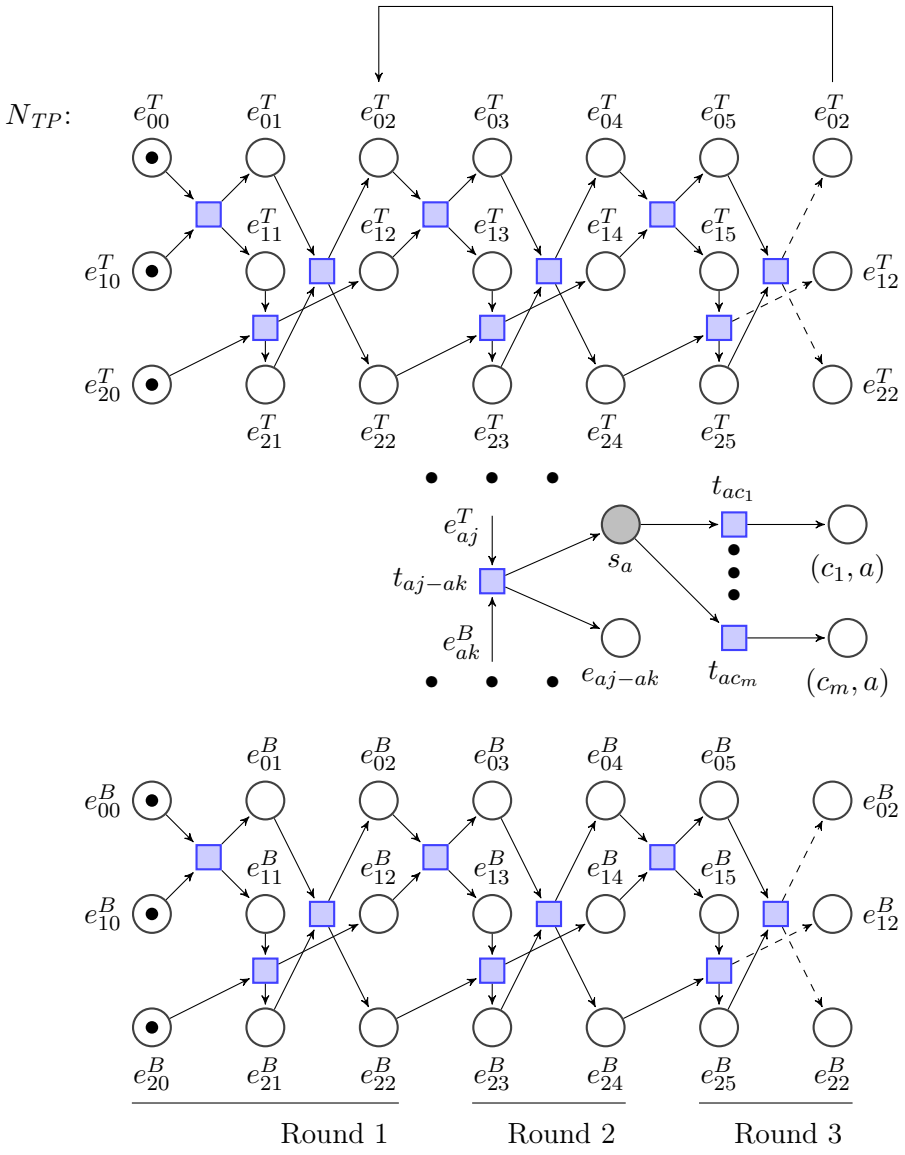


Figure 4.3: Petri net N_{TP} of the Petri game G_{TP} of an ω-3-DTP with the set of colours $Cl = \{c_1, \ldots, c_m\}$.

e^T_{05} is a round ahead of the label e^T_{24}. If the two top players are in the same round a horizontal constraint is checked. If the roles of the bottom and top player are swapped, a vertical constraint is checked, i.e. it is checked if the vertical pattern is a forbidden pattern.

Definition 4.9 (Rounds). Let TP be an instance of the ω-3-DTP and N_{TP} its net from Def. 4.8. We compare the rounds of two players $a, b \in \{0, 1, 2\}$ from the same part, $X \in \{T, B\}$. We say player a and b are in the *same round* if and only if for their places e^X_{aj} and e^X_{bk} it holds that $(j, k \in \{0, 1, 2\})$ or $(j, k \in \{3, 4\})$ or $(j, k \in \{5, 2\})$. We say that a is one *round ahead* of b (or b is one *round behind* of a) if and only if ($j \in \{3\}$ and $k \in \{2\}$) or ($j \in \{5\}$ and $k \in \{4\}$). Other combinations of indices are not possible. $\lhd$

Note that it is possible to define whether a place in the *net* is a round ahead (or behind) since every round in the net consists of at least two places for every player indicating if the player is at the start of a round or the end of a round. There is no reachable marking where two players are at the end of different rounds in the same part (top or bottom) of the net. Therefore, if a player is at the end of a round, the other two players of the same part are either in the same round (start or end) or at the start of the *next* round in the net.

We continue with the definition of the Petri game of an instance of the ω-3-DTP. The places are partitioned into a set of environment places and a set of system places as shown in Fig. 4.3. For the set of bad markings, five different sets of bad subsets of places are defined. The set of bad markings is defined in a way such that every reachable marking that contains a bad subset of places is a bad marking. Now speaking in the context of the unfolding, the sets in $\mathcal{B}_{Same}$ ensure that the same colour is chosen if two check events are fired where the top rounds of the two check events coincide and the bottom rounds of the two check events coincide, i.e. the system players do not cheat by choosing different colours for the same grid position. This is necessary since a check between two different rounds in the same part is only possible if one round is at its end and the other has just started. It is not possible to compare the colours chosen at the end of two different rounds in a single firing sequence. This means, that it is necessary to prevent the system players to change their choice within a round. The set $\mathcal{B}_{DP}$, $\mathcal{B}_{HP}$ and $\mathcal{B}_{VP}$ ensure that the diagonal colouring constraints, the horizontal colouring constraints and the vertical colouring constraints are met, respectively. Finally, a set $\mathcal{B}_{init}$ ensures that the initial colour is chosen according to the constraint. Round 1 of the Petri game in Fig. 4.3 is necessary to define the set $\mathcal{B}_{init}$ since the bad markings are defined on the net itself where it cannot be distinguished if a place is reached for the first time. Therefore, the game starts in round 1 and alternates between round 2 and round 3 afterwards such that the places in round 1 are only reached once at the start of the game. This is necessary to define the bad markings that are reached if the initial colour is not chosen according to its constraint.

Definition 4.10 (Petri game of an ω-3-DTP). Let $TP = (Cl, (Cl_i, DP, VP, HP))$ be an ω-3-DTP with colours Cl and constraints Cl_i, DP, VP and HP. The components of the Petri net of the Petri game G_{TP} are the components of N_{TP} from Def. 4.8 where

$\mathcal{P}^S = \{s_0, s_1, s_2\}$ and $\mathcal{P}^E = \mathcal{P}_{TP} \setminus \mathcal{P}^S$. The bad markings of the Petri game G_{TP} are defined as follows:

$\mathcal{B} = \{M \in \mathcal{R}(N_{TP}) \mid \exists S \in \mathcal{B}_{Same} \cup \mathcal{B}_{DP} \cup \mathcal{B}_{HP} \cup \mathcal{B}_{VP} \cup \mathcal{B}_{init} : S \subseteq M\}$, where
$\mathcal{B}_{Same} = \{\{e_{a^T j - a^B j'}, e_{b^T k - b^B k'}, (c_u, a), (c_v, b)\} \mid$
a^T and b^T are in the same round and a^B and b^B are in same the round, $c_u \neq c_v\}$,
$\mathcal{B}_{DP} = \{\{e_{a^T j - a^B j'}, e_{b^T k - b^B k'}, (c_u, a), (c_v, b) \mid$
a^T and a^B are one round ahead of b^T and b^B respectively, $(c_v, c_u) \in DP\}$,
$\mathcal{B}_{VP} = \{\{e_{a^T j - a^B j'}, e_{b^T k - b^B k'}, (c_u, a), (c_v, b)\} \mid$
a^T is a round ahead of b^T and a^B and b^B are in the same round, $(c_v, c_u) \in VP\}\}$,
$\mathcal{B}_{HP} = \{\{e_{a^T j - a^B j'}, e_{b^T k - b^B k'}, (c_u, a), (c_v, b)\} \mid$
a^B is a round ahead of b^B and a^T and b^B are in the same round, $(c_v, c_u) \in HP\}\}$,
$\mathcal{B}_{init} = \{\{e_{a^T j - a^B j'}, (c_u, a)\} \mid j, j' \in \{0, 1\}, c_u \notin Cl_i\}$. ◁

To translate a strategy of a game G_{TP} into an ω-tiling and vice versa, we define the number of rounds played in the unfolding. We say t is a transition of part X, $X \in \{T, B\}$, if $t \in \cup\{t^X_{01-r} \mid r \in \{0, 1, 2\}\} \cup \{t^X_{12-r} \mid r \in \{0, 1, 2\}\} \cup \{t^X_{02-r} \mid r \in \{0, 1, 2\}\}$.

Definition 4.11 (Rounds in unfolding). Let $u(G_{TP})$ be the unfolding of a game G_{TP} from Def. 4.10. The *number of rounds* played in part X, $X \in \{T, B\}$, in a condition $b \in \mathcal{P}_{u(G_{TP})}$ is defined as

$$r^X(b) = max(\lceil |\{e \in \mathcal{T}_{u_{TP}} \mid e \leq b \text{ and } h_u(e) \text{ is a transition in part } X\}|/3 \rceil, 1).$$

◁

The divisor of the fraction inside the upper Gaussian bracket is 3 because every round consists of 3 events. So, after the 4th event, the second round starts. Then, after the 7th events, the third round starts, and so on. Note that a player is already in the first round if no event has been taken which is expressed by the maximum over the cardinality of that set of events and the natural number 1. The number of rounds is not increased after a check event or an event choosing a colour.

Theorem 4.12 (Characterization of ω-3-DTP as a Petri game). *There exists a reduction of the ω-3-DTP to the synthesis problem of 6-player Petri games with a global safety condition.*

Proof. Let TP be an ω-3-DTP and G_{TP} its Petri game from Def. 4.10. We show that there exists a solution for TP if and only if there exists a winning strategy for G_{TP}.

We start with the implication that the existence of a solution of TP implies the existence of a winning strategy in G_{TP}. Let $f : \mathbb{N} \times \mathbb{N} \mapsto Cl$ be a solution for TP.

After the environment player has chosen a check event the system player must choose a colour in its condition with label s_a, $a \in \{0, 1, 2\}$. Since the system player knows the entire causal history, she knows the number of rounds played in the top and the bottom part of the Petri game in the unfolding. Let $r^T(s_a) = y$ denote the number of rounds in the top part and $r^B(s_a) = x$ the number of rounds in the bottom part. Then, the winning

strategy for the system players is to choose the colour $f(x, y)$. This way no bad marking is reachable. The bad markings in $\mathcal{B}_{Same}$ are avoided because $f(x, y)$ is unique. The bad markings in $\mathcal{B}_{DP}, \mathcal{B}_{VP}$ and $\mathcal{B}_{HP}$ are avoided because f avoids all forbidden patterns. Finally, the bad markings $\mathcal{B}_{init}$ are avoided because $f(1, 1)$ is an initial colour.

The deadlock avoidance property and determinism property are satisfied since a system player chooses exactly one colour after every check event. The justified refusal property is satisfied since the environment players are not restricted and there is no cut where the system players might violate the justified refusal property since there are no multiple events with the same label when choosing a colour.

Now, we assume that there is a winning strategy τ for the Petri game G_{TP}. We define the colouring $f : \mathbb{N} \times \mathbb{N} \mapsto Cl$ derived from the winning strategy as follows: $f(x, y) = c$ if there exists a system condition $s_a \in \mathcal{P}_\tau$ with $r^B(s_a) = x$ and $r^T(s_a) = y$ and $t_{ac} \in h_\tau(post(s_a))$, i.e. the system condition s_a has an event with label t_{ac} in its postset.

We show that for every colour that a system player has to choose, there are sequences of check events such that all colouring constraints are met. These sequences of checks are *not* event sequences. In these sequences of check events, any two consecutive checks can be carried out concurrently while the system players do not know whether the other check is the previous check or the next check, such that the winning strategy must play correctly for both checks. This means, that any two consecutive check events of these sequences are concurrent. We say such a sequence is a *sequence of subsequently concurrent events*. Two concurrent check events fired in the same event sequence coincide with a check whether the chosen colours meet the constraints of the ω-3-DTP, i.e. if the chosen colours form a forbidden pattern, or the system players chose the same colour in the same grid position. The grid position is determined by the number of rounds played in the bottom and top part, respectively.

Informally speaking, these sequences of subsequently concurrent check events lead to infinite sequences of dependencies since a colour choice after a check event depends on the choice of the colour of the previous check event and again that colour choice depends on the colour choice after the previous check event and so on. We can observe equivalent dependencies when looking at the ω-3-DTP. If we start colouring the infinite plane with an initial colour in $f(1, 1)$ followed by neighbouring tiles in an upwards vertical way, in an rightwards horizontal way or upwards (and rightwards) diagonal way, every colour we choose for such a tile is dependent on the colour of the previous tile to avoid forbidden patterns.

Since the system players do not know which pattern is checked due to the concurrency of the check events, they have to play correctly for all checks, i.e. for all colour choices due to another check event. A single sequence of subsequently concurrent check events is not sufficient to check all patterns whether they are forbidden. Even considering a single grid position, one such sequence is not sufficient. A single grid position has at most 6 patterns that need to be checked: horizontal left and ride, vertical up and down, and diagonally to the bottom left and the top right. Roughly speaking, for every such pattern, a sequence of subsequently concurrent check events is needed. We say roughly

speaking, since a single grid position does not relate to a single event in the unfolding but to all check events that have the same numbers of rounds as the grid position. So, in the sequences of subsequently concurrent check events the check if a grid position satisfies all colouring constraints considering the patterns it is part of involves many more such sequences. Informally speaking, the bad markings $\mathcal{B}_{Same}$ guarantee that after all check events relating to the same grid position the same colour must be chosen.

We show that there is such a sequence of subsequently concurrent check events for every pattern of every grid position that needs to be checked. We stress that the system players do not know which pattern is checked due to the concurrency of the check events. Therefore, they have to play correctly for all possible concurrent check events. We also stress, that a check event can be part of many different sequences of subsequently concurrent check events such that there is at least one for every pattern that needs to be check for every grid position. This way, all patterns of the ω-tiling f are checked whether they are forbidden patterns and whether the system players choose an unambiguous colour for every grid position.

	e^X_{00}	e^X_{10}	e^X_{20}	e^X_{01}	e^X_{11}	e^X_{21}	e^X_{02}	e^X_{12}	e^X_{22}	e^X_{03}	e^X_{13}	e^X_{23}	e^X_{04}	e^X_{14}	e^X_{24}
e^X_{00}		||	||												
e^X_{10}	||		||												
e^X_{20}	||	||		||	||										
e^X_{01}			||		||	||		||							
e^X_{11}			||	||												
e^X_{21}				||				||								
e^X_{02}								||	||							
e^X_{12}				||		||	||		||						
e^X_{22}							||	||		||	||					
e^X_{03}									||		||	||		||	
e^X_{13}									||	||						
e^X_{23}										||				||	
e^X_{04}														||	||
e^X_{14}										||		||	||		
e^X_{24}													||	||	

Figure 4.4: Table of concurrent conditions (only their labels are shown) of the first two rounds in the unfolding of the Petri game G_{TP}. Empty cells denote that the conditions are causally related. Filled cells denote that the places are concurrent.

Table 4.4 shows the concurrency of the conditions of the first two rounds in the unfolding of the Petri game G_{TP}. Empty cells denote that the places are causally related. Cells which are filled with || denote that the conditions are concurrent. The pattern continues identically for the conditions beyond round 2. With the help of this table we can see the sequences of checks which ensure that no forbidden pattern occurs in f and the system players choose a unique colour for every coordinate $(x, y) \in \mathbb{N}^2$. For example, to check that the initial colour chosen after an event with label t_{00-00} and after an event with label t_{22-00} in the same round is the same, we can perform the following sequence of checks of the indices of the places in the top part: $00 \to 20 \to 01 \to 12 \to 22$. There are no events fired in the bottom part, i.e. this refers to the sequence of of

subsequently concurrent check events with labels $t_{00-00} \to t_{20-20} \to t_{01-00} \to t_{12-10} \to t_{22-20}$. Note that there could also be events in the bottom part, e.g. to check if after the check events with labels t_{00-00} and t_{22-22} in the same bottom and top rounds the same colours are chosen. However, we have a more detailed look on the presented sequence of subsequently concurrent check events. Since the initial conditions with labels e^T_{00} and e^T_{20} are concurrent, there exists an event sequence where t_{00-00} and t_{20-20} are both checked concurrently. To avoid a bad marking induced by the set $\mathcal{B}_{Same}$ the system players on the conditions with labels s_0 and s_2, respectively, need to choose the same colour. Additionally, to avoid a bad marking induced by the set $\mathcal{B}_{init}$ the chosen colour must be an initial colour. After the check t_{20-20}, the system player 2 does not know whether the other check is t_{00-00} or t_{01-00} since check events with these labels are both concurrent. Thus, in the event sequence with the checks t_{01-00} and t_{20-20} the colour chosen by system player 0 after the check t_{01-00} must be the same colour as after the check t_{20-20}. Transitively, the colour after check t_{00-00} is the same as the colour after check t_{01-00}. This continues for the checks t_{12-10} and t_{22-20}.

