Saarland University
Faculty of Natural Sciences and Technology I
Department of Computer Science

Bachelor's Thesis

A Game-Based Semantics for CSP

submitted by
Jesko Hecking-Harbusch

submitted on
Mai 5th, 2015

Supervisor
Prof. Bernd Finkbeiner, Ph.D.

Advisor
Prof. Bernd Finkbeiner, Ph.D.

Reviewers
Prof. Bernd Finkbeiner, Ph.D.
Prof. Dr. Ernst-Rüdiger Olderog

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Statement in Lieu of an Oath

I hereby confirm that I have written this thesis on my own and that I have not used any other media or materials than the ones referred to in this thesis.

Einverständniserklärung

Ich bin damit einverstanden, dass meine (bestandene) Arbeit in beiden Versionen in die Bibliothek der Informatik aufgenommen und damit veröffentlicht wird.

Declaration of Consent

I agree to make both versions of my thesis (with a passing grade) accessible to the public by having them added to the library of the Computer Science Department.

Saarbrücken, _____ _____

(Datum/Date) (Unterschrift/Signature)

Abstract

Communicating Sequential Processes (CSP) which dates back to Hoare, 1985, facilitates the modeling of concurrent processes and the analysis of the behavior of concurrent processes. CSP provides a textual representation of processes which communicate via actions. Existing semantics transform a CSP expression into a transition system representing the possible behaviors. As desirable feature in CSP, it should be possible to indicate processes whether they are controlled by the system or by the environment, respectively.

Petri nets model decisions during a concurrent execution explicitly. They visualize which processes take part in a synchronous transition. In a Petri net, processes are modeled by tokens in places. CSP and Petri nets have been connected via several semantics. Petri games extend Petri nets by explicitly marking places as belonging to the environment or to the system. Environment places are assumed to behave non-deterministically whereas the system places are controlled by a global strategy. This stipulation characterizes a safety game where the system attempts to avoid certain places in order to win the game.

In this Bachelor's thesis, a game-based semantics for CSP is presented. This semantics shows how to define a system and an environment player for CSP in order to develop a game model to answer the realizability question and the synthesis problem for the system's strategy. A CSP expression is converted into a Petri game. Therefore, the syntax of CSP is modified slightly to model system and environment. Rules for a structural operational semantics are presented which allow the derivation of the transitions of the Petri game.

Acknowledgments

I am very grateful to Prof. Bernd Finkbeiner for generously offering me this interesting and challenging topic. Especially, I am thankful for the aspiring guidance, invaluably constructive criticism, and friendly advice during the last months. I deeply appreciated the many and illuminating views on a number of issues related to my thesis.
I would also like to thank Prof. Ernst-Rüdiger Olderog for his contribution and feedback to the thesis. I am very grateful for his offer to review the thesis.
Moreover, I also place on record, my sincere thank you to my family, friends, and fellow students for their support. Especially, I want to give a warm thanks to David Gembalczyk and Sebastian Schirmer for proofreading the thesis. Nevertheless, all remaining errors are mine alone.

Contents

1 Introduction

Communicating Sequential Processes (CSP) [4] is a formal language for the modeling and description of concurrent processes. It introduces a notation which describes processes by their ability to communicate via actions. The processes are able to decide between different actions to communicate. Furthermore, it is possible to synchronize several processes with each other. This results in an action only occurring if particular processes communicate the action at the same time. The behavior of a CSP expression can be defined by a transition system consisting of edges labeled by actions and nodes representing the current state of all processes. The transformation of an expression following the CSP syntax into a transition system is based on a *structural operational semantics (SOS)* [13].

In CSP, it is not possible to mark processes as belonging to the system or to the environment. This distinction is worthwhile because it can be used to define games [14]. The system player controls all processes belonging to the system whereas the processes belonging to the environment remain uncontrollable. The goal of such a game is to decide whether a winning strategy for the system exists (*realizability question*) and, if such a strategy exists, to construct it automatically (*synthesis problem*). These games are named after the condition which has to be achieved by a winning strategy. In a safety game for example, the environment player aims at reaching certain bad places whereas the system player tries to avoid these places.