We show that for *every* two check events that there exists a sequence of subsequently concurrent check events starting and ending with one of the two check events, respectively. This way, we ensure that all patterns are checked for forbidden patterns and the system players cannot cheat by choosing different colours in the same rounds, i.e. for the same coordinate $(x, y) \in \ltimes^2$. For that, we show all conditions in the top part are in one equivalence class of the transitive closure of the concurrency relation. This proof goes analogously for the conditions in the bottom part. The proof is by induction over the finite configurations of events containing no check event and no event of the bottom part. The finite configurations of the unfolding of N_{TP} are ordered w.r.t. a total adequate order $\preceq$. This order is restricted to the set of finite configurations which contain no check event and no event of the bottom part.

Base case of the induction: The first configuration is the empty configuration. The conditions in the initial marking are pairwise concurrent, in particular the condition in the top part.

Induction assumption: The set of conditions $\{b \in \mathcal{P}_{u(N_{TP})} \mid b \text{ is a condition in the top part and } [b] \preceq C_n\}$, where C_n is the n-th configuration regarding the total adequate order $\preceq$ restricted to the finite configuration of u_{TP} containing no check event and no event of the bottom part, is a single equivalence class of the transitive closure of the concurrency relation.

Let C_{n+1} be the next configuration regarding the restricted total adequate order $\preceq$. If C_{n+1} is not a local configuration of an event, the property holds by induction assumption since $\{b \in \mathcal{P}_{u(N_{TP})} \mid [b] \preceq C_n\} = \{b \in \mathcal{P}_{u(N_{TP})} \mid [b] \preceq C_{n+1}\}$ in this case.

Otherwise, we show for all $b \in \mathit{Cut}(C_{n+1})$ that there exists a concurrent condition in $\{b \in \mathcal{P}_{u(N_{TP})} \mid b \text{ is a condition in the top part and } [b] \preceq C_n\}$.

The event e such that $[e] = C_{n+1}$ is an event with two participating players of the top part leaving the third player of the top on the condition of the cut of the configuration $C_{n+1} \setminus \{e\}$. The conditions in the postset of e are concurrent to the condition of the player that did not participate in the top part, which is a condition

in $Cut(C_{n+1} \setminus \{e\})$. Since $C_{n+1} \setminus \{e\} \prec C_{n+1}$ that condition is in $\{b \in \mathcal{P}_{u(N_{TP})} \mid b \text{ is a condition in the top part and } [b] \preceq C_n\}$. Thus, the conditions of the postset of e are in the same equivalence class of the transitive closure of the concurrency relation.

Since the top and bottom part are independent, i.e. any two nodes from different parts are concurrent, the nodes of the top and bottom part in the unfolding form a single equivalence class in the transitive closure of the concurrency relation. It follows that for every two check events there exists a sequence of subsequently concurrent check events such that this sequence starts with one of those two events and ends with the other: Let e_{ai-aj} and $e_{bi'-bj'}$ be any two events in the unfolding with labels t_{ai-aj} and $t_{bi'-bj'}$, respectively. We look at the cuts before firing e_{ai-aj} and $e_{bi'-bj'}$, respectively. Firstly, we fix the conditions of the cut $Cut([e_{ai-aj}] \setminus \{e_{ai-aj}\})$ in the bottom part. Since the nodes in the top part are in a single equivalence class of the transitive closure of the concurrency relation there exists a sequence of subsequently concurrent events in the top part such that after each of those events a check event can be fired which is concurrent to the previous check event. This way, a sequence of subsequently concurrent check events is constructed until the conditions in the top part coincide with the conditions in the top part of the cut $Cut(e_{bi'-bj'} \setminus \{e_{bi'-bj'}\})$. Then, this is repeated for the bottom part. In particular, we can check for every grid position if there is any forbidden pattern vertically, horizontally or diagonally.

The ω-tiling f is well-defined as the set of bad markings $\mathcal{B}_{Same}$ ensures that every two colours chosen when both top rounds and both bottom rounds are the same the colours chosen also must be the same in a winning strategy. The ω-tiling does not contain any forbidden patterns because the infinite sequences of subsequently concurrent check events allow to check every pattern if it is a forbidden pattern. □

It is easy to see that the Petri game presented here fires at most 3 check transitions in every possible transition sequence since every check transition requires a player from the top and from the bottom part. Therefore, this Petri game has at most 3 system players. For the undecidability result it is sufficient to have 2 concurrent checks. Therefore, the number of system players can be further reduced to 2 by adapting the Petri game. This could be done by adding two environment players that are consumed performing a check. However, in that case the number of players is increased to eight. Reducing the number of system players to two while keeping six players in total is a challenging task and we do not know if it is possible.

Adding a transition for each bad marking to a bad place, the set of bad markings can be simplified to a set of bad places, which represents a local safety condition. We also note that the determinism property of a winning strategy is not necessary to show undecidability since a nondeterministic choice of a colour is not beneficial for the system players. In fact, a nondeterministic choice, i.e. at least two different colours are chosen after a check event, always leads to a bad marking due to the set $\mathcal{B}_{Same}$ which prevents the system players to choose different colours while being in the same top and same bottom round.

4.3 Related Work

The presented result lines up to several previous undecidability results for the synthesis of distributed systems. For temporal specifications (e.g. LTL), the synthesis problem is undecidable even for simple synchronous concurrent architectures [PR90, SF06]. As mentioned, in [Gim22], the synthesis problem for control games with 6 processes and a local termination condition is undecidable. This result is transferable to 6-player Petri games with a local termination condition [BFH19]. In the translated Petri games, all players alternate between being a system or an environment player. Here, the number of system players is at most three.

According to [FGHO22], it is undecidable whether a winning strategy exists for the system players in Petri games involving two system players and one environment player when the winning condition is a combination of good and bad markings. The winning condition states that a good marking must be reached eventually and no bad marking is reached beforehand; after a good marking is reached, bad markings may be visited. This condition is expressible in LTL.

Another undecidability result presented in [FGHO22] considers only good markings. If the game has at least three players – one of which is an environment player, and the other two alternate between being a system and an environment player – undecidability is shown.

Overall, the synthesis problem for asynchronous distributed systems is undecidable even for simple winning conditions such as local safety and local termination with 6 players. For more complex winning conditions like LTL, the synthesis problem for Petri games is already undecidable with at least 3 players [FGHO22].

4.4 Outlook

In this chapter, we have shown that the synthesis problem for 6-player Petri games with a local safety condition is undecidable. Due to the translation of Petri games to control games in [BFH19], it follows that control games with 6 processes equipped with a local safety condition are also undecidable. In [Gim22], it is shown that the synthesis problem of control games with 6 processes equipped with a local termination condition is undecidable. Obviously, the synthesis problem for Petri games and control games is also undecidable for more general winning properties. An example for more general winning strategies are regular properties. The local termination condition, i.e. all processes stop in a state from a given set of accepting states, is a regular property since the property can be checked by a finite automaton which reaches an accepting state itself if and only if all processes are in an accepting state.

One way to avoid undecidability is to restrict the causal memory in certain ways. In Chapter 5, this is done by limiting the part of the causal memory on which the system players base their decisions. The substantial novelty of the approach presented in Chapter 5 is the construction of the bounded causal memory in a labelled transition system (LTS). This enables us to use more complex winning conditions such as linear

temporal logic (LTL) while keeping decidability.

We conjecture that the synthesis problem for 2-player Petri games equipped with a winning condition expressed in LTL is already undecidable. The conjecture is based on the undecidability of Church's synthesis problem for distributed system. Church's distributed synthesis problem asks whether there exists a family of finite-state stream transducer satisfying a given input-output specification. A transducer takes a set of input streams over a finite alphabet and produces a set of output streams over a finite alphabet letter by letter depending on its state reached from the input streams. For example, a simple transducer with a single input and output stream might invert or delay an input stream depending on the specification it should satisfy. The transducers operate concurrently or produce/take the input/output stream for/from another transducer. In [PR90], for two concurrent transducers with single distinct input and output streams and a winning condition expressed in LTL, it is shown that the synthesis problem is undecidable. We assume that the proof is transferable to 2-player Petri games where each player refers to a transducer: the input stream is generated by a nondeterministic environment player and after each input letter the player is converted to a system player which produces an output letter followed by the next input letter from the player which is converted back to an environment player and so on.

Chapter 5

Finite-Memory Strategies

In this chapter, the basic concepts of how to keep track of a finite part of the (possibly infinite) causal memory are introduced. A finite state space is constructed, where two causal histories are matched to the same state if their finite parts of the causal history are the same regarding some predefined property. The finite state space is represented as a labelled transition system (LTS), which is suitable for effectively searching for winning strategies. Due to undecidability results in the setting where the causal history grows infinitely in [Gim22] and [Han23], restricting the memory to a finite part is a reasonable approach.

It is shown that the synthesis problem is decidable for a fixed finite memory in the setting presented here. Additionally, a further extension of the proposed construction of a finite memory is presented that allows us to refine key properties of the memory such as partial observation and asynchronous communication without increasing the time complexity class. In Section 5.2, we present refinements which cannot be directly expressed in the original model of Petri games without significant increase in complexity.

Furthermore, we argue that this approach is applicable for winning conditions expressed in any logic with decidable model checking problem on Petri nets, e.g. linear temporal logic (LTL). Bounded synthesis as introduced in [FS13] uses specifications expressed in LTL. In [FS13], *synchronous* distributed systems are considered and the output, i.e. the implementation, is restricted to finite state automata. Opposed to that, our approach considers *asynchronous* distributed systems and uses a more sophisticated memory. In the approach proposed here, the memory is another net itself, called a *memory net*. Using a net as the memory of a player enables to separate precisely between *which* events are of interest for that player and *when* the player gets to know of those events. For example, an occurring event is of interest for a player but she cannot observe that event directly and she has to take another event (e.g. communicating with another player) to gather the information about that event. Such a situation is characteristic for causal memory. Here, the implementation can be interpreted as a finite state automaton where its states are sub states of the memory net and we can specify when the sub states contain which information. The presented approach is an extension of how Petri games were introduced originally. Opposed to original Petri games, the memory model

presented here is not restricted to be a causal memory – but it still can be such.

Considering a memory that is not a causal memory of the Petri game, it is sensible to understand the memory as part of the specification, i.e. as part of the game. Since the memory is a net it can be specified which finite part of *its* causal history is the accessible memory for each player. Informally speaking, a memory net can be seen as an extra layer between the events of a Petri game and the causal memory of a player allowing to separate efficiently between control and information flow.

5.1 Last Known Marking Memory

We start with a finite memory where each token's memory is its last known marking. In every cut of a branching process, each token has its own last known marking.

All nets considered in this chapter are token-partitioning nets. This is not a necessary restriction to keep track of a finite memory, but it simplifies the formalism significantly because we do not need to track which token moves to which place and how many tokens are in a marking.

For a token-partitioning Petri net, its branching processes are also token-partitioning. For such a branching process $B = (N, h)$, we observe how the last known cut of a token in a cut $Cut(C)$ of a configuration C changes after firing an event. Reminder: the condition b of a token j, $j \in \{1, \dots, \nu\}$, in any cut $K \subseteq \mathcal{P}$ is denoted as $p_j(K) = b$, where $b \in K \cap P_j$ (Def. 2.29). By convention, we denote $lkc(p_j(Cut(C)))$ by $lkc_j(C)$ and $h(lkc_j(C))$ by $lkm_j(C)$. For example in Fig. 2.3, the cut of the configuration $C = \{a^1, c_a^1\}$ is $Cut(C) = \{S_1^1, A^1\}$, $p_1(Cut(C)) = S_1^1$ and so $lkc(S_1^1) = \{S_1^1, A^1\}$, which is denoted as $lkc_1(C)$ by this convention.

In the following lemma, it is stated how a last known cut is updated after firing an event e. Note that for a set of conditions S the mapping $\max_{\leq}(S)$ is defined as $\max_{\leq}(S) = b$, such that $\neg\exists b' \in S : b \leq b'$. This is well-defined if b is unambiguous. By Lemma 2.31, all conditions of one token i in $In \cup pre(C) \cup post(C)$ are causally related for any configuration C. Thus, $\max_{\leq}((In \cup pre(C) \cup post(C)) \cap P_i)$ is well-defined for any configuration C and token index $i \in I_P$.

Now, it is stated how the last known cut of token j is updated after firing an event. Therefore, for each token i its condition $p_i(lkc_j(C \cup \{e\}))$ is determined.

Lemma 5.1 (Last known cut update). *Let C be a finite configuration in a branching process B, e an event enabled in $Cut(C)$. For the condition $p_i(lkc_j(C \cup \{e\}))$ the following holds:*

Case:	$j \notin Ind(e)$	$j \in Ind(e)$	
		$i \in Ind(e)$	$i \notin Ind(e)$
$p_i(lkc_j(C \cup \{e\})) =$	$p_i(lkc_j(C))$	$p_i(post(e))$	$\max_{\leq}\{p_i(lkc_l(C)) \mid l \in Ind(e)\}$

Proof. **Case $j \notin Ind(e)$:** Since token j does not participate in e, the local configuration $[p_i(lkc_j(C \cup \{e\}))]$ is the same configuration as $[p_i(lkc_j(C))]$. Therefore, $lkc_j(C \cup \{e\}) = lkc_j(C)$ and the condition of token i does not change in particular, $p_i(lkc_j(C \cup \{e\})) = p_i(lkc_j(C))$.

Case $j, i \in Ind(e)$: By Lemma 2.22, the last known cut $lkc_j(C \cup \{e\})$ contains $post(e)$. Since the net is token-partitioning, there is exactly one condition of token i in every cut, this condition is $p_i(post(e))$.

Case $j \in Ind(e)$ and $i \notin Ind(e)$: Let C_j denote the local configuration $[p_j(Cut(C))]$ and C'_j denotes $C_j \cup \{e\} = [p_j(Cut(C \cup \{e\}))]$. According to Lemma 2.23, we have $Cut(C'_j) = (In \cup (\bigcup_{k \in Ind(e)} lkc_k(C)) \cup post(e)) \setminus pre(C_j)$.

This means, that the last known cut $lkc_j(C)$ can only contain conditions of In, conditions of the last known cuts $lkc_k(C)$, $k \in Ind(e)$, of the conditions in the preset of e or conditions in $post(e)$. By Lemma 2.21, these conditions are maximal. By Lemma 2.31, the successor relation is a total order on this set of conditions. Thus, since token i does not participate in e and its condition is not in the postset of e, the condition of token i is $\max_{\leq}\{p_i(lkc_l(C)) \mid l \in Ind(e)\}$. □

Example 5.2 (Update last known cut). In Fig. 5.1, all three cases of the update of a last known cut are observable. There, a net N_0 and two branching process B_1 and B_2 are shown. The partition of N_0 is given by $P_1 = \{E_1, F_1\}$, $P_2 = \{E_2, F_2\}$ and $P_3 = \{E_3, U, D\}$. Looking at the branching process B_1 the first case of Lemma 5.1 occurs for the configuration $C = \emptyset$, $e = u^1$ and $j = 2$. Token 2 on condition E_2^1 does not participate in the event e such that its last known cut remains unchanged, i.e. for all $i \in \{1, 2, 3\}$ the condition in the last known cut of token 2 remains unchanged: $p_i(lkc_2(C \cup \{e\})) = p_i(lkc_2(C))$.

The second case of Lemma 5.1 occurs for the configuration $C = \emptyset$, the event $e = u^1$ and the tokens $j = 1$ and $i = 3$. Then, $p_3(lkc_1(C \cup \{e\})) = p_3(post(e)) = p_3(\{E_1^2, U^1\}) = U^1$ is the new condition of token 3 in the last known cut of token 1 on condition E_1^2, which is the condition of token 1 in the cut of the configuration $C \cup \{u^1\} = \{u^1\}$.

The third case of Lemma 5.1 occurs for the configuration $C = \{u^1, r^1, d^1\}$, the event $e = f^1$ and the tokens $j = 1$ and $i = 3$. Here, the maximum $\max_{\leq}\{p_i(lkc_l(C)) \mid l \in Ind(e)\}$ is determined as follows: the participating tokens are $1, 2 \in Ind(f^1)$. The condition of token 3 in the last known cuts of of those tokens are $p_3(lkc_1(C)) = p_3(lkc(p_1(Cut(C))) = p_3(lkc(p_1(\{E_1^2, E_2^2, D^1\})) = p_3(lkc(E_1^2) = p_3(\{E_1^2, E_2^1, U^1\}) = U^1$ and $p_3(lkc_2(C)) = p_3(\{E_1^2, E_2^2, D^1\}) = D^1$. Since $U^1 < D^1$, $\max_{\leq}\{p_i(lkc_l(C)) \mid l \in Ind(e)\} = \max_{\leq}\{U^1, D^1\} = D^1$. ◁

We notice that the updated condition $p_i(lkc_j(C \cup \{e\}))$ can be determined if the condition $\max_{\leq}\{p_i(lkc_l(C)) \mid l \in Ind(e)\}$ is known, the postset of e is known and the last known cuts of each token in $Cut(C)$ are known (before firing e). To determine the condition $\max_{\leq}\{p_i(lkc_l(C)) \mid l \in Ind(e)\}$, we need to know the conditions $p_i(lkc_l(C))$ for all participating tokens l and their causal relation.

Since there are possibly infinitely many conditions, it is not possible to keep track of the conditions $p_i(lkc_l(C))$ in a finite state space. However, keeping track of only the labels of those conditions is possible. Therefore, instead of keeping track of the last known *cuts*, the finite state space keeps track of the last known *markings*. Last known markings repeat themselves eventually as opposed to last known cuts.

It follows, how to keep track of the causal relation needed to determine the condition $\max_{\leq}\{p_i(lkc_l(C)) \mid l \in Ind(e)\}$. We observe, how the causal relation is updated after firing an event.

Lemma 5.3 (Causal relation update). *Let C be a finite configuration in a branching process B, e an event enabled in $Cut(C)$. For the successor relation $\leq$ between the conditions $p_i(lkc_j(C \cup \{e\}))$ and $p_i(lkc_k(C \cup \{e\}))$ the following holds: (the question mark ? must be replaced by one of the symbols $=, <, >$ according to the stated cases in the second and third column.)*

Case:	$j,k \notin Ind(e)$	$j,k \in Ind(e)$
$p_i(lkc_j(C \cup \{e\}))$? $p_i(lkc_k(C \cup \{e\}))$	$\begin{cases} = & \text{if } p_i(lkc_j(C)) = p_i(lkc_k(C)) \\ < & \text{if } p_i(lkc_j(C)) < p_i(lkc_k(C)) \\ > & \text{if } p_i(lkc_j(C)) > p_i(lkc_k(C)) \end{cases}$	$=$

Case:	$j \in Ind(e), k \notin Ind(e)$	
	$i \in Ind(e)$	$i \notin Ind(e)$
$p_i(lkc_j(C \cup \{e\}))$? $p_i(lkc_k(C \cup \{e\}))$	$>$	$\begin{cases} = & \text{if } \max_{\leq}\{p_i(lkc_l(C)) \mid l \in Ind(e)\} = p_i(lkc_k(C)) \\ < & \text{if } \max_{\leq}\{p_i(lkc_l(C)) \mid l \in Ind(e)\} < p_i(lkc_k(C)) \\ > & \text{if } \max_{\leq}\{p_i(lkc_l(C)) \mid l \in Ind(e)\} > p_i(lkc_k(C)) \end{cases}$

Proof. In this proof, the conditions $p_i(lkc_j(C \cup \{e\}))$ and $p_i(lkc_k(C \cup \{e\}))$ are updated as defined in Lemma 5.1.

Case $j,k \notin Ind(e)$: Both conditions remain unchanged. Thus, the causal successor relation remains unchanged too.

Case $j,k \in Ind(e)$: Here, $p_i(lkc_j(C \cup \{e\})) = p_i(lkc_k(C \cup \{e\}))$. Therefore, the relation $=$ is correct.

Case $j,i \in Ind(e)$ and $k \notin Ind(e)$: Here, $p_i(lkc_j(C \cup \{e\})) = p_i(post(e))$ and $p_i(lkc_k(C \cup \{e\}))$ remains unchanged. By Lemma 2.21 and Lemma 2.22, $p_i(lkc_j(C \cup \{e\}))$ is maximal w.r.t. $\leq|_{In \cup pre([e])post([e])}$. Since $p_i(lkc_k(C \cup \{e\})) = p_i(lkc_k(C)) \neq p_i(post(e)) = p_i(lkc_j(C \cup \{e\}))$, and $p_i(lkc_j(C \cup \{e\}))$ and $p_i(lkc_k(C \cup \{e\}))$ are causally related by Lemma 2.31, $p_i(lkc_j(C \cup \{e\})) > p_i(lkc_k(C \cup \{e\}))$ holds.

Case $j \in Ind(e)$ and $i,k \notin Ind$: Here, $p_i(lkc_j(C \cup \{e\})) = \max_{\leq}\{p_i(lkc_l(C)) \mid l \in Ind(e)\}$ and $p_i(lkc_k(C \cup \{e\})) = p_i(lkc_k(C))$ remains unchanged. Therefore, the causal successor relation is the relation between $\max_{\leq}\{p_i(lkc_l(C)) \mid l \in Ind(t)\}$ and $p_i(lkc_k(C)) = p_i(lkc_k(C \cup \{e\}))$ as stated in the table. (These conditions are causally related by Lemma 2.31). □

Example 5.4 (Update causal relation). The update of the causal relation between the conditions $p_i(lkc_j(C \cup \{e\}))$ and $p_i(lkc_k(C \cup \{e\}))$ after firing an event e in a configuration C is is conducted as follows:

The first case in the upper table is the case where neither token j nor token k participates in the event e. Here, the causal relation between $p_i(lkc_j(C \cup \{e\}))$ and $p_i(lkc_k(C \cup \{e\}))$ is the same causal relation as before firing the event e, namely the causal

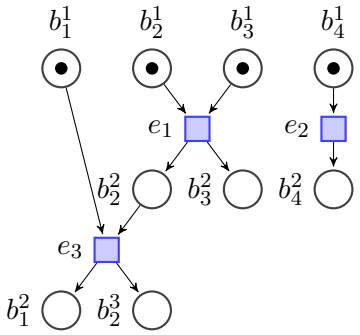
relation between $p_i(lkc_j(C))$ and $p_i(lkc_k(C))$. For example in Fig. 5.1, the configuration in B_1 is $\{u^1\}$ and the next event e to be fired is r^1. Since $p_3(lkc_1(\{u^1\})) = U^1 > p_3(lkc_2(\{u^1\})) = E_3^1$, after firing r^1 it still holds that $p_3(lkc_1(\{u^1\} \cup \{r^1\})) = U^1 < E_3^1 = p_3(lkc_2(\{uq\} \cup \{r^1\}))$.

For the second case of the upper table, both tokens j and k participate in the event e. From the definition of a last known cut it follows that for all $b \in post(e) : lkc(b) = lkc(e)$. Therefore, the last known cuts of token j and k are equal after firing event e. For example in Fig. 5.1, the last known cuts of U^1 and E_1^2 are equal after firing u^1. In particular, $p_2(lkc_1(C \cup \{u^1\})) = E_2^1 = p_2(lkc_3(C \cup \{u^1\}))$, where $C = \emptyset$.

For the first case of the lower table, token j and token i participate in the event e while token k does not. Informally speaking, token j always has the new condition of token i (the condition of token i in the postset of e) in its last known cut. This condition is guaranteed to be a causal successor ($<$) of the condition of token i in the last known cut of token k. In Fig. 5.1 for the configuration $C = \emptyset$ the event u^1, and $j = 1$, $k = 2$ and $i = 3$, it holds that $p_3(lkc_1(C \cup \{u^1\})) = U^1 > E_3^1 = p_3(lkc_2(C \cup \{e\})) = p_3(\{E_1^1, E_2^1, E_3^1\})$.

For the second case in the lower table, token j participates in the event e an token k and token i do not. The condition of token i in the last known cut of token j ($p_i(lkc_j(C \cup \{e\}))$) is the condition $\max_{\leq}\{p_i(lkc_l(C)) \mid l \in Ind(e)\}$ by Lemma 5.1. So, this case of the table states that the causal relation between $p_i(lkc_j(C \cup \{e\}))$ and $p_i(lkc_k(C \cup \{e\}))$ is the same causal relation between $\max_{\leq}\{p_i(lkc_l(C)) \mid l \in Ind(e)\}$ and $p_i(lkc_k(C)) = p_i(lkc_k(C \cup \{e\}))$. The former conditions are the same by Lemma 5.1 and the latter conditions are the same conditions because k does not participate in e and her last known cut remains unchanged. For example in Fig. 5.1, for the configuration $C = \{u^1\}$, the event $e = r^1$ and tokens $j = 3$, $k = 1$ and $i = 2$ it holds that $\max_{\leq}\{p_2(lkc_l(C)) \mid l \in Ind(r^1)\} = \max_{\leq}\{E_2^1\} = E_2^1 = p_2(lkc_3(C \cup \{r^1\}))$ and $p_2(lkc_1(C \cup \{r^1\})) = p_2(lkc_1(C)) = E_2^1$. Thus, $p_2(lkc_3(C \cup \{r^1\})) = E_2^1 = p_2(lkc_1(C \cup \{r^1\}))$.

For the cases $>$ and $<$, the following examples show when these cases occur.



For the branching process on the left, there are four partitions numbered from 1 to 4. Each condition is subscripted with its partition. Choosing $j = 4$, $k = 2$, $i = 3$, $C = \{e_1\}$ and $e = e_2$ the second case ($<$) of the second case in the lower table occurs: $p_3(lkc_4(\{e_1\} \cup \{e_2\})) = \max_{\leq}\{p_3(lkc_l(C)) \mid l \in Ind(e_2)\} = b_3^1 < b_3^2 = p_3(lkc_2(\{e_1\} \cup \{e_2\}))$.
The third case ($>$) of the second case in the lower table occurs if $j = 1$, $i = 3$, $k = 4$, $C = \{e_1\}$ and $e = e_3$. Then, $p_3(lkc_1(\{e_1\} \cup \{e_3\})) = \max_{\leq}\{p_3(lkc_l(C)) \mid l \in Ind(e_3)\} = \max_{\leq}\{b_3^1, b_3^2\} = b_3^2 > b_3^1 = p_3(lkc_4(\{e_1\} \cup \{e_2\}))$. ◁

Lemma 5.1 and Lemma 5.3 showed how to determine the last known cuts of each token in a configuration and the causal relation between their conditions, respectively.

This takes only a finite set of information, namely the last known cuts and the causal relation between the conditions of that token in the last known cuts of each token. This set of information must be further reduced since there might be infinitely many last known cuts. Therefore, the finite memory of a token in a Petri game is its last known marking. There are at most as many last known markings as reachable markings.

Beforehand, we present an example where it is necessary to store the causal order relation additionally to the last known markings. This case relates to the last case in Lemma 5.1, where token j participates but token i does not. It is possible that in two cuts, where all players have equal last known markings, the last known markings differ after firing events e and e' with $h(e) = h(e')$, respectively, i.e. after firing the same transition in the underlying Petri net. In Fig. 5.1 is an example of this behaviour.

Example 5.5 (Causal relation is necessary). In Fig. 5.1, the homomorphisms h_1 and h_2 from B_1 and B_2 to N_0 are given by dropping the superscripts, e.g., $h_1(E_1^1) = h_1(E_1^2) = E_1$. The partition of N_0 is given by $P_1 = \{E_1, F_1\}$, $P_2 = \{E_2, F_2\}$ and $P_3 = \{E_3, U, D\}$. In B_1 and B_2, the following last known markings are the same: $lkm(E_1^2) = lkm(E_1'') = \{U, E_1, E_2\}$ and $lkm(E_2^2) = lkm(E_2'') = \{D, E_1, E_2\}$. However, the last known markings $lkm(F_1^1) = \{D, F_1, F_2\}$ and $lkm(F_1') = \{U, F_1, F_2\}$ differ after firing f^1 and f' respectively. This shows that the last known markings are not a sufficient criterion for repetition. After firing f^1 in the branching process B_1 and f' in the branching process B_2 the last known markings differ although the last known markings were the same for each token before firing f^1 and f', respectively. Therefore, additionally to the last known markings the causal order relation between the conditions in the last known cuts is stored. ◁

We store the causal order relation of the conditions of every token as follows. For every token there is one causal order relation storing the causal order between the conditions of that token, one condition from the last known cut of each token. Equivalence classes of counters are introduced that capture the causal order relation.

Definition 5.6 (Counter equivalence classes). We define an equivalence relation $\sim \subseteq \mathbb{N}^\nu \times \mathbb{N}^\nu$ as follows. $(X_1, \ldots, X_n) \sim (X_1', \ldots, X_n') \Leftrightarrow \forall i, j \in \{1, \ldots, \nu\} : X_i < X_j \Leftrightarrow X_i' < X_j'$. The set of equivalence classes is denoted as EQ. The equivalence class of a tuple of numbers $(X_1, \ldots, X_n)$ is denoted as $[X_1, \ldots, X_n]$. ◁

For example, $[3, 2, 2, 4] = [5, 1, 1, 8]$. In the finite state space, every token j has a counter for every token i storing how many events token i has taken in the causal history of token j. We denote this counter as X_j^i. If X_j^i is greater than X_k^i, token j has more events of token i in its causal history than token k has events of token i in its causal history. Since such a counter is not bounded and only the causal order relation needs to be stored an equivalence class of counters is stored for every token i. The counter equivalence class $\equiv_i \in EQ$ for a token i consists of one counter of each token counting the events of token i, i.e., it is of the form $\equiv_i = [X_1^i, X_2^i, \ldots, X_\nu^i]$.

Example 5.7 (Counter equivalence class). In Fig. 5.1, token 1 has seen 1 event of token 3 on condition E_1^2, thus $X_1^3 = 1$, and token 2 on condition E_2^2 has seen 3 events of

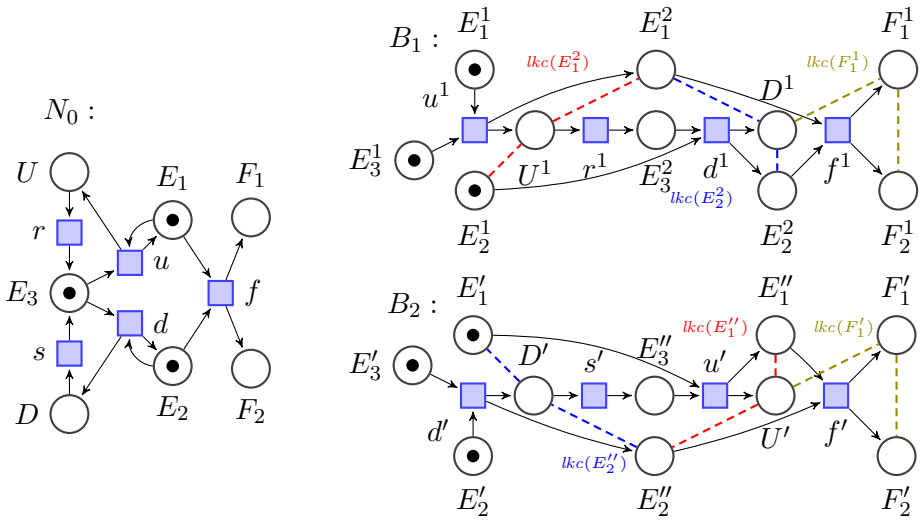


Figure 5.1: A Petri net N_0 on the left and two branching processes B_1 and B_2 of N_0 on the right.

token 3, thus $X_2^3 = 3$. Therefore, token 3's place in the last known marking of condition F_1^1 is place D of the last known marking of token 2 since it has seen more events of token 3 than token 1 has seen. It is crucial to see that the exact numbers of events are not stored but only their relation regarding the causal order relation $<$, $>$ and $=$. $\lhd$

In Fig. 5.2, we start with an example of a Petri net keeping track of the last known markings and the counter equivalence classes by constructing a finite labelled transition system (LTS).

Example 5.8 (LKM-memory LTS). In Fig. 5.2, an example of a memory LTS keeping track of the last known marking of each token is shown. The constructed LTS is an extended reachability graph of the net N. Each state consists of three rows: the last known markings of token 0 and token 1 are in the first row, the equivalence classes of the causal order relation are in the second row and the current marking is in the third row.