In this thesis, a game-based semantics for CSP is presented which describes the concurrent processes as a *Petri game* [2]. Petri games define games as introduced above on *Petri nets* [1].

A Petri net models a distributed system in which processes interact with each other. Processes are represented by tokens which reside in places. Places are connected via transitions which define the possible flow of tokens through the net. If more than one place precedes and follows a transition, respectively, then the transition models a synchronization. The difference to CSP is that the synchronization of processes is modeled explicitly in a Petri net. For each transition, it is clearly visible which places take part in it. Petri nets are delineated as a directed bipartite graph of places and transitions. The two theories are connected because both can be used to describe processes and they only differ in the way they are notated which affects synchronization.

The extension made by Petri games marks every place to either belong to the global system player or to the global environment player. Every token represents a local system player or a local environment player depending on to which global player the place belongs the token resides in. The global player controls all local players. A local environment player is explicitly

modeled in the actions which a token possibly can perform at an environment place but the decision which actions are actually taken is decided non-deterministically. Therefore, the environment remains uncontrollable. Local system players are controlled by the global strategy which decides which transition to perform. Petri games are by definition safety games. This is achieved by marking certain places as bad behavior. The goal of the environment is to reach such bad places whereas the system has to avoid these places in order to win. The places are bad from the system's point of view. A strategy is called winning if it can assure to never reach a bad places regardless of the behavior of the environment. There can be several local system players and several local environment players if there exist enough tokens which are all controlled by the respective global player.

A game is a natural way to model a distributed system which interacts with the environment because the distinction between the system and the environment is explicit. Furthermore, the synthesis problem challenges us to automatically derive winning strategies which represent a correct implementation for the model. A CSP expression can also model the interaction between processes and based on the resulting transition system it is possible to see all possible developments. Nevertheless, it is not possible to automatically derive strategies due to the missing classification of processes to belong either to the system or to the environment. This underspecification prevents the definition of a game. For example, a meaningful vending machine can only be defined if the behavior of the costumers is modeled. The same holds for a distributed alarm system which is hard to design without making assumptions about the burglar's behavior.

The main part of this thesis introduces a game-based semantics for CSP which results in Petri games. This defines a game model for CSP which can be used for synthesis. Realizability and synthesis questions can be answered for certain Petri games [2]. Therefore, a subset of the syntax of CSP is extended in order to deal with the explicit division into system and environment. Rules of the structural operational semantics enable the transformation to Petri games. For each used operator of CSP, a set of rules is given to derive the corresponding transitions. The semantics shows that the deterministic and non-deterministic choice of CSP coincide with the decisions on places belonging to the system or to the environment, respectively.

This thesis is structured as follows: In Section 2, all relevant definitions of Petri nets, Petri games, and CSP used in the following are provided. The new semantics for CSP is defined in Section 3. In Section 4, the semantics is illustrated by examples. In Section 5, related work is discussed and in Section 6 conclusions are drawn and future work is addressed.

2 Background

In this section, the necessary background for the remainder of this thesis is introduced. In Section 2.1, Petri nets are defined. In the next subsection, they are extended to games by the definition of Petri games. In Section 2.3, Communicating Sequential Processes (CSP) becomes established.

2.1 Petri Nets

Petri nets [1] are used to model distributed systems. They consist of places, represented by circles, and of transitions, represented by bars. Places hold tokens which represent parts of a distributed system. Transitions enable tokens to flow through the Petri net by defining arrows between the circles and bars. They illustrate the possible developments of parts of the distributed system. A transition can be used if all places preceding it hold the required amount of tokens. When using the transition the tokens in all places preceding it are removed and tokens are added to all places after the transition.