The LTS in Fig. 5.2 is the LTS of the Petri game G of Fig. 2.2. The partition P consists of $P_0 = \mathcal{P}^S = \{S_0, S_1, A', B'\}$ and $P_1 = \mathcal{P}^E = \{E_0, E_a, E_b, A, B\}$. We have assigned additional labels to the states (e.g. q_a) to refer to them. For example, in q_a the last known marking of the system token (token 0) is $\{S_0, E_0\}$ and the last known marking of the environment token (token 1) is $\{S_0, E_a\}$. The counters from left to right are: number of transitions taken by the system token in the causal memory of the system token, number of transitions taken by the environment token in the causal memory of the system token, number of transitions taken by the system token in the causal memory of the environment token and the number of transitions taken by the environment token in the causal memory of the environment token, which is 1 because only the environment player participates in transition a (transition from q_0 to q_a). Note

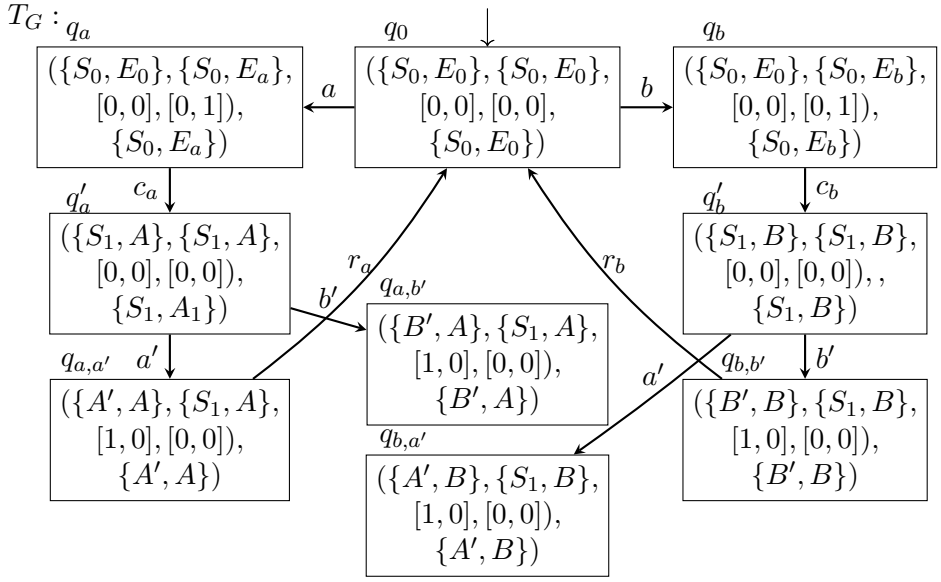


Figure 5.2: Memory LTS T_G of the Petri game G of Fig. 2.2.

that $[0,0] = [1,1] = [2,2]$ in q'_a after firing c_a. The reached marking is $\{S_0, E_a\}$. A bad marking is reached in the states $q_{a,b'}$ ($\{B', A\}$) and $q_{b,a'}$ ($\{A', B\}$). $\lhd$

The formal definition of the LKM-memory LTS follows. The *set of last known markings* of a token i in a net N is denoted as $LKS_i(N) = \{M \mid \exists b \in \mathcal{P}_u : b \in P_i \wedge lkm_u(b) = M\}$. The LTS is called a LKM-memory LTS since for every token its last known marking is tracked in the states of the LTS. However, it is additionally necessary to keep track of the causal relation between the conditions in the last known cuts to track the last known marking correctly as shown in Example 5.5. This is not reflected in the name LKM-memory LTS since it is not possible to construct such an LTS without the causal relations in general.

The set of inputs Σ is the set of transitions $\mathcal{T}$ of the net N for which the LKM-memory LTS $T_N = (Q, q_I, \Sigma, \delta)$ is constructed. A state $q \in Q$ of a LKM-memory LTS consists of the last known marking of each token, the counter equivalence class for each token and the reached marking. Note that the reached marking is a redundant information since it can be derived from the last known markings: the place of a token in the reached marking coincides with its place in its own last known marking. However, it is added for better readability. In the initial state q_I, each last known marking is the initial marking and all counters of the equivalence classes are equal (e.g. 0).

The partial transition function $\delta : Q \times \Sigma \nrightarrow Q$ of a such an LKM-memory LTS is closely aligned to the update of a last known cut in Lemma 5.1 and the update of the causal order relation in Lemma 5.3. The three cases of the update of the last known markings (item 2 of the transition function δ) in the LKM-memory LTS match the three cases in Lemma 5.1, but there are three general differences. The first one is that a last

known marking is determined instead of a last known cut, i.e. only the new labels of the conditions of the last known cuts are determined and not the conditions themselves. To determine the labels of the conditions in a last known cut it suffices to use the label of an event, *i.e.* the label $t \in \mathcal{T}$ is used instead of an event as in Lemma 5.1. So in the first case, token j does not participate in the transition t and its last known marking remains unchanged. In the second case, both tokens j and i participate in t such that the new place of token i in the last known marking of token j is its place in the postset of t.

The third difference occurs in the third case. Here, the counters of the equivalence classes are used instead of the causal successor relation as in Lemma 5.1. The counter equivalence class $[X_1^i, \ldots, X_\nu^i]$ captures the causal relation between the conditions of token i in the last known cuts of each token, e.g. $X_1^i > X_2^i$ means that the condition of token i in the last known cut of token 1 is a true causal successor of the condition of token i in the last known cut of token 2. So, in the third case $\max\{X_l^i \mid l \in Ind(t)\}$ is used to determine the correct place instead of $\max_{\leq}\{p_i(lkc_l(C)) \mid l \in Ind(e)\}$ as in Lemma 5.1.

The explanation for the update of the counter equivalence classes in item 3 of the partial transition function δ follows after the definition.

Definition 5.9 (LKM-memory LTS). We define the labelled transition system *LKM-memory LTS* $T_N = (Q, q_I, \Sigma, \delta)$ of a Petri net N with partition P, $|I_P| = \nu$, as follows:

- $Q = LKS_1(N) \times \ldots \times LKS_\nu(N) \times EQ^\nu \times \mathcal{R}(N)$ is the set of states. We refer to the last component of a state q as its marking $M(q) \in \mathcal{R}(N)$.
- $q_I = (In, \ldots, In, [0, \ldots, 0], \ldots, [0, \ldots, 0], In) \in Q$ is the initial state.
- $\Sigma = \mathcal{T}$ is the set of inputs.
- $\delta : Q \times \Sigma \nrightarrow Q$ is the partial transition function. The transition function δ is defined in (q, t) if and only if the transition t is enabled in $M(q)$. Then, for $\delta((q,t)) = q' = (lks'_1, \ldots, lks'_\nu, \equiv'_1, \ldots, \equiv'_\nu, M')$ the following holds:

  1. $M[t\rangle M'$,
  2. $lks'_j = (s'_1, \ldots, s'_\nu)$ with $s'_i = \begin{cases} p_i(lks_j) & \text{if } j \notin Ind(t) \\ p_i(post(t)) & \text{if } j, i \in Ind(t) \\ p_i(lks_k) & \text{if } j \in Ind(t) \wedge i \notin Ind(t) \\ & \wedge X_k^i = max\{X_l^i \mid l \in Ind(t)\}, \end{cases}$
  3. and $\equiv'_i = [X_1^{i\prime}, \ldots, X_\nu^{i\prime}]$ with $X_j^{i\prime} = \begin{cases} X_j^i & \text{if } j \notin Ind(t) \\ X_i^i + 1 & \text{if } j, i \in Ind(t) \\ \max\{X_l^i \mid l \in Ind(t)\} & \text{if } j \in Ind(t) \wedge i \notin Ind(t). \end{cases}$

$\triangleleft$

By convention, the memory LTS of a Petri net N is denoted as T_N and analogously for other Petri nets.

In item 3 of the transition function δ, it is stated how the counter equivalence class $\equiv_i' = [X_1^{i\prime}, \ldots, X_\nu^{i\prime}]$ is determined in the new state $\delta((q,t)) = q' = (lks_1', \ldots, lks_\nu', \equiv_1', \ldots, \equiv_\nu', M')$. Recapture that $X_1^{i\prime} > X_2^{i\prime}$ means that the condition of token i in the last known cut of token 1 is a true causal successor of the condition of token i in the last known cut of token 2. To understand the three cases of item 3 it is helpful to imagine $X_j^{i\prime}$ as the exact counter of how many events token i has taken in the causal history of token j. This is not completely accurate since $\equiv_i'$ is the equivalence class of those counters and might be another representative. However, for the explanation we imagine that $\equiv_i'$ is the representative that has the exact counters.

So in the first case the counter X_j^i remains unchanged since token j does not participate in t. In the second case, token i and token j participate in t. The counter X_i^i is always a maximal counter in the equivalence class $\equiv_i$ (from the previous state q) since a token i has complete knowledge about its own events. Therefore, the counter $X_j^{i\prime} = X_i^i + 1$ since token i has taken one more event now and token j has all events of token i in its causal history after firing a joint event with token i. For the third case, the counter $X_j^{i\prime}$ is the maximum $\max\{X_l^i \mid l \in Ind(t)\}$. This is the maximal counter of how many events token i has taken in a causal history of the participating tokens $l \in Ind(t)$. This is the number of events token i has taken in the causal history of token j since token i does not participate in t.

The cases of Lemma 5.3 cannot be directly matched to item 3 of the partial transition function. However, those cases are contained in item 3 if we compare two updated counters $X_j^{i\prime}$ and $X_k^{i\prime}$. For example, in the second case of the upper table in Lemma 5.3 where token j and token k participate in an event e (with label t) the condition of token i in the last known cut of token j is always the same condition as the condition of token i in the last known cut of token k. However, the counters are either updated to $X_j^{i\prime} = X_i^i + 1 = X_k^{i\prime}$ if token i participates in t or $X_j^{i\prime} = \max\{X_l^i \mid l \in Ind(t)\} = X_k^{i\prime}$ if token i does not participate in t. Thus, the cases in Lemma 5.3 are not directly matched in item 3 but they are matched as soon as the updates of two counters $X_j^{i\prime}$ and $X_k^{i\prime}$ with a third token k are considered.

In the following Lemma 5.10, it is shown that the update of two counters $X_j^{i\prime}$ and $X_k^{i\prime}$ coincide with the update of the causal successor relation in Lemma 5.3 in all cases. For this, we show that the counter equivalence classes and last known markings coincide after every finite event sequence in the unfolding.

For an event sequence $e_1 \ldots e_n$ of a branching process B of a Petri net N_0, the firing sequence in the memory LTS T_{N_0} is the firing sequence $h(e_1), \ldots, h(e_n)$. The last known markings and the counter equivalence classes are correct if for every event sequence its firing sequence in T_{N_0} leads to a state such that the last known markings of that state coincide with the last known markings of the tokens in the unfolding and the counter equivalence classes coincide with the causal successor relation of the conditions in the last known cuts.

Note that the branching process B in the following lemma can be chosen as the

unfolding.

Lemma 5.10 (Memory LTS tracks last known markings). *Let T_{N_0} be a memory LTS, $e_1 \dots e_n$ an event sequence of a branching process B of N_0 and $q_I \xrightarrow{h(e_1)} q_1 \dots \xrightarrow{h(e_n)} q_n$, its firing sequence in T_{N_0}. The configuration of the event sequence $e_1 \dots e_n$ is denoted as $C = \{e_1, \dots, e_n\}$. For $q_n = (lks_1, \dots, lks_\nu, \equiv_1, \dots, \equiv_\nu, M_n)$, the following holds:*

1. $\forall j : lks_j = lkm_j(C)$ *and*
2. $\forall i, j, k : X^i_j < X^i_k \Leftrightarrow p_i(lkc_j(C)) < p_i(lkc_k(C))$.

Proof. We prove the correctness by induction over the length of sequences of transitions. We start by noticing that property 1 is the same as $\forall i, j \in \{1, \dots, \nu\} : p_i(lks_j) = p_i(lkm_j(C))$.

Starting with the empty transition sequence, the initial state of the LTS is $q_I = (In, \dots, In, [0, \dots, 0], \dots, [0, \dots, 0], In)$. For all $i, j, k \in \{1, \dots, \nu\}$, it holds that $X^i_j = 0 = X^i_k$. The last known cut of each token in the initial marking of a branching process is the initial marking of that branching process, such that for all $i, j, k \in \{1, \dots, \nu\}$ it holds that $p_i(lkc_j(C)) = p_i(lkc_k(C))$. The last known marking of each token in the initial marking of a branching process is the initial marking of the Petri net. Thus, 1 and 2 hold.

The induction assumption is that the property holds for any firing sequence $e_1 \dots e_{n-1}$ of length $n-1$. In the following, it is shown that the property holds for any firing sequence $e_1, \dots e_{n-1}, e_n$ of length n, if the induction assumption holds.

The proof is now split into two parts. The first part shows the induction step for property 1 and the second part for property 2 For the induction step, we fix the following notations: $C_{n-1} = \{e_1, \dots, e_{n-1}\}$, $C_n = \{e_1, \dots, e_n\}$, $q_{n-1} = (lks_1, \dots, lks_\nu, \equiv_1, \dots, \equiv_\nu, M_{n-1})$ and $q_n = (lks'_1, \dots, lks'_\nu, \equiv'_1, \dots, \equiv'_\nu, M_n)$.

We show that the last known states are updated according to the table in Lemma 5.1.

Case $j \notin Ind(e_n)$: Here, $p_i(lks'_j) = p_i(lks_j)$. In Lemma 5.1, the condition of token i in $lkc_j(C_n)$ remains the same as in $lkc_j(C_{n-1})$ in this case. Thus, 1 holds by induction assumption.

Case $j, i \in Ind(e_n)$: Here, $p_i(lks'_j) = p_i(post(h(e_n)))$. In Lemma. 5.1, the condition of token i in lkc_j is updated to $p_i(post(e_n))$. With $p_i(post(h(e_n))) = h(p_i(post(e_n)))$, $p_i(lks'_j) = p_i(lkm_j(C_n))$ follows.

Case $j \in Ind(e_n)$ and $i \notin Ind(t)$: Here, $p_i(lks'_j) = p_i(lks_k)$ and the new counter is $X^i_k = max\{X^i_l \mid l \in Ind(t)\}$. By induction assumption $\equiv_i$ coincides with the causal relation between the conditions of token i in the last known cut of each player. The condition $p_i(lkc_j(C_n))$ is the maximum $\max_{\leq}\{p_i(lkc_l(C_{n-1})) \mid l \in Ind(e)\}$, which therefore is $p_i(lkc_k(C_{n-1}))$. Also by induction assumption, $lks_k = lkm_k(C_{n-1})$ holds. Thus, the update in Lemma 5.1 coincides with the update done here and 1 holds.

We show that the counter equivalence class is updated according to the table in Lemma 5.3.

Case $j, k \notin Ind(e_n)$**:** Both conditions $p_i(lkc'_j) = p_i(lkc_j)$ and $p_i(lkc'_k) = p_i(lkc_k)$ and their counters $X_j^{i\prime} = X_j^i$ and $X_k^{i\prime} = X_k^i$ remain unchanged. Thus, the property 2 holds by induction assumption.

Case $j, k \in Ind(e_n)$**:** Here, $p_i(lkc'_j) = p_i(lkc'_k)$. If $i \in Ind(e_n)$, we get $X_j^{i\prime} = X_i^i + 1 = X_k^{i\prime}$. If $i \notin Ind(e_n)$, we get $X_j^{i\prime} = max\{X_l^i \mid l \in Ind(t)\} = X_k^{i\prime}$. The property 2 holds in both subcases.

Case $j, i \in Ind(e_n)$ **and** $k \notin Ind(e_n)$**:** Here, $p_i(lkc_j(C_n)) = p_i(post(e_n))$ is updated and $p_i(lkc_k(C_n)) = p_i(lkc_k(C_{n-1}))$ remains unchanged. By Lemma 5.3 $p_i(lkc_k(C_n)) < p_i(post(e_n))$. The counter X_i^i is maximal in $\equiv_i$, since the token i participates in all of its own transitions (and by induction assumption). Therefore, $X_j^{i\prime} = X_i^i + 1 > X_k^i = X_k^{i\prime}$ and the property 2 holds.

Case $j \in Ind(e_n)$ **and** $i, k \notin Ind(e_n)$**:** Here, $p_i(lkc_j(C_n)) = \max_{\leq}\{p_i(lkc_l(C_{n-1})) \mid l \in Ind(e_n)\}$ and $X_j^{i\prime} = \max\{X_l^i \mid l \in Ind(e_n)\}$ are updated, and $p_i(lkc_k(C_n)) = p_i(lkc_k(C_{n-1}))$ and $X_k^{i\prime} = X_k^i$ remain unchanged. By induction assumption, the causal relation coincides with the relation between $\max\{X_l^i \mid l \in Ind(t)\}$ and X_k^i. Thus, the property 2 holds. □

Now, we define strategies where each player uses its last known marking as memory. This means, that a player decides equally in two conditions if its last known markings are the same in these two conditions. A strategy can be interpreted as a finite state automaton where its state is its last known marking.

LKM-Strategy. Let T_{N_0} be the LTS of a Petri game G_0 with token-partitioning net N_0. A *last known marking strategy* σ of G_0 is a family of mappings: $\sigma = \{\sigma_i \mid i \in I_P\}$, such that $\sigma_i : LKS_i(N_0) \to 2^{\mathcal{T}}$ and for all $lks \in LKS_i$ holds that $\sigma_i(lks) = \mathcal{T}$ if $p_i(lks) \in \mathcal{P}^E$.

Player i's strategy σ_i determines for every possible last known marking a set of transitions that are allowed to be fired. A LKM-strategy is independent from counter equivalence classes. If and only if all players in a preset of a transition allow that transition, it is added to the branching process of the LTS-strategy. Purely environmental transitions (all players in the preset are environment players) cannot be restricted.