2.1.1 Definition of Petri Nets

A *Petri net* $\mathscr{N}$ is a tuple $(\mathscr{P}, \mathscr{T}, \mathscr{F}, In)$, where:

- $\mathscr{P}$ is a finite non-empty set of *places*.
- $\mathscr{T}$ is a finite non-empty set of *transitions*.
- $\mathscr{F}$ as a multiset over $(\mathscr{P} \times \mathscr{T}) \cup (\mathscr{T} \times \mathscr{P})$ is the *flow relation*.
- $In : \mathscr{P} \to \mathbb{N}_0$ is the *initial marking*.

The set of places and the set of transitions are disjoint ($\mathscr{P} \cap \mathscr{T} = \emptyset$). The initial marking describes the initial distribution of tokens. It defines the number of tokens for every place (including zero if the place is empty). The progress of a Petri net can be described by *markings* $M_i : \mathscr{P} \to \mathbb{N}_0$. A marking M_i works like the initial marking In but describes a distribution of tokens after i transitions were fired starting from the initial marking. This implies that $M_0 = In$ holds.

2.1.2 Enabledness and Firability

The use of transitions is called *firing*. The flow relation $\mathscr{F}$ defines via the pairs $(\mathscr{P} \times \mathscr{T})$ in which places tokens are necessary in order to fire a transition. A transition is called *enabled* if all preceding places hold the required

amount of tokens. When a transition is fired all preceding tokens are consumed and new tokens are produced based on the pairs $(\mathscr{T} \times \mathscr{P})$ of $\mathscr{F}$. A joint transition between several places is called *synchronization*.

As $\mathscr{F}$ is a multiset it is possible to consume or produce more than one token from or in a place, respectively. If $\mathscr{F}$ contains the pair (p, t) once then one arrows from the place p to the transition t exists. If $\mathscr{F}$ contains the pair (p, t) more than once then for every occurrence one arrow would be produced. This case is abbreviated to one arrow annotated by a number n which is equal to the number of occurrences of the pair (p, t) in $\mathscr{F}$. The arrow then represents that n tokens are required in p in order to fire t. This implies that an arrow without a number represents that one token is required. Pairs of the form (t, p) representing the production of tokens are treated in an analog manner.

2.1.3 Example Modeling of a Vending Machine

In Fig. 1 (a), two tokens are delineated, where one depicts a person at place A and the other a machine at place M. The person can decide whether she wants a cup of coffee or a cup of tea via the transitions *decCoffee* and *decTea*, respectively. Only these two transitions are enabled and only one of them can be fired depending on the decision of the person.

After firing the transition *decTea* the marking of the Petri net changes to the one displayed in panel (b) of Fig. 1 representing that the person has decided to order a cup of tea. Only the joint transition *tea* is enabled. It represents that the person orders and gets a cup of tea from the machine. After this transition the machine returns to its original position because the arrow from M to *tea* is bidirectional. The machine is ready to serve the next costumer despite of only one costumer being modeled in this example. The person is finished with ordering and reaches the place in the right corner representing that she received a cup of tea.

The only way to recognize A as a person is the explanation in the text above. The person represents the environment in a game-theoretic sense because the vending machine should work properly for all behaviors of the person. The machine represents the system in this setting. The game between M and A can lead to a decision whether a correct vending machine exists. The usage of CSP instead of a Petri net would suffer from the same problem. With the introduction of Petri games in the next section the distinction between system and environment can be given explicitly for Petri nets and with the game-based semantics introduced in Section 3 the distinction can be given explicitly for CSP. The example is extended in Fig. 2 in the next subsection showing that the system has to perform internal decisions.

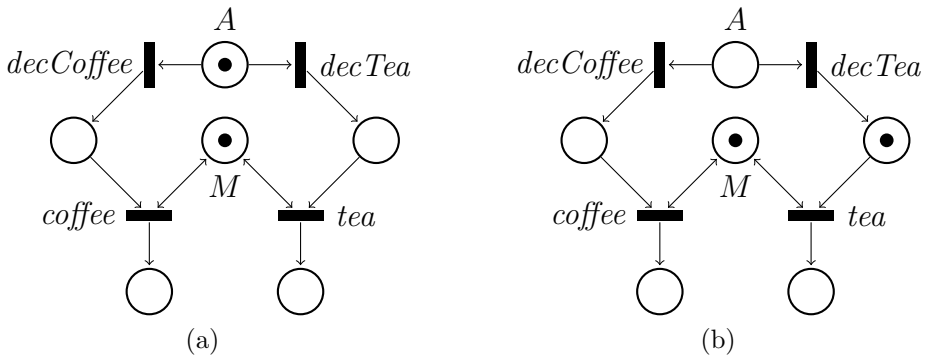


Figure 1: This simple Petri net models the interaction between a costumer and a machine which can produce a cup of coffee or a cup of tea. The Petri net is displayed with its initial marking (cf. left panel (a)) and after firing the transition *decTea* (cf. right panel (b)).

2.2 Petri Games

Petri games [2] are an extension of Petri nets. The goal is to model the independent development of local players which only communicate when taking part in a joint transition. Therefore, each token symbolizes a local player. During communication the participating players exchange their history of visited places and taken transitions which then can be used for future decisions by the other player(s). The local players are divided to either belong to the global system player or to the global environment player. This introduces the extensions from nets to games.

The set of places is divided into places belonging to the system ($\mathscr{P}_S$) and to the environment ($\mathscr{P}_E$), respectively. This implies that there may exist several local players of the system and of the environment, respectively, depending on the number of tokens in the Petri game. Players of the environment behave non-deterministically since no information about their decisions are available. There exists a set $\mathscr{B} \subseteq \mathscr{P}_S \cup \mathscr{P}_E$ of *bad places* which marks places the environment wants to reach and the system wants to avoid. Notice that these places are bad from the point of view of the system. They are annotated with $\bot$.

In order to win the Petri game it is required to find a winning strategy for the system players. A strategy restricts only the decisions taken at places belonging to the systems. Strategies are required to be deterministic and are assumed to be global meaning that they control every local system player. A strategy is called winning if it can ensure that no bad place is reached regardless of the non-deterministic decisions the environment takes. This

implies that Petri games are defined as safety games.

2.2.1 Definition of Petri Games

A *Petri game* $\mathscr{G}$ is a tuple $(\mathscr{P}_S, \mathscr{P}_E, \mathscr{T}, \mathscr{F}, In, \mathscr{B})$, where:

- $\mathscr{P}_S$ is a finite non-empty set of places belonging to the *system*.
- $\mathscr{P}_E$ is a finite non-empty set of places belonging to the *environment*.
- $\mathscr{B}$ is a finite non-empty set of *bad places*.

Notice that the multisets $\mathscr{T}$, $\mathscr{F}$, and the function In remain the same as in a Petri net with $\mathscr{P} = \mathscr{P}_S \cup \mathscr{P}_E$ which shows that the extension is based on dividing places into two different groups in order to introduce the global system and the global environment player. The distribution is required to be disjoint ($\mathscr{P}_S \cap \mathscr{P}_E = \emptyset$).

In the following, $\mathscr{G}^1$, $\mathscr{G}^2$, etc. are used to address individual Petri games. The superscripts are also used for the elements of the tuple corresponding to the Petri game, e.g., $\mathscr{G}^1 = (\mathscr{P}_S^1, \mathscr{P}_E^1, \mathscr{T}^1, \mathscr{F}^1, In^1, \mathscr{B}^1)$. Places belonging to the system are filled grey whereas places belonging to the environment in contrast remain white. System places can be viewed as the controllable part of the Petri game whereas environment places stay uncontrollable. The progress of a Petri game is also described by markings: $M_i : \mathscr{P}_S \cup \mathscr{P}_E \to \mathbb{N}_0$.