Branching process of a LKM-strategy. Let σ be a LKM-strategy of a Petri game G_0. The branching process $B(\sigma)$ is defined as the branching process B such that

$$\forall e \in \mathcal{T}_{u_0} : e \in \mathcal{T} \Leftrightarrow \forall e' \in [e] : (\forall i \in I_P : \forall p_i \in pre_{u_0}(e') \cap P_i : h_{u_0}(e') \in \sigma_i(lkm_{u_0}(p_i))).$$

Winning LKM-strategy. An LTS-strategy σ of a Petri game G is a winning strategy if and only if $B(\sigma)$ is a winning strategy in G. A LKM-strategy can be interpreted as a restriction of the memory LTS obtained by deleting transitions and states.

Example 5.11 (Winning LKM-strategy). For the Petri game "Same Choice" in Fig. 2.2 a winning LKM-strategy is given by $\sigma = \{\sigma_0, \sigma_1\}$, where for all $M \in lks_0 : \sigma_0 = \mathcal{T}$ and $\sigma_1(In) = \{c_a, c_b\}$, $\sigma_1(\{S_1, A\}) = \{a'\}$ and $\sigma_1(\{S_1, B\}) = \{b'\}$. The LTS belonging to σ is obtained by deleting the states $q_{a,b'}$ and $q_{b,a'}$ in the LTS in Fig. 5.2 ◁

Summary. In this section, we showed that we can construct a finite LTS that keeps track of the last known markings of each token in a branching process. The last known marking is a suitable choice for the memory in a Petri game, as we will see on the benchmark results in Section 5.3. Generally, this approach is not limited to last known markings. We conjecture that this approach is adaptable to keep track of any finite part of a causal history. For example, one could keep track of a sequence of last known markings for each token. It is also possible that each token keeps track of the sequences of last known markings for itself *and* for every other token. Any finite part of a causal history defined through last known markings can be stored. Then, the causal relation of the conditions from which the labels are stored for the last known markings need to be stored too. It is also possible to store only specific parts of a finite prefix, e.g. all last known markings that contain a specific place or specific subsequences of last known markings.

This construction opens up Petri games to a broad field of winning conditions. For a logic with decidable model checking problem on nets, any automata-based model checking approach can be directly applied for a given strategy with finite causal memory (e.g. LKM-strategies). This is possible because a memory LTS and a LKM-strategy (and generalizations of those) define a finite space of a strategy, which can be checked by a model checking algorithm. Since there are only finitely many strategies with a fixed finite memory, deciding whether there exists such a winning strategy is decidable by checking every possible strategy. To obtain an effective algorithm, checking if a strategy is winning must be combined with the search for such a strategy. For LKM-strategies, a simple combination encoding the existence of a winning strategy as a SAT-Problem is done later on in Section 5.3.

Generally, the synthesis of any system, in particular a distributed system, suffers a state explosion problem. For a single given strategy, checking if it is winning is a model checking problem. For model checking, there exist many well-working approaches that reduce the state space, e.g. partial order reduction using stubborn sets [Val90], symbolic model checking using binary decision diagrams (BDDs) [BCM$^+$92] or counterexample guided abstraction refinement (CEGAR) [CGJ$^+$00]. Combining these approaches with the search for winning strategies is at least non-trivial. However, this is the first setting using causal memory where this seems feasible.

In the next section, we propose a generalization of the LKM-memory. There, an additional net is used to store a finite part of the causal history. This approach to bounded synthesis differs from previous approaches [FS13], where the finite memory is an LTS, or [Hec21], where it is part of the system player's strategy to match suitable causal histories when to repeat a strategy.

5.2 Memory Net Structures

We start with a motivating example where the last known marking memory is insufficient to find a winning strategy but an alternative finite memory is sufficient.

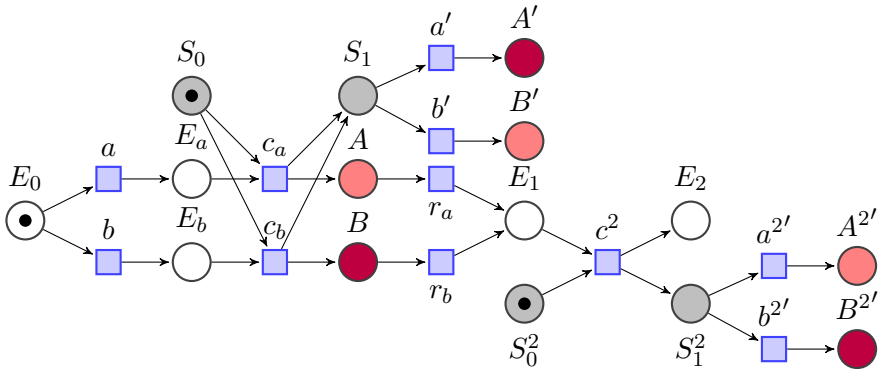


Figure 5.3: The Petri game "Double Same Choice". Any marking with two places in red or two places in purple is a bad marking.

Example 5.12 (LKM-memory insufficient). The Petri game "Double Same Choice" in Fig. 5.3 is very similar to the Petri game "Same Choice" in Fig. 2.2. The difference is an extension with another system player in S_0^2. This player communicates with the environment player (transition c^2) and has to repeat the choice (a or b) of the environment player to win the game. There are additional bad markings to force the second system player to repeat the choice. In the "Same Choice" game the bad markings are $\{A, B'\}$ and $\{B, A'\}$. Here, any marking that contains $\{A, B'\}$, $\{B, A'\}$, $\{B', A^{2'}\}$ or $\{A', B^{2'}\}$ is a bad marking. Therefore, the system player in S_1^2 needs to copy the decision of the system player in S_1 or a bad marking is reached.

The initial choice of the environment player is not captured by the last known marking of the second system player (starting on condition S_0^2) since the place of the environment player in the last known marking of the second system player on S_1^2 after firing c^2 is E_2 in both cases (a and b). Thus, no winning LKM-strategy exists for the game "Double Same Choice" although there exist a winning strategy. ◁

In the following, we present an approach where the finite memory stores any finite part of the causal history. Then, for the game "Double Same Choice", it is possible to define a finite memory in a way that the second system player knows the choice of the environment player and can choose accordingly.

Informally speaking, the finite memory is captured by an additional net (N_A in the following) that "listens" to the net of a game (N)), i.e. every time a transition t is fired in N a transition in N_A which listens to t fires to alter the state of the finite memory.

Definition 5.13 (Memory net structure). For a Petri net N, a *memory net structure* is a tuple $S = (N, N_A, l)$, where N_A is a token-partitioning Petri net with a partition P_A and l is a mapping $l : \mathcal{T}_A \to 2^{\mathcal{T}}$. The net N_A of a memory net structure $S = (N, N_A, l)$ is called a *memory net*. ◁

The memory net N_A is an observer of the net N, i.e. the net N_A does not hinder the behaviour of the net N and there is no synchronisation between the nets. However,

the state space considered later on is the product state space of the net N which is its reachability graph and the state space of N_A that is reachable by observing N which is its LKM-memory LTS. The concept of an observing automata is a broadly used concept [SSL+95, BFLM10, BHJP04].

In a memory net structure S, l maps every transition of the memory net to a set of transitions in N. This means, that a transition in the memory net listens to a set of transitions in the original Petri net. Every time a transition is fired in the net N, a transition that listens to that transition is fired in the memory net. If multiple different transitions in N_A listen to the same transition that is fired in the net N only one of those different transition is fired in the memory net. This is done due to simplicity reasons. One could allow multiple transitions listening at once. Note that there can be transitions in N where no transition in N_A listens to.

The memory net N_A of a memory net structure S is a token-partitioning Petri net while the net N does not need to have a fixed number of tokens. There is no dependency between the number of tokens in the Petri net and the number of tokens in the memory net. Therefore, there is a need to specify for each place in N, which token in N_A carries the memory in that place. For that, we introduce a memory allocation function $alloc : \mathcal{P} \to I_{P_A}$ that assigns each place of N a token index in N_A. Generally, the memory of a place does not need to be exactly one token in the memory net, it could also be more than one token or even no memory. For simplicity, we keep the memory of a place to be the memory of one token in the memory net.

Example 5.14 (Memory net structure). In Fig. 5.4, is an example of a memory net structure S, where N is the net of the "Double Same Choice" game. The memory net N_A only has the transitions a^A, b^A, c_a^A, c_b^A and c^{2A} where $l(t^A) = \{t\}$ for all these transitions.

For the memory net structure in Fig. 5.4, the partition of N is given by $P = P_0 \dot\cup P_1 \dot\cup P_2$, where $P_0 = \{E_0, E_a, E_B, A, B, E_1\}$, $P_1 = \{S_0, S_1, A', B'\}$, $P_2 = \{S_0^2, S_1^2, A^{2'}, B^{2'}\}$ and the partition of N_A is $P_A = P_{A_0} \dot\cup P_{A_1} \dot\cup P_{A_2}$, where $P_{A_0} = \{E_0^A, E_a^A, E_b^A\}$, $P_{A_1} = \{S_0^A\}$, $P_{A_2} = \{S_0^{2A}\}$. A suitable allocation is $alloc(p) = \begin{cases} 0 & \text{if } p \in P_0 \\ 1 & \text{if } p \in P_1 \\ 2 & \text{if } p \in P_2 \end{cases}$. ◁

In the following, the Definition 5.9 of a memory LTS is generalized to memory net structures, where the memory of each token in the memory net is its last known marking. Beforehand, some definitions are introduced.

For a memory net structure S, the reversal function of l is defined as $l^{-1}(t) = \{t^A \in \mathcal{T}_A \mid t \in l(t^A)\}$. For every $t \in \mathcal{T}$ the set $l^{-1}(t)$ is the set of transitions listening to t. For a marking $M^A \in \mathcal{R}(N_A)$, the reversal function restricted to M^A is defined as $l^{-1}_{M^A}(t) = \{t^A \in \mathcal{T}_A \mid t \in l(t^A) \wedge pre(t^A) \subseteq M^A\}$. This is necessary to determine which transitions in the memory net N_A listen to a transition t in the net N when the marking M_A is reached in N_A. We formalize that exactly one transition that listened is fired in N_A if there is such a transition. This is not a necessary restriction. However, it is done here due to simplicity reasons.

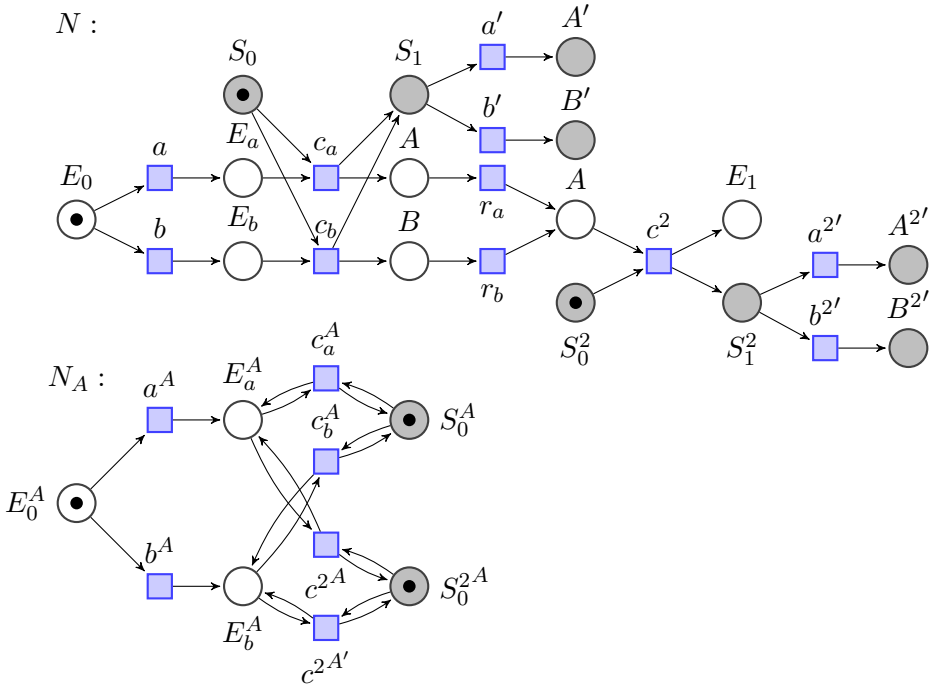


Figure 5.4: A memory net structure (N, N_A, l), where l is given by $l(t^A) = \{t\}$ for all $t \in \mathcal{T}_A$.

We define memory sequences of a memory net structure that are pairs of firing sequences (s_N, s_{N_A}), where s_{N_A} is the firing sequence of transitions that listened to a transition in s_N.

Definition 5.15 (Memory sequence). A *memory sequence* of a memory net structure $S = (N, N_A, l)$ is a pair (s_N, s_{N_A}), where $s_N = t_1 \dots t_n$, is a firing sequence in N and $s_{N_A} = t_1^A \dots t_k^A$, $In_A = M_0^A \xrightarrow{t_1^A} \dots \xrightarrow{t_k^A} M_k^A$, $n \geq k \geq 0$, is a firing sequence in N_A, such that $\exists f : \{1, \dots, k\} \to \{1, \dots, n\}$ strictly monotonically increasing

1. $\forall i \in \{1, \dots, k\} : t_i^A \in l^{-1}(t_{f(i)})$
2. $\forall i \in \{1, \dots, k-1\} : \forall j \in \{f(i)+1, \dots, f(i+1)-1\} : l^{-1}_{M_{i-1}^A}(t_j) = \emptyset$.
3. if $k \geq 1$: $\forall j \in \{1, \dots, f(1)-1\} : l^{-1}_{M_0^A}(t_j) = \emptyset$
4. if $k \geq 1$: $\forall j \in \{f(k)+1, \dots, n\} : l^{-1}_{M_k^A}(t_j) = \emptyset$
5. if $k = 0$: $\forall j \in \{1, \dots, n\} : l^{-1}_{M_0^A}(t_j) = \emptyset$

The set of all memory sequences of S is denoted as $Seq(S)$. ◁

Example 5.16 (Memory sequence). For the memory net structure in Fig. 5.4, $(s_N, s_{N_A}) = (ac_a r_a c^2 a', a^A c_a^A c^{2A})$ is a memory sequence. Property 1 of Def. 5.15 states that every transition t_i^a of s_{N_A} listens to the transition $t_{f(i)}$ in s_N. The properties 2–5 ensure that the firing sequence in the memory net N_A has a transition for every transition in the net N where at least one transition listened to which is enabled in the marking M_i^A reached in N_A so far. Property 2 states that for two succinct transitions t_i^A and t_{i+1}^A there are no transitions listening to any transition in s_N between $t_{f(i)}$ and $t_{f(i+1)}$, e.g. $(ac_a r_a c^2, a^A c^{2A})$ is not a memory sequence due to 2 in Fig. 5.4 because c_a^A listens to c_a. The properties 3-5 cover the remaining edge cases. For example in the memory net structure of Fig. 5.4, (a, ε) is not a memory sequence due to 5, (ac_a, a^A) is not a memory sequence due to 4, and (ac_a, c_a^A) is not a memory sequence due to 3. ◁

The definition of the LTS of a memory net structure follows.

Definition 5.17 (Memory net LTS). The *memory net LTS* $T_S = (Q, q_I, \Sigma, \delta)$ of a memory net structure $S = (N, N_A, l)$ is defined as:

- $Q = \{(M, q) \mid \exists (s_N, s_{N_A}) \in Seq(S) : In|s_N\rangle M \wedge q_{N_A} \xrightarrow{s_{N_A}} q\}$ is the set of states where q_{N_A} is the initial state of the LKM-memory LTS T_{N_A} of N_A.
- $q_I = (In, q_{N_A})$ is the initial state.
- $\Sigma = \mathcal{T}$ is the set of inputs.
- $((M, q), t, (M', q')) \in \delta$ if and only if $M|t\rangle M'$ and $q \xrightarrow{t_a} q'$ if $t_a \in l^{-1}_{M(q)}(t)$ or $q = q'$ if $l^{-1}_{M(q)}(t) = \emptyset$, where $M(q)$ is the marking in state q.

◁

Example 5.18 (Memory net LTS). In Fig. 5.5, an example of a memory net structure and its LTS is shown. There, the net N is the net of the game "Same Choice" without the reset transitions r_a and r_b. The LTS T_S of a memory net structure S is a finite state space of the memory net structure, where the last known marking of every token in N_A is tracked. Again the listening function is defined as $l(t^A) = \{t\}$ for all $t^A \in \mathcal{T}_A$. In Fig. 5.5, the first line of a state tracks the reached marking in the net N while the second line depicts the state of the LKM-memory LTS T_{N_A} of the net N_A. ◁

A memory net structure does not define the memory a player of a Petri game has access to. This is done by an allocation function $alloc : \mathcal{P} \to I_{P_A}$ which determines the memory in each place of a game. For example in Fig. 5.5, all places have the memory of token 0 in N_A since it is the only token in N_A. However, it is not necessary to restrict the allocation function to exactly one token. Nevertheless, it is done here for simplicity.

A memory net structure and an allocation function can be chosen in a way such that a system player in a Petri game has more information than it is available in its own causal history. A Petri game as introduced here does not specify a memory net structure. It only consists of a Petri net, a partitioning of the places in system and environment places and a winning condition. The memory of a player is its whole causal history in

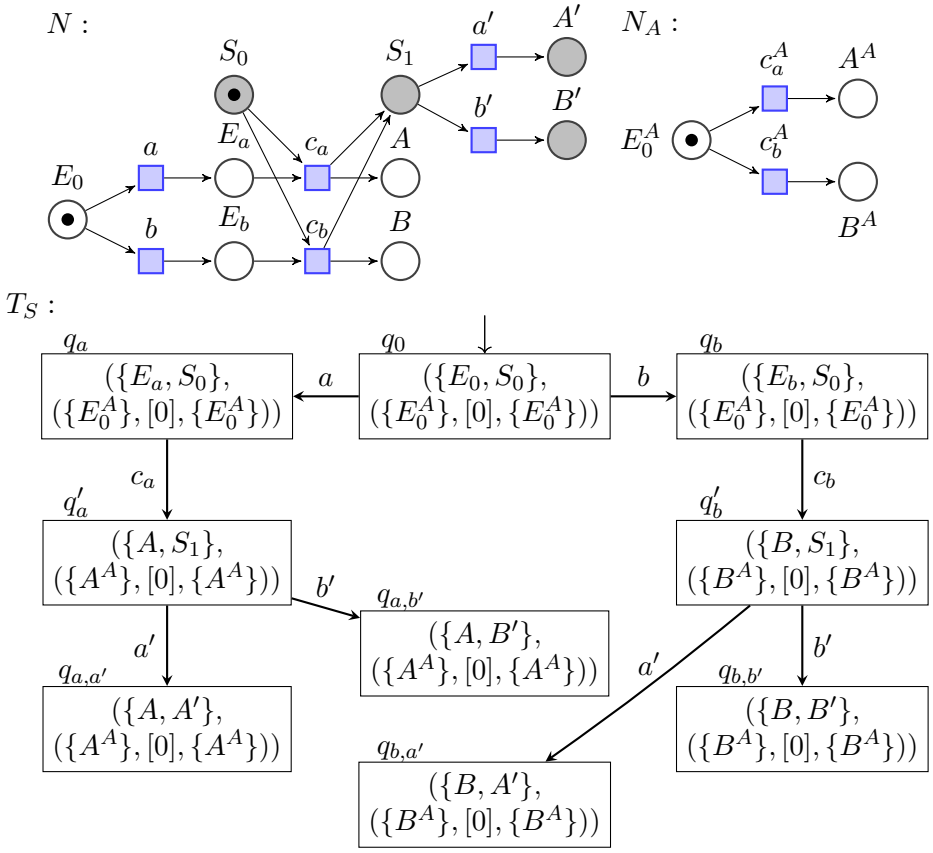


Figure 5.5: The LTS T_S of the memory net structure $S = (N, N_A, l)$, where $l(c_a^A) = \{c_a\}$ and $l(c_b^A) = \{c_b\}$.

a Petri game (it is played on the unfolding of the net of the game). In the following, we characterize pairs of memory net structures and allocation functions such that the memory only depends on the own causal history of a system player.

A memory net structure S is called *trace-closed* if for all memory sequences $(s_N^1, s_{N_A}^1)$, $(s_N^2, s_{N_A}^2)$ holds that if s_N^1 and s_N^2 are linearizations of the same trace in N the firing sequences $s_{N_A}^1$ and $s_{N_A}^2$ are in the same trace in N_A.

So, to apply the construction of an LTS of a memory net structure to a Petri game, we restrict the pairs of the memory net structures and allocation functions to those, such that every player in the Petri game only uses its own causal history. Given a trace-closed memory net structure S, the memory sequence of a configuration C in u is defined as $mems(C) = (s_N, s_{N_A})$, such that $C = Cfg_u(s_N)$ is the configuration of s_N. We refer to the components of $mems(C)$ as $s_N(C)$ and $s_{N_A}(C)$.

Definition 5.19 (Consistent memory net structure). A memory net structure S and allocation function *alloc* are *consistent with a Petri game* G if

- the memory net is trace-closed and
- for all finite configurations C, $C' \subseteq \mathcal{T}_u$ for all conditions $b \in Cut(C) \cap Cut(C')$ it holds $lkc_{alloc(h(b))}(Cut(Cfg_{u_A}(s_{N_A}(C)))) = lkc_{alloc(h(b))}(Cut(Cfg_{u_A}(s_{N_A}(C'))))$.

$\lhd$

The second property of Def. 5.19 looks complicated, but it is not. The term $Cut(Cfg_{u_A}(s_{N_A}(C)))$ starts with the configuration C in the unfolding u of N. Then, $s_{N_A}(C)$ is second component of the memory sequence $mems(C)$, i.e. an event sequence in u_{N_A}. Afterwards, Cfg_{u_A} yields the configuration of that event sequence. At last, the last known cut of token $alloc(h_u(b))$ in the cut of that configuration in N_A is determined. Informally speaking, this property ensures that a condition b has the same memory state (the last known cut of its token assigned via *alloc*) in all cuts that b can be part of. This means, that the assigned token only takes the events in the causal history of condition b into account, i.e. the assigned memory of b only depends on the causal history of b.

We extend the definition of an LKM-strategy to memory net structures.

Memory net strategy. Let T_S be the LTS of a memory net structure S and $alloc : \mathcal{P} \to I_{P_A}$ a memory allocation function. A *memory net strategy* σ of G is a family of mappings: $\sigma = \{\sigma_i \mid i \in I_P\}$, where $\sigma_i : \{p \in \mathcal{P} \mid alloc(p) = i\} \times LKS_i(N_A) \mapsto 2^{\mathcal{T}}$ and $\sigma_i((p, lks^A)) = \mathcal{T}$ if $p \in \mathcal{P}^E$, $lks^A \in LKS_i(N_A)$.

The mapping σ_i of a memory net strategy σ determines for every place, that has token i allocated as memory, which transitions are allowed to be fired depending on the last known marking of token i in the memory net.

The branching process of a memory net strategy is defined analogously to the branching process of a LKM-strategy Again, if and only if all players in a preset of a transition allow that transition, it is added to the branching process of the memory net strategy and purely environmental transitions (all players in the preset are environment players) cannot be restricted.

Branching process of a memory net strategy. Let σ be a memory net strategy of a Petri game G, and S and *alloc* a consistent memory net structure and allocation function. The branching process $B(\sigma)$ is defined as the branching process B_σ, such that

$$\forall e \in \mathcal{T}_u : e \in \mathcal{T}_\sigma \Leftrightarrow \forall e' \in [e] : (\forall b \in pre_u(e') : h_u(e') \in \sigma_{alloc(h_u(b))}((h_u(b), lkm_{alloc(h_u(b))}(Cut(Cfg_{u_A}(s_{N_A}([b]_u)))))).$$

The term $Cut(Cfg_{u_A}(s_{N_A}([b]_u)))$ starts with the local configuration of condition b in the unfolding u of N. Then, $s_{N_A}([b]_u)$ is second component of the memory sequence $mems([b]_u)$, i.e. an event sequence in u_{N_A}. Afterwards, C_{u_A} yields the configuration of that event sequence. At last, the last known marking of token $alloc(h_u(b))$ in the cut of that configuration in N_A is determined.

Winning memory net strategy. A memory net strategy σ of a Petri game G is a winning strategy if and only if $B(\sigma)$ is a winning strategy in G.

Example 5.20 (Memory net strategy). The memory net N_A in Fig. 5.5 has only one token, i.e. $I_{P_A} = \{0\}$. Using the allocation function $alloc(p) = 0$ for all $p \in \mathcal{P}$, a winning memory net strategy is given by $\sigma = \{\sigma_0\}$, where $\sigma_0((S_1, \{A^A\})) = \{a'\}$ and $\sigma_0((S_1, \{B^A\})) = \{b'\}$, and $\sigma_0((p, lks^A)) = \mathcal{T}$ for all other $p \in \mathcal{P}$, $lks^A \in lks_0(N_A)$. ◁

5.2.1 Refinement of the Memory Model

In the previous section, pairs of memory net structures and allocation function are restricted to pairs that are consistent with a given Petri game. However, it is desirable to specify a memory that is not restricted to the causality in a net. There are a few desirable properties for a memory in different scenarios.

For example in Fig. 5.6, a memory net structure is given, where a sender on the place S_0 sends a message which is received by the memory net. The process on place P_0 has the token in N_A allocated as its memory. The process does not need to communicate synchronously with the sender to receive the message and may process the message after its memory has received it.

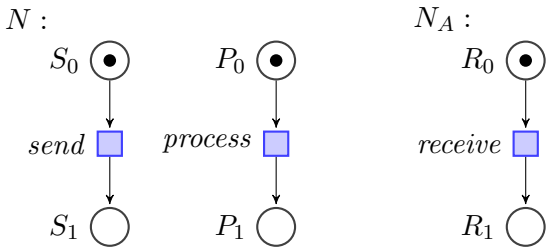


Figure 5.6: A memory net structure with asynchronous communication, $l(receive) = \{send\}$.

Another example of a refinement of the causal memory model is shown in Fig. 5.7. Here, the transition t in the memory net listens to both transitions t_1 and t_2 in N, such that these transitions are indistinguishable for any process using the token in the memory net as allocated memory. Furthermore, it is possible that a transition resets parts of the memory.

Overall, the presented construction is significantly more versatile than the original model using purely causal memories. The major benefit of this construction is that the refinements of the memory model do not increase the time complexity bound as the introduction of partial observation usually comes with.

5.3 Implementation

We encode the existence of a winning LKM-strategy of a game with token-partitioning net as a SAT solving problem. The encoding can be adapted to memory net strategies for memory net structures and allocation functions which are consistent to the Petri game.

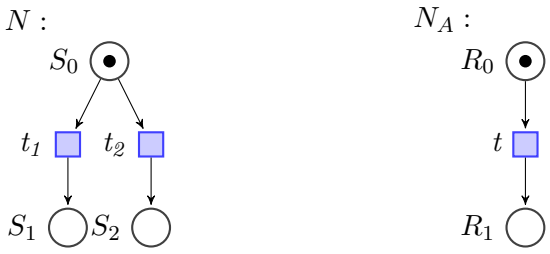


Figure 5.7: A memory net structure with indistinguishable transitions t_1 and t_2, $l(t) = \{1, 2\}$.

Each boolean assignment of the encoding defines a strategy. Before defining the encoding ϕ_G of the existence of a winning LKM-strategy, we introduce several auxiliary variables. The set of boolean variables is $var(\phi_{T_G}) = \{lkm_i(t) \mid lkm_i \in LKS_i(N), t \in \mathcal{T}\}$.

The variable $lkm_i(t)$ encodes that transition t is allowed in all conditions of player i with last known marking lkm if and only if this variable is true. We require $lkm_i(t) = true$ if $p_i(lkm) \in \mathcal{P}^E$, i.e. an environment condition allows all events. For all outgoing transitions t of a state $q = (lkm_1, \ldots, lkm_\nu, \equiv_1, \ldots, \equiv_\nu, M)$ we define $t^q = \bigwedge_{P_i \cap pre(t) \neq \emptyset} lkm_i(t)$. The set of variables of t^q is denoted as $var(t^q) = \{lkm_i(t) \mid lkm_i(t) \text{ is a conjunct of } t^q\}$. For a transition sequence of $s = t_1 \ldots t_n$ of N, the state reached after firing $t_1 \ldots t_k$, $k \leq n$, in T_N is denoted as $q_k(s)$.

At last, we introduce auxiliary variables for the three winning properties that must hold in a state q: B_q for safety, D_q for determinism, and A_q for deadlock avoidance.

$$B_q = \begin{cases} true & M(q) \notin \mathcal{B} \\ false & \text{else} \end{cases}$$

$$D_q = \bigwedge_{(q,t,q') \in \delta \wedge p \in pre(t) \cap \mathcal{P}^S} (t_q \Rightarrow \bigwedge_{(q,t',q'') \in \delta \wedge p \in pre(t) \wedge t' \neq t} \neg t'_q)$$

$$A_q = \begin{cases} true & \neg \exists (q, t, q') \in \delta \\ \bigvee_{(q,t,q') \in \delta} t_q & \text{else} \end{cases}$$

Note that we always assume an empty conjunction to be true. The *justified refusal* property is always satisfied by the unfolding of an LKM-strategy since a decision of a strategy is interpreted as a set of transitions that is always allowed (in those places with the same finite memory state).

Now, we formalize in which states the of memory LTS the formulae B_q, D_q and A_q must hold. For that, the set of all traces $Paths(T_N)$ that do not loop in the memory LTS is considered. A trace does not loop, if no linearization of that trace repeats a state in the memory LTS. The idea is that the winning properties must hold in a state q if all transitions in a linearization of a trace leading to q are allowed by the strategy (i.e. the

boolean assignment). Formally,

$$Paths(T_N) = \{\pi \mid \pi \text{ is a finite trace of } N \wedge \\ \forall s \in \pi : \forall j,k \in \{1,\dots,|s|\} : j \neq k \Rightarrow q_j(s) \neq q_k(s).\}$$

The set of variables of a trace π is defined as $var(\pi) = \bigcup_{s\in\pi}(\bigcup_{k\in\{1,\dots,|s|\}} var(t^{q_k(s)}))$, which is the set of all variables needed to fire all transitions of the trace in the memory LTS. This set does not change, if only one linearization s of the trace π is considered in the outer set union. The state reached after firing any linearization of a trace π is denoted as $q(\pi)$. Note that $q(\pi)$ does not depend on which linearization is fired.

Definition 5.21 (SAT-encoding). Let T_N be the LTS of a Petri game G and $Paths(T_N)$ the set of traces in T_N without a loop. The encoding ϕ_G is

$$\phi_G = \bigwedge_{\pi\in Paths(T_N)} \Big(\bigwedge_{v\in var(\pi)} \Rightarrow B_{q(\pi)} \wedge D_{q(\pi)} \wedge A_{q(\pi)}\Big)$$

◁

The LKM-strategy of a boolean assignment $\mathcal{A} : var(\phi_G) \to \{true, false\}$ of the variables in ϕ_G is defined as follows:

$$\forall i \in I_P : \forall lkm \in LKS_i(N) : \sigma_i(lkm) = \begin{cases} \{t \in \mathcal{T} \mid \mathcal{A}(lkm_i(t)) = true\} & \text{, if } p_i(lkm) \in \mathcal{P}^S \\ \mathcal{T} & \text{, if } p_i(lkm) \in \mathcal{P}^E \end{cases}$$

Example 5.22 (SAT-encoding). The encoding of the "Same Choice" game from Fig. 2.2 when reduced to its relevant terms is

$$\begin{aligned} \phi_G = & In_1(c_a) \wedge In_1(c_b) \\ & \wedge (In_1(c_a) \Rightarrow ((\{S_1,A\}_1(a') \vee \{S_1,A\}_1(b')) \wedge (\neg\{S_1,A\}_1(a') \vee \neg\{S_1,A\}_1(a'))) \\ & \wedge (In_1(c_b) \Rightarrow ((\{S_1,B\}_1(a') \vee \{S_1,B\}_1(b')) \wedge (\neg\{S_1,B\}_1(a') \vee \neg\{S_1,B\}_1(a'))) \\ & \wedge ((In_1(c_a) \wedge \{S_1,A\}_1(b')) \Rightarrow false) \\ & \wedge ((In_1(c_b) \wedge \{S_1,B\}_1(a')) \Rightarrow false) \end{aligned}$$

Referring to the states of the memory LTS in Fig. 5.2, the first line is due to the deadlock avoidance property in q_a and q_b (D_{q_a} and D_{q_b}). The antecedent in the second line is the path to the state q'_a. The first conjunct of the consequent is due to the deadlock avoidance property in q'_a and the second conjunct is due to the determinism property in q'_a. The third line is analogously to the second line for the state q'_b. Line 4 and 5 encode the paths to the bad markings. ◁

In the following, it is sketched that the satisfiability of the SAT-Encoding coincides with the existence of a winning LKM-strategy.

Lemma 5.23 (LTS to SAT). *The SAT-formula ϕ_G is satisfiable if and only if there is a winning LKM-strategy σ in G.*

Proof sketch. Lemma 5.10 states that the memory LTS tracks the last known markings correctly. The formulas B_q, A_q and D_q encode the winning properties that must hold in a state q of the memory LTS. If the winning properties B_q, D_q and A_q are met in a state q of the memory LTS, the winning conditions (Def. 2.26) are met in all cuts of the branching process of the LKM-strategy, where the reached marking and the last known markings of each token are the same as in the state of the memory LTS. Given a boolean assignment, the encoding ensures that the winning properties hold in every state that can be reached in the memory LTS under the given assignment. Thus, the winning conditions hold in every cut of the branching process of the LKM-strategy. □

We give an upper bound of the complexity determining if there exists a winning LKM-strategy. We could not find a matching lower bound. A trivial lower bound is NP since the Petri game used in Lemma 3.10 has a winning strategy if and only if it has a winning LKM-strategy. We conjecture that a lower bound is at least exponential time (EXP) since there exist Petri nets of polynomial size and tokens which have exponentially many last known markings.