2.2.2 Unfolding and Strategy

As already mentioned, in order to win a Petri game it is required to find a winning strategy for the system. In this subsection, strategies for Petri games are defined and it is outlined when they are called winning. Strategies rely on unfoldings which represent the possibly different history known to a place because of the transitions which were used to reach it.

The *unfolding* $\beta_U = (\mathscr{G}^U, \lambda)$ of an underlying Petri game $\mathscr{G}$ consists of a Petri game $\mathscr{G}^U$ in which all joins of places with equal label in the underlying net have been removed and an homomorphism λ from $\mathscr{G}^U$ to $\mathscr{G}$. The removal of all joins results in a replication of places for each possible history they can be reached with. History refers to the places and transitions which the player has taken along with the places and transitions other places have taken up to the joint transition they participated in. On a joint transition, players exchange their complete history including their knowledge about the history of other players. This implies that information transfer works in a transitive manner.

The unfolding represents the specific information a player can have in a place by having the place copied for each level of informedness. λ maps the possibly replicated places of $\mathscr{G}^U$ to their original places in $\mathscr{G}$ in order to show the relationship between the two games. Notice that the unfolding enumerates all possible choices of the system and of the environment. Consequently it represents all possible ways the game can develop.

The Petri game $\mathscr{G}^U$ is required to have the same outgoing transitions for each copy of a place as the original game. Furthermore, the initial marking stays the same. Notice that a place which need not be replicated is taken over from $\mathscr{G}$ to $\mathscr{G}^U$. The unfolding unwraps recursion because the history incorporates the taken transitions during each unwrapping of the recursion. This implies that the unfolding stores how often the recursion is unwrapped which may result in infinite unfoldings.

A *strategy* is defined based on the unfolding because the unfolding provides all information per place which can be used to find a strategy for the system players. In a strategy, every place belonging to the system has to decide on engaging in exactly one of the possible transitions or to stop movement. This requirement ensures deterministic strategies. When making decisions for the system places the information about the history of the place can be used. The unfolding also models all transitions of the environment. However, as the environment is assumed to behave non-deterministically it is not possible to restrict environment transitions.

The function $pre(t)$ generates as output the finite multiset of places which precede the transition t. It is defined by $pre(t)(p) = \mathscr{F}(p,t)$, returning for each place p the number of pairs (p,t) in $\mathscr{F}$ (including zero if the pair is not contained). The multiset $pre(t)$ is viewed as a function which returns for each place how often it precedes t. A *sub-process* $\beta' = (\mathscr{G}', \lambda')$ of an unfolding $\beta = (\mathscr{G}, \lambda)$ is produced by removing transitions and the following places from $\mathscr{G}$ resulting in $\mathscr{G}'$ as well as closing unwrapped recursion. The homomorphism λ' relates the fewer places to the underlying Petri game of both unfoldings.

Formally, a *(global) strategy* σ for all local system players in a Petri game $\mathscr{G}$ is a finite sub-process $\sigma = (\mathscr{G}^\sigma, \lambda^\sigma)$ of the unfolding $\beta^U = (\mathscr{G}^U, \lambda)$ of the underlying game $\mathscr{G}$ for which the following two conditions must hold:

- if $p \in \mathscr{P}_S^\sigma$ then σ is deterministic at place p
- if $p \in \mathscr{P}_E^\sigma$ then $\forall\, t \in \mathscr{T}^U : ((p,t) \in \mathscr{F}^U \wedge \forall\, p' \in pre(t) : p' \in \mathscr{P}_E) \Rightarrow (p,t) \in \mathscr{F}^\sigma$

A strategy σ is called *deterministic* at a place p when for all reachable markings M in $\mathscr{N}^\sigma$ it holds: $p \in M \Rightarrow \exists^{\leq 1}\, t \in \mathscr{T}^\sigma : p \in\ pre(t) \subseteq M$. This

means that the strategy can activate at most one transitions per system player at every possible decision point which is represented by a reachable marking for the strategy. The second condition ensures that the strategy does not impose restrictions on transitions which require only local environment players to fire. $pre(t)$ is defined on the underlying Petri game $\mathscr{G}$.