Lemma 5.24 (Size of memory LTS). *The memory LTS T_N of a net N has at most exponentially many states in the number of tokens ν and polynomially many states in the number of places $|\mathcal{P}|$.*

Proof. The LTS T_N has exponentially many states: there are at most $|\mathcal{P}|^\nu$ different reachable markings in a safe net and at most 3^{ν^2} equivalence classes of counters. The exponent is ν^2 since we have at most ν^2 pairs of counters X_j^i, X_k^i (for one equivalence class) and and the base is 3 since their relation is $X_j^i < X_k^i$, $X_j^i = X_k^i$ or $X_j^i > X_k^i$ (this is an over approximation). Since there are ν last known markings and ν equivalence classes in a state, the LTS has at most $|Q| \leq |\mathcal{P}|^{\nu^2} \cdot 3^{\nu^3}$ states. □

We continue by investigating the number of traces in $Paths(T_N)$.

Lemma 5.25 (Number of traces). *For a memory LTS T_N, the set $Paths(T_N)$ has exponential size in the number of states in T_N, $|Q|$, and in the number of inputs of T_N, $|\mathcal{T}|$.*

Proof. There are at most $|Q| \cdot (|Q|)! \cdot (|\mathcal{T}|)!$ many input sequences in T_N where no state is reached twice. The first factor is $|Q|$ for the length of the input sequence. The second factor is $(|Q|)!$ as an over approximation of the permutations of the states. The third factor is $(|\mathcal{T}|)!$ as an over approximation of the permutations of the inputs. Since there are at most as many traces as input sequences without a loop, the size of the set $|Paths(T_N)|$ is bounded by $|Q| \cdot (|Q|)! \cdot (|\mathcal{T}|)! < |Q| \cdot 2^{(|Q|^2+|\mathcal{T}|^2)}$. □

Overall, the existence of a winning LKM-strategy can be determined within nondeterministic double exponential time (2-NEXP).

Lemma 5.26 (Existence of LKM-strategy). *The existence of an LKM-strategy of a Petri game G with net N can be determined in 2-NEXP in the number of tokens ν.*

Proof. By Lemma 5.24, the memory LTS has at most exponentially many states in the number of players ν. By Lemma 5.25, there at most exponentially many traces in $Paths(T_N)$ in the number of states $|Q|$ in T_N, which results in double exponentially many traces in the number of tokens ν.

Since the length of the SAT encoding is linear in the number of traces in $Paths(T_N)$ and SAT is in NP, the resulting complexity is in 2-NEXP. □

The 2-NEXP complexity is significantly worse than the EXP complexity of the results in [FO17] where Petri games with an arbitrary bounded number of system players and a single environment player are considerer. There, a matching lower bound in EXP is established, i.e. the synthesis problem for those Petri games is EXP-complete. However, the winning condition used in [FO17] is a local safety condition and the generalization of the results in [FGHO22] to a global safety condition requires double exponential time (2-EXP). So in[FGHO22], Petri games with an arbitrary bounded number of system players and a single environment equipped with a global safety condition, the bad markings, take double exponential time to be solved. Here, we allow an arbitrary bounded number of system players *and* and arbitrary number of environment players while having a global safety condition as winning condition. Thus, the 2-NEXP bound here is good considering the number of players which are allowed. However, opposed to [FGHO22] our approach cannot rule out the existence of a winning strategies if no winning strategy is found. In [FGHO22] , no matching lower bound was found either but the lower bound in EXP from [FO17] holds.

5.3.1 Experimental Results

The benchmark results on various scalable Petri games are presented. Firstly, the results are compared to the bounded synthesis approach in [Hec21] in Table 5.8. Secondly, the results are compared to the results of the tool ADAM [Gie22] in Table 5.9. The descriptions of the benchmark families are slightly modified descriptions of those in [Gie22, Hec21]. The benchmark results finding an LKM-strategy were obtained on an 8th gen. 1.8 GHz quad-core Intel i7 processor with 16GB RAM.

Benchmark families. Alarm system (AS). There are n distinct locations. Every location is secured by an alarm system. There is one burglar that may intrude an arbitrary location. The alarm systems can inform each other about burglaries. The goal is that no alarm system is triggered without an intrusion and all alarm systems indicate the correct intrusion point in case of a burglary.
Parameters: n alarm systems.

Concurrent Machines (CM). There are n machines which should process m orders. The orders can be processed concurrently, but no machine is allowed to process more than one order. The hostile environment chooses one machine to be defective. The goal is that all orders are processed.
Parameters: n machines and k orders.

Document Workflow (DW). A document is circularly passed on by the clerks. The environment decides which clerk receives the document first. The goal is that all clerks sign the document.
Parameters: n clerks

Disjoint Routing (DR). There are n paths and the goal is to route n packets disjointly.
Parameters: n packets.

Production Line (PL). There are n concurrent robots which are able to repair or ignore n features of a product. Depending on the product chosen by the environment, some features need to be repaired while others must not be repaired.
Parameters: n robots and also n features.

The implementation of the LKM-memory approach handles only token-preserving Petri games. The benchmark families CM and DR are not token-preserving in their original version. These were slightly modified by adding dummy places every time a token is consumed or created. The implementation is a stand-alone python-script that can be found here `https://github.com/phannibal5/Petri-Game-LKM-Solver`. It comes with a stand-alone generator script for the benchmark families. The instructions on how to use the scripts are in the readme file.

The implementation of the LKM-memory approach is optimized in the sense that it uses a memory net structure where the memory net is the same net as the original net where transitions that are not relevant are cut out. For example, the games of the DW benchmark do not need any LKM-memory. In fact, there exists a winning strategy that uses only the own place as memory for the games of the DW benchmark. Furthermore, the implementation encodes the SAT-Problem directly into a CNF-formula.

For all benchmark families, the SAT encoding is at most exponentially long since the number of paths without loops is polynomially bounded in every benchmark family opposed to the exponential worst case.

In Table 5.8, the bounded synthesis approach in [HM19] and the approach presented here are compared. In [HM19], two approaches for bounded synthesis are compared and the better performing result is stated in every cell. In most cases, the approach in [HM19] which uses an encoding based on true concurrency performs better. Roughly speaking, the true concurrency encoding checks the traces of a Petri game instead of the transition sequences.

Time is measured in seconds and the timeout is at 1800 seconds. A strategy was found in every case where there is a time. Neither the bounded synthesis approach nor the LKM-memory approach have a limitation on the number of environment players and system players.

Observation Table 5.8. The LKM-memory approach outperforms the bounded synthesis approach in every case except for the Document Workflow benchmark with 12 clerks. There, both time out. The LKM-memory approach starts with significantly less time needed, but the time of the LKM-memory approach increases faster than the time of the bounded synthesis approach.

Benchmark	Parameter	time LKM-memory	time bounded synthesis
AS	2	0.003	11.15
	3	0.027	timeout
	4	0.382	-
	5	9.854	-
	6	341.673	-
	7	timeout	-
DR	2	0.003	6.05
	3	0.028	10.07
	4	1.361	65.31
	5	84.317	timeout
	6	timeout	-
PL	1	0.001	5.59
	2	0.002	5.85
	3	0.005	6.95
	4	0.014	12.54
	5	0.049	41.95
	6	0.226	742.36
	7	1.034	timeout
	8	4.753	-
	9	20.601	-
	10	87.547	-
	11	450.803	-
	12	timeout	-
DW	1	0.001	5.79
	2	0.001	6.44
	3	0.003	7.80
	4	0.013	11.22
	5	0.054	16.59
	6	0.173	26.71
	7	0.731	49.69
	8	3.026	90.41
	9	11.356	239.47
	10	40.802	716.61
	11	187.407	1304.14
	12	timeout	timeout

Figure 5.8: Comparison to the bounded synthesis approach in [Hec21].

Interpretation The bounded synthesis approach uses a QBF-Encoding while the LKM-memory approaches encodes the state space explicitly. Usually, encoding a state space symbolically is largely beneficial. The faster increase in time of the LKM-memory approach might be due to this. Both approaches are applicable independently of the number of system and environment players. Also, both approaches either find a winning strategy or no statement about the existence of such a strategy is possible. The results are encouraging for the presented approach.

Observation Table 5.9. In Table 5.8, the tool ADAM [Gie22] and the LKM-memory approach are compared. Again, the time is measured in seconds and the timeout is at 1800 seconds. There are two additional columns (compared to Table 5.8) stating if there exists a winning LKM-strategy and if there exists a winning strategy at all. Opposed to the LKM-memory approach the tool ADAM is applicable only to Petri games with at most one environment player. Solving Petri games with at most one environment player and an arbitrary number of system players is EXP-complete [FO17]. The worst case complexity of the LKM-memory approach is 2-NEXP. However, the worst case does not occur within the benchmark families. All games of the benchmark families are solvable in NEXP. The double exponential complexity is caused by the exponential size of the memory LTS followed by potentially exponentially many traces in the memory LTS. Here, there are only as many traces as there are states in the memory LTS such that the complexity stays in NEXP.

The tool ADAMSYNT [Gie22] is slower for small instances of the benchmark games, but outperforms the LKM-memory approach for larger instances and achieves more results within the timeout of 1800 seconds. The game "Double Same Choice" in Fig. 5.3 is an example of a game where no winning LKM-strategy exists but a winning strategy exists. However, for every game in the benchmark families there exists a winning LKM-strategy if there exists a winning strategy.

Interpretation. The SAT-Encoding of the presented approach finding LKM-strategies performs worse overall compared to the tool ADAM [Gie22]. This is expected since the complexity of finding a LKM-strategy is NEXP for the benchmark families (worst case 2-NEXP) opposed to the EXP complexity solving the synthesis problem for Petri games with a bounded number of system players and one environment player [FO17]. Furthermore, the tool ADAM uses BDDs (binary decision diagrams) opposed to the approach here building the state space explicitly. The symbolic approach is expected to perform better than the explicit approach in a rapidly growing state space. On the other hand, this leaves a potential optimization of the approach finding LKM-strategies.

5.4 Related Work

Bounded synthesis for Petri games searches for winning strategies within b-bounded unfoldings, which limit the number of copies of each place to a given natural number b [Fin15a]. This method supports arbitrarily many system and environment players and

Benchmark	Par.	time LKM-mem.	time ADAM	ex. LKM-strat.	ex. strat.
AS	2	0.003	1.14	✓	✓
	3	0.027	1.70	✓	✓
	4	0.382	4.11	✓	✓
	5	9.854	13.60	✓	✓
	6	341.673	139.92	✓	✓
	7	timeout	535.96	?	✓
	8	-	timeout	?	?
DW	1	0.001	0.68	✓	✓
	2	0.001	0.77	✓	✓
	⋮	...	...	...	...
	11	187.407	7.90	✓	✓
	12	timeout	7.48	✓	✓
	⋮	...	...	...	...
	31	-	317.82	?	✓
	32	-	timeout	?	?
CM	2-1	0.001	0.64	✓	✓
	2-2	0,002	0.58	✗	✗
	3-1	0.003	0.80	✓	✓
	3-2	0.008	1.12	✓	✓
	3-3	0.029	0.92	✗	✗
	4-1	0.016	1.05	✓	✓
	4-2	0.053	1.34	✓	✓
	4-3	0.251	2.74	✓	✓
	4-4	1.185	8.49	✗	✗
	5-1	0.107	1.27	✓	✓
	5-2	0.512	2.02	✓	✓
	5-3	3.340	4.94	✓	✓
	5-4	19.527	40.75	✓	✓
	5-5	104.466	1469.70	✗	✗
	6-1	0.925	1.74	✓	✓
	6-2	5.230	3.46	✓	✓
	6-3	46.509	12.78	✓	✓
	6-4	361.268	179.34	✓	✓
	6-5	timeout	timeout	?	✓
	7-1	9.886	2.70	✓	✓
	7-2	68.552	9.32	✓	✓
	7-3	1092.068	60.74	✓	✓
	7-4	timeout	timeout	?	?

Figure 5.9: Comparison to the tool ADAM from [FGO15, Gie22].

encodes the problem into quantified Boolean formulas (QBFs), solvable by tools like QuAbS [Ten16]. If a QBF is satisfiable, the solver yields a winning strategy; if not, it provides counterexamples to explain the absence of such a strategy.

We compare our results to the two algorithms implemented in AdamSYNT [FGHO17]. One of these algorithms is the bounded synthesis described in [Hec21]. The other algorithm is based on the decidability result of Petri games with a single environment player [FO17]. We refer to the latter algorithm as unbounded synthesis.

For the bounded synthesis algorithm, two algorithms generating bounded unfoldings are presented in [Hec21], depending on whether a loop exists in the graph of reachable markings. Without any loop, McMillan's [McM95] algorithm is applied which constructs the entire unfolding; with loops, the unfolding uses the bound to limit the number of places. Then, if another copy of a place would be made in the (normal) unfolding it is part of the system players' strategy to choose the place from the set of existing copies to continue the strategy. These are the two algorithms we compare our results to implemented in AdamSYNT [FGHO17]. We refer to our approach as the LKM-memory approach.

In comparisons, unbounded synthesis solves more cases, but bounded synthesis tends to produce smaller strategies due to its memory constraints [FGHO17]. However, unbounded synthesis benefits from more mature, optimized algorithms.

The LKM-memory approach finds slightly less strategies than the unbounded approach if the unbounded approach is applicable, i.e. the Petri game has at most one environment player. For Petri games of small size, the LKM-memory approach outperforms the unbounded synthesis. This might be due to the overhead of the symbolic approach of the unbounded synthesis using binary decision diagrams (BBDs).

Compared to the bounded synthesis approach of [Hec21], which is also applicable for any bounded number of environment and system players, the LKM-memory approach finds more winning strategies restricting each players' causal memory to its last known marking in the memory net. We observe that our approach is much faster for small Petri games, but the time needed increases faster with growing size of the Petri game. This might be due to explicitly constructing the state space of the memory.

5.5 Outlook

The memory LTS and the memory net LTS are constructed given a Petri net or a memory net structure, respectively. It is detached from the winning condition of a Petri game. This allows for generalization of the winning condition of a Petri game. For example, one could use linear temporal logic (LTL, [Pnu77]), computation tree logic (CTL, [CE81, CES86], or CTL*, [EH86]) or even hyper-LTL, [CFK$^+$14, FRS15], to express the winning condition. Since the model checking problem of those logics are decidable over safe Petri nets, model checking if a single strategy with finite memory is winning is decidable. Since there are only finitely many strategies given a memory net structure the synthesis problem is decidable in those cases. Furthermore, a memory net structure allows to refine the causal memory model without increasing the complexity as

presented in Subsection 5.2.1. As mentioned before, the finite part of the causal history can be any finite part and does not need to be the last known markings.

Overall, the presented setting is the first setting for the synthesis of asynchronous distributed systems that is decidable and comes with a huge expressive power while keeping the complexity at a reasonable level. It also allows to adapt existing techniques to tackle the state explosion problem. For example, partial order reduction techniques for LTL on Petri nets might be applied when constructing a memory net LTS or when searching for a winning strategy. Applying those techniques might be highly non-trivial and there is a lot of future work exploring the capabilities. However, the state explosion problem is expected to remain a bottleneck.

Apart from synthesis, we strongly conjecture that the memory LTS can also be used to refine a Petri net in a natural way. The goal here is to construct a refined finite(!) Petri net, where every place is a pair consisting of its original place and its last known marking, i.e. constructing a bounded unfolding up to a bound where a finite part of the causal history (e.g. the last known marking) repeats. The refined Petri net can be constructed in a way that its unfolding is isomorphic to the unfolding of the original Petri net. However, constructing such a refined finite Petri net is non-trivial due to situations where a pair of a place and its last known marking occur in multiple states of the memory LTS as in the memory LTS of the net N_0 in Fig. 5.1. Then, it is necessary to have multiple places of the same pair of place and last known marking in the refined net.

Chapter 6

Conclusions

We addressed the synthesis problem for asynchronous distributed systems with causal memory via Petri games. The main positive decidability result is the decidability of 4-player Petri games within nondeterministic exponential time. Opposed to existing results [FG18, FO17] solving Petri games in deterministic exponential time, we do not restrict the type of the players. Only one environment player is allowed in [FO17] and only one system player is allowed in [FG18]. In contrast, we allow any combination of environment and system players, especially the previously uncovered case of two system and two environment players. This extends the kinds of asynchronous distributed systems that can be synthesized using Petri games.

We adapted the algorithm for constructing a complete finite prefix [ERV02] to construct winning prefixes. These prefixes are complete in the sense that all relevant behaviour to win the Petri game is contained such that a winning strategy is constructed by repeating parts of the prefix.