As already mentioned, the unfolding is infinite when it unwraps recursion because of the additional history of each unfolding. Nevertheless, the strategy is required to be finite by definition. This implies that the strategy can contain loops in order to deal with loops in the underlying Petri game. Notice that the loop may be unwrapped finitely often resulting in different decisions by the system but there must exists a finite point from which on the system always repeats its decision.

2.2.3 Safety Assumption and Deadlock Avoiding Strategies

The winner of a Petri game $\mathscr{G}$ and its strategy $\sigma = (\mathscr{G}^\sigma, \lambda^\sigma)$ is determined by checking whether there exists a possible sequence of transitions visiting a bad place. In this case, the environment wins else the system wins. It is clear that the environment player does not benefit from stopping movement if it has not reached a bad place. Due to this safety condition the assumption is made that the environment player does not stall the game but always picks a transition. This licenses the enumeration of all of its choices in order to find the winner of a Petri game.

The system has to be forced to engage in transitions because otherwise it would win every game by not moving at all, in which the environment cannot reach a bad place locally. For the remainder of this thesis, strategies are stipulated to be *deadlock avoiding*. It is required that for all reachable markings M, it must hold that $\exists\ t \in \mathscr{T}^U : pre(t) \subseteq M \Rightarrow\ \exists t \in \mathscr{T}^\sigma : pre(t) \subseteq\ M$. This ensures that if there exists an enabled transition in the underlying unfolding of the strategy then there must also exist an enabled transition in the strategy. When no transition is enabled the game has *terminated* and it is possible to determine the winner. The system wins the game if a bad place is never reached independent of the decisions the environment makes.

2.2.4 Example Modeling of a Vending Machine

Consider the example in Fig. 2 which is an extension to the example for Petri nets of Fig. 1. The token which represents the person is now in an environment place A. The machine starts on the system place M. The environment can either decide that it wants a cup of coffee or a cup of tea and thus orders according to this decision. The system must take the order

and then it must decide to produce either a cup of coffee or a cup of tea because it is defined to be deadlock avoiding. The distribution into system and environment introduces a game setting to the underlying net.

On the synchronous transitions a system and an environment token are required to fire it and a system and an environment token are produced. This implies that both tokens flow through the net by only visiting system and environment places, respectively. The bad place $\perp$ models that it is a bad behavior of the system to produce the wrong product, i.e., a bad place is reached when a cup of coffee is ordered but a cup of tea is produced or vice versa. In Petri games, it is a useful property that the number of tokens stays constant. This enables the distribution of the global strategy to local controllers [2]. The system players taking part in the transitions to the bad places return to their origin in order to achieve this property.

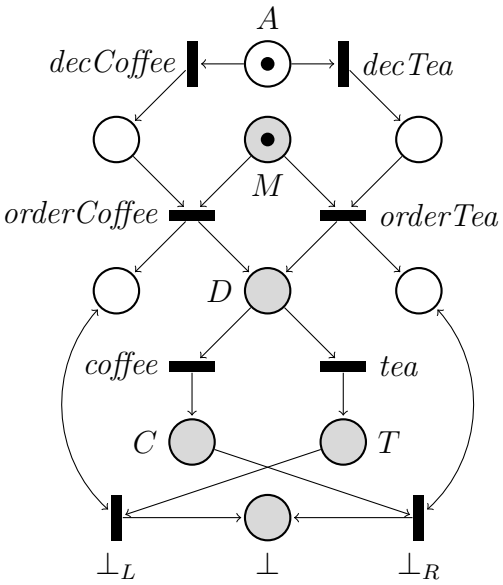


Figure 2: An extended example for the interaction between a person and a machine producing either a cup of coffee or a cup of tea is displayed. The person is modeled as the environment whereas the machine is modeled as the system and a bad place is introduced to prohibit unintended behavior. *coffee* and *tea* represent the production of coffee and tea, respectively.

The strategy to avoid the bad place is clear on an intuitive level. Since the machine is forced to wait for an order the only difficulty is to produce the right product. Because of the exchanged history which is already indicated by the labels of the joint transitions *orderCoffee* and *orderTea*, it is always clear which product the environment ordered and therefore it can be produced. In

order to become familiar with the terminology, the unfolding and the strategy are shown in Fig. 3 and Fig. 4, respectively. The figures illustrate the formal approach to solve Petri games.

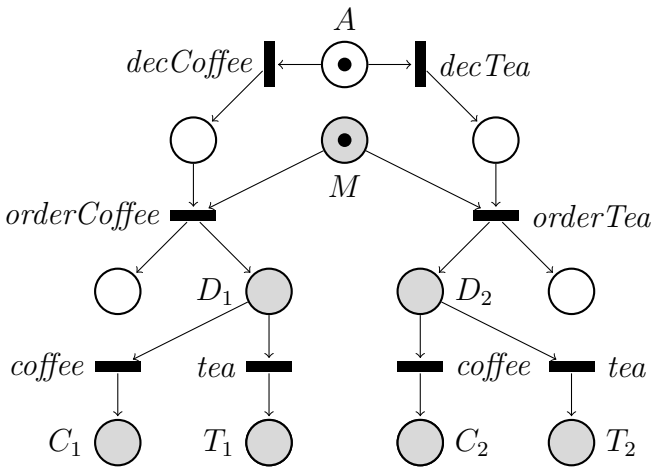


Figure 3: The unfolding for the example Petri game of Fig. 2 is displayed. Transitions to bad places are omitted to increase readability. For $i = 1, 2$, the labels D_i, C_i, and T_i indicate places which have been unfolded depending on different histories. *coffee* and *tea* represent the production of coffee and tea, respectively.

The unfolding has copied three places D, C, and T each one time. The resulting six places are renamed to D_1, D_2, C_1, C_2, T_1, and T_2. The places D_1 and D_2 both represent the place where the system has to decide which hot beverage to produce. D_1 represents the situation where the joint transition *orderCoffee* was fired and therefore the system knows that the environment expects a cup of coffee. In the place D_2, the system's history incorporates that a cup of tea is expected by the environment. After the production of either a cup of coffee or a cup of tea one of the places C_1, T_1, C_2, or T_2 is reached. They represent which product was expected by subscript 1 standing for coffee and subscript 2 for tea and which product was produced by C standing for coffee and T for tea.

The transitions to bad places are omitted to increase readability because each of the four lowermost places has one outgoing transition to a unique bad place. The bidirectional arrows from and to the environment place in the transitions to bad places incorporate a loop which would need to be unfolded infinitely often resulting infinitely many places.

C_1 and T_2 are the places which the strategy must reach in order to avoid a bad place because both places represent that the order hot beverage was

produced. The transitions to bad places are not enabled for these places. The places C_2 and T_1 symbolize that the wrong product was served. In place D_1, the strategy has to decide for the transition *coffee* to reach C_1 and in D_2 for the transition *tea* tor each T_2. The corresponding strategy is displayed in Fig. 4. Comparing the unfolding (cf. Fig. 3) and the strategy it becomes obvious that the essential difference lays in the pruning of transitions originating from system places.

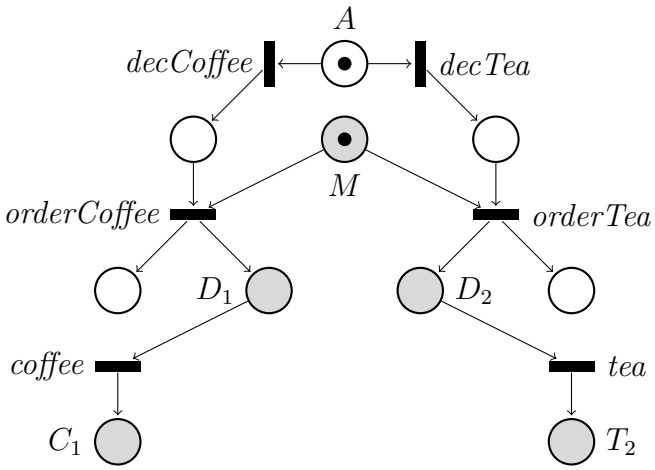


Figure 4: The strategy for the example Petri game of Fig. 2 is displayed. It shows that the system has to produce a cup of coffee at place D_1 and a cup of tea at place D_2. Notice how the system in place M has to keep both transitions because only one will be enabled depending on the decision of the environment. It can therefore aggregate the information about which hot beverage to produce and still be deterministic.

2.3 Communicating Sequential Processes (CSP)

Communicating Sequential Processes (CSP) [4] provides a language which is used to describe concurrent processes and their interactions. CSP is a textual representation of processes which can be transformed into a transition system describing the concurrent behavior. In the following, we focus on the textual representation and will develop an intuitive understanding of the operators. The introduced game-based semantics of Section 3 will give a different meaning than the original semantics resulting in the transition system.

2.3.1 Definition of a CSP Process

The following subset of the syntax of CSP will be used in this thesis (for more information on CSP operators see, e.g., [4] and [13]). P, P_1, and P_2 represent one process each, e is an event, X a set of events called the synchronization alphabet, and L is a label. The intuitive meaning of the operators is already given as comments and will be deepened in this subsection.

$$
\begin{array}{lll}
P ::= & \text{“}STOP\text{”} & \text{// no communication} \\
| & e \text{ “}\rightarrow\text{” } P & \text{// event} \\
| & P_1 \text{ “}\underset{X}{\parallel}\text{” } P_2 & \text{// synchronization} \\
| & P_1 \text{ “}\mid\text{” } P_2 & \text{// deterministic choice} \\
| & P_1 \text{ “}\sqcap\text{” } P_2 & \text{// non-deterministic choice} \\
| & L \text{ “}=\text{” } P & \text{// recursion}
\end{array}
$$

The simplest CSP process is $STOP$ which cannot perform any events. An event is emitted with the prefix operator ($\rightarrow$). It takes an event e and a process P and produces the process $e \rightarrow P$ which first emits the event e and then behaves like P. A simple example is a process $hello \rightarrow STOP$ which only emits the event *hello*.

The next operator is the parallel operator. It takes two processes as arguments and enforces synchronization on all events which are part of the synchronization alphabet X. In $(a \rightarrow (e \rightarrow STOP)) \underset{\{e\}}{\parallel} (e \rightarrow STOP)$, the synchronization operator determines the order of the occurring actions. e can only occur after the event a occurred because e is in the synchronization alphabet whereas a is not.

There are two forms of choices between processes. The first form is the deterministic choice ($\mid$) and the second form is the non-deterministic choice ($\sqcap$). Both choice options take two processes P and Q as input and produce the respective choice between the two processes ($P \mid Q$ and $P \sqcap Q$). The processes $(hello \rightarrow STOP) \mid (bye \rightarrow STOP)$ and $(hello \rightarrow STOP) \sqcap (bye \rightarrow STOP)$ constitute an example for the deterministic choice and the non-deterministic choice, respectively. In both cases, a decision is made to either emit the event *hello* or *bye*, in the former case the decision is controllable whereas it remains uncontrollable in the latter case.

The distinction between deterministic and non-deterministic choice becomes