The undecidability result for 6-player Petri games equipped with a local safety condition is achieved by a two-step reduction. Starting with the undecidable ω-Post correspondence problem [Fin15b, Gir86, Ruo85] a reduction to a tiling problem of an infinite plane is followed by a reduction of this tiling problem to 6-player Petri games. This result is similar to the undecidability result of 6-process control games [Gim22] because the theoretical findings in one model are transferable to the other with an exponential overhead [BFH19]. These results imply that the synthesis problem for 6-player Petri games is also undecidable for more complex winning conditions.

To avoid undecidability, we presented an approach to bounded synthesis in which we limit the part of the causal history that is tracked. So, the players make their decisions solely based on that bounded part and cannot distinguish between situations where the causal histories differ but the bounded parts are equivalent. The experimental results look promising for our approach compared to existing algorithms [FGO15, HM19]. However, state space explosion remains a bottleneck. One significant advantage of our approach is its versatility, in two respects. It can be combined with more complex winning conditions such as linear temporal logic, and we can specify which part of the causal history is used as the memory of the players. Furthermore, we are not limited to a causal memory

model but can retain it if desired. Adding non-causal parts in the memory model does not increase the complexity bound.

6.1 Future Work

The synthesis problem for 5-player Petri games equipped with a local (or global) safety condition remains an open question. If we had to make a guess, we would conjecture that it is undecidable. Regarding the decidability result for 4-player Petri games, a precise lower bound on its complexity is desirable. Furthermore, the exact complexity of 3-player Petri games could be investigated.

The approach using only a bounded memory opens up Petri games to more complex winning conditions. The construction of the state space is decoupled from the winning condition. This allows straightforward combinations with other winning conditions since the product state space of the finite memory and the model checking of the winning condition is fairly easy to construct. However, efficient algorithms that combine existing approaches of reducing the state space, like stubborn sets [Val90] or counterexample guided abstraction refinement [CGJ+00], are yet to be found.

Bibliography

[ABP21] Federica Adobbati, Luca Bernardinello, and Lucia Pomello. A two-player asynchronous game on fully observable Petri nets. *Trans. Petri Nets Other Model. Concurr.*, 15:126–149, 2021.

[BCM+92] Jerry R. Burch, Edmund M. Clarke, Kenneth L. McMillan, David L. Dill, and L. Jean Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.

[BF88] Eike Best and César Fernández. *Nonsequential Processes - A Petri Net View*, volume 13 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1988.

[BFH19] Raven Beutner, Bernd Finkbeiner, and Jesko Hecking-Harbusch. Translating asynchronous games for distributed synthesis. In Wan J. Fokkink and Rob van Glabbeek, editors, *30th International Conference on Concurrency Theory, CONCUR 2019*, volume 140 of *LIPIcs*, pages 26:1–26:16. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019.

[BFLM10] Patricia Bouyer, Uli Fahrenberg, Kim G. Larsen, and Nicolas Markey. Timed automata with observers under energy constraints. In *Proceedings of the 13th International Conference on Hybrid Systems: Computation and Control (HSCC)*, pages 61–70. ACM, 2010.

[BHJP04] Johan Blom, Anders Hessel, Bengt Jonsson, and Paul Pettersson. Specifying and generating test cases using observer automata. In *Proceedings of the 4th International Workshop on Formal Approaches to Testing of Software (FATES 2004)*, volume 3395 of *Lecture Notes in Computer Science*, pages 125–139. Springer, 2004.

[BL69] J. Richard Büchi and Lawrence H. Landweber. Solving sequential conditions by finite-state strategies. *Transactions of the American Mathematical Society*, 138:295–311, 1969.

[CE81] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Logics of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer, 1981.

[CES86] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 8(2):244–263, 1986.

[CFK+14] Michael R. Clarkson, Bernd Finkbeiner, Masoud Koleini, Kristopher K. Micinski, Markus N. Rabe, and Cesar Sanchez. Temporal logics for hyperproperties. In *2014 IEEE 27th Computer Security Foundations Symposium (CSF)*, pages 265–279. IEEE, 2014.

[CGJ+00] Edmund M. Clarke, Orna Grumberg, Somesh Jha, Yuan Lu, and Helmut Veith. Counterexample-guided abstraction refinement. In *Proceedings of the 12th International Conference on Computer Aided Verification (CAV)*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2000.

[Chu62] Alonzo Church. Logic, arithmetic and automata. In *Proceedings of the International Congress of Mathematicians*, pages 23–35, 1962.

[Coo71] Stephen A. Cook. The complexity of theorem-proving procedures. In Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman, editors, *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, pages 151–158. ACM, 1971.

[EH86] E. Allen Emerson and Joseph Y. Halpern. Sometimes and not never revisited: On branching versus linear time temporal logic. *Journal of the ACM*, 33(1):151–178, 1986.

[EH08] Javier Esparza and Keijo Heljanko. *Unfoldings: A Partial-Order Approach to Model Checking*. Monographs in Theoretical Computer Science. An EATCS Series. Springer, 2008.

[Eng91] Joost Engelfriet. Branching processes of Petri nets. *Acta Inf.*, 28(6):575–591, 1991.

[ERV02] Javier Esparza, Stefan Römer, and Walter Vogler. An improvement of McMillan's unfolding algorithm. *Formal Methods Syst. Des.*, 20(3):285–310, 2002.

[FG18] Bernd Finkbeiner and Paul Gölz. Synthesis in Distributed Environments. In Satya Lokam and R. Ramanujam, editors, *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017)*, volume 93 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 28:1–28:14. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2018.

[FGHO17] Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. Symbolic vs. bounded synthesis for petri games. In Dana Fisman and Swen Jacobs, editors, *Proceedings Sixth Workshop on Synthesis, SYNT@CAV 2017, Heidelberg, Germany, 22nd July 2017*, volume 260 of *EPTCS*, pages 23–43, 2017.

[FGHO22] Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and E.-R. Olderog. Global winning conditions in synthesis of distributed systems with causal memory. In Florin Manea and Alex Simpson, editors, *30th EACSL Annual Conference on Computer Science Logic, CSL 2022*, volume 216 of *LIPIcs*, pages 20:1–20:19. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022.

[FGO15] Bernd Finkbeiner, Manuel Gieseking, and Ernst-Rüdiger Olderog. Adam: Causality-based synthesis of distributed systems. In Daniel Kroening and Corina S. Păsăreanu, editors, *Computer Aided Verification*, pages 433–439, Cham, 2015. Springer International Publishing.

[Fin15a] Bernd Finkbeiner. Bounded synthesis for Petri games. In Roland Meyer, André Platzer, and Heike Wehrheim, editors, *Correct System Design – Symposium in Honor of Ernst-Rüdiger Olderog on the Occasion of His 60th Birthday, Proceedings*, volume 9360 of *Lecture Notes in Computer Science*, pages 223–237. Springer, 2015.

[Fin15b] Olivier Finkel. The exact complexity of the infinite Post correspondence problem. *Inf. Process. Lett.*, 115(6-8):609–611, 2015.

[FO17] Bernd Finkbeiner and Ernst-Rüdiger Olderog. Petri games: Synthesis of distributed systems with causal memory. *Information and Computation*, 253:181–203, 2017.

[FO24] Bernd Finkbeiner and Ernst-Rüdiger Olderog. Ten years of Petri games. In Nils Jansen, Sebastian Junges, Benjamin Lucien Kaminski, Christoph Matheja, Thomas Noll, Tim Quatmann, Mariëlle Stoelinga, and Matthias Volk, editors, *Principles of Verification: Cycling the Probabilistic Landscape - Essays Dedicated to Joost-Pieter Katoen on the Occasion of His 60th Birthday, Part III*, volume 15262 of *Lecture Notes in Computer Science*, pages 399–422. Springer, 2024.

[FRS15] Bernd Finkbeiner, Markus N. Rabe, and César Sánchez. Algorithms for model checking hyperltl and hyperctl*. In *Proceedings of the 27th International Conference on Computer Aided Verification (CAV 2015)*, volume 9206 of *Lecture Notes in Computer Science*, pages 30–48. Springer, 2015.

[FS05] Bernd Finkbeiner and Sven Schewe. Uniform distributed synthesis. In *Proceedings of the 20th IEEE Symposium on Logic in Computer Science (LICS)*, pages 321–330. IEEE, 2005.

[FS13] Bernd Finkbeiner and Sven Schewe. Bounded synthesis. *International Journal on Software Tools for Technology Transfer*, 15(5):519–539, Oct 2013.

[GGMW13] Blaise Genest, Hugo Gimbert, Anca Muscholl, and Igor Walukiewicz. Asynchronous games over tree architectures. In Fedor V. Fomin, Rusins Freivalds, Marta Z. Kwiatkowska, and David Peleg, editors, *Automata, Languages, and Programming - 40th International Colloquium, ICALP 2013, Proceedings, Part II*, volume 7966 of *Lecture Notes in Computer Science*, pages 275–286. Springer, 2013.

[Gie22] Manuel Gieseking. *Correctness of Data Flows in Asynchronous Distributed Systems: Model Checking and Synthesis*. PhD thesis, University of Oldenburg, Germany, 2022.

[Gim17] Hugo Gimbert. On the control of asynchronous automata. In Satya V. Lokam and R. Ramanujam, editors, *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 2017)*, volume 93 of *LIPIcs*, pages 30:1–30:15. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2017.

[Gim22] Hugo Gimbert. Distributed asynchronous games with causal memory are undecidable. *Log. Methods Comput. Sci.*, 18(3), 2022.

[Gir86] Françoise Gire. Two decidability problems for infinite words. *Inf. Process. Lett.*, 22(3):135–140, 1986.

[GSZ09] Paul Gastin, Nathalie Sznajder, and Marc Zeitoun. Distributed synthesis for well-connected architectures. *Formal Methods Syst. Des.*, 34(3):215–237, 2009.

[Han23] Paul Hannibal. (Un)decidability bounds of the synthesis problem for Petri games. In Antonis Achilleos and Dario Della Monica, editors, *Proceedings of the Fourteenth International Symposium on Games, Automata, Logics, and Formal Verification, GandALF 2023, Udine, Italy, 18-20th September 2023*, volume 390 of *EPTCS*, pages 115–131, 2023.

[Hec21] Jesko Hecking-Harbusch. *Synthesis of asynchronous distributed systems from global specifications*. PhD thesis, Saarland University, Saarbrücken, Germany, 2021.

[HLO26] Paul Hannibal, Dennis Lisiecki, and Ernst-Rüdiger Olderog. Finite-memory strategies for Petri games. In Martin Fränzle, Jürgen Niehaus, and Bernd Westphal, editors, *Engineering Safe and Trustworthy Cyber Physical Systems: Essays Dedicated to Werner Damm on the Occasion of His 71st Birthday*, pages 183–200, Cham, 2026. Springer Nature Switzerland.

[HM19] Jesko Hecking-Harbusch and Niklas O. Metzger. Efficient trace encodings of bounded synthesis for asynchronous distributed systems. In Yu-Fang Chen, Chih-Hong Cheng, and Javier Esparza, editors, *Automated Technology for Verification and Analysis - 17th International Symposium, ATVA 2019, Taipei, Taiwan, October 28-31, 2019, Proceedings*, volume 11781 of *Lecture Notes in Computer Science*, pages 369–386. Springer, 2019.

[HO22] Paul Hannibal and Ernst-Rüdiger Olderog. The synthesis problem for repeatedly communicating Petri games. In Luca Bernardinello and Laure Petrucci, editors, *Application and Theory of Petri Nets and Concurrency - 43rd International Conference, PETRI NETS 2022, Proceedings*, volume 13288 of *Lecture Notes in Computer Science*, pages 236–257. Springer, 2022.

[McM95] Kenneth L. McMillan. A technique of state space search based on unfolding. *Formal Methods Syst. Des.*, 6(1):45–65, 1995.

[MT02] P. Madhusudan and P. S. Thiagarajan. A decidable class of asynchronous distributed controllers. In Lubos Brim, Petr Jancar, Mojmír Kretínský, and Antonín Kucera, editors, *CONCUR 2002 - Concurrency Theory, 13th International Conference, Proceedings*, volume 2421 of *Lecture Notes in Computer Science*, pages 145–160. Springer, 2002.

[MTY05] P. Madhusudan, P. S. Thiagarajan, and Shaofa Yang. The MSO theory of connectedly communicating processes. In Ramaswamy Ramanujam and Sandeep Sen, editors, *FSTTCS 2005: Foundations of Software Technology and Theoretical Computer Science, 25th International Conference, Proceedings*, volume 3821 of *Lecture Notes in Computer Science*, pages 201–212. Springer, 2005.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 46–57. IEEE, 1977.

[PR89] Amir Pnueli and Roni Rosner. On the synthesis of an asynchronous reactive module. In *Proceedings of the 16th International Colloquium on Automata, Languages and Programming (ICALP 1989)*, volume 372 of *Lecture Notes in Computer Science*, pages 652–671. Springer, 1989.

[PR90] Amir Pnueli and Roni Rosner. Distributed reactive systems are hard to synthesize. In *31st Annual Symposium on Foundations of Computer Science, Volume II*, pages 746–757, 1990.

[Rei85] Wolfgang Reisig. *Petri Nets: An Introduction*, volume 4 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1985.

[Ruo85] Keijo Ruohonen. Reversible machines and Post's correspondence problem for biprefix morphisms. *J. Inf. Process. Cybern.*, 21(12):579–595, 1985.

[SF06] Sven Schewe and Bernd Finkbeiner. Synthesis of asynchronous systems. In *Logic-Based Program Synthesis and Transformation (LOPSTR 2006)*, volume 4407 of *Lecture Notes in Computer Science*, pages 127–142. Springer, 2006.

[SSL+95] Meera Sampath, Raja Sengupta, Stéphane Lafortune, Kasim Sinnamohideen, and Demosthenis Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on Automatic Control*, 40(9):1555–1575, 1995.

[Ten16] Leander Tentrup. Non-prenex QBF solving using abstraction. In Nadia Creignou and Daniel Le Berre, editors, *Theory and Applications of Satisfiability Testing - SAT 2016 - 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*, volume 9710 of *Lecture Notes in Computer Science*, pages 393–401. Springer, 2016.

[Val90] Antti Valmari. A stubborn attack on state explosion. *Formal Methods in System Design*, 1(4):297–322, 1990.

[Vog91] Walter Vogler. Executions: A new partial-order semantics of Petri nets. *Theor. Comput. Sci.*, 91(2):205–238, 1991.

[WCHP24] Nick Würdemann, Thomas Chatain, Stefan Haar, and Lukas Panneke. Taking complete finite prefixes to high level, symbolically. *Fundam. Informaticae*, 192(3-4):313–361, 2024.

[Wür21] Nick Würdemann. Exploiting symmetries of high-level petri games in distributed synthesis. *it Inf. Technol.*, 63(5-6):321–331, 2021.

[Wür24] Nick Würdemann. *Taking Synthesis of Distributed Systems via Petri Games to High Level, Symbolically.* PhD thesis, University of Oldenburg, Germany, 2024.

[Zie87] Wieslaw Zielonka. Notes in finite asynchronous automata. *RAIRO – Theoretical Informatics and Applications – Informatique Théorique et Applications*, 21(2):99–135, 1987.

List of Figures

List of Algorithms

Index

Symbol Overview

Petri nets, branching processes and traces		
Petri net	$N = (\mathcal{P}, \mathcal{T}, pre, post, In)$	7
Place	$p \in \mathcal{P}$	7
Transition	$t \in \mathcal{T}$	7
Set of reachable markings	$\mathcal{R}(N)$	8
Branching process	$B = (N, h)$	9
Condition	$b \in \mathcal{P}$	10
Event	$e \in \mathcal{T}$	10
Node	$x \in \mathcal{P} \cup \mathcal{T}$	8
Causal successor relation	$\leq$	8
Concurrent	$\|\|$	8
Conflict	$\#$	8
Unfolding of a net N_0	$u = (N_u, h_u)$	11
Homomorphism	h	9, 10
Isomorphic to	$\cong$	10
Subprocess relation	$\sqsubseteq$	10
Subprocess homomorphism	$h^B_{B'} : \mathcal{P} \cup \mathcal{T} \to \mathcal{P}' \cup \mathcal{T}'$	10
Configuration	C	11
Cut	K	11
Configuration of a cut	$Cfg(K)$	11
Cut of a configuration	$Cut(C)$	11
Local configuration	$[x]$	12
Last known cut	$lkc(x)$	12
Last known marking	$lkm(x)$	12
Future	$Fut(B, K)$ or $Fut(B, C)$	12
Finite extension	$\oplus$	12
Adequate order	$\preceq$	12
Subprocess isomorphism in unfolding	$h^{Fut(u,[e])}_{Fut(u,[e'])} = I^{[e]}_{[e']}$	12
Partition	P	20
Indices of partition	$I_P = \{1, \ldots, \nu\}$	20
Token projection	p_i, $i \in I_P$	20
Indices of transition/event	$Ind : \mathcal{T} \to 2^{I_P}$	20
Trace	π	23
Firing sequence	s	11
Event sequence of firing sequence	$events(s)$	11
Trace of firing sequence	$\overline{s}$	24
Trace of configuration	$\pi(C)$	24
Configuration of trace	$Cfg(\pi)$	24
Possible extensions	$PE(B)$	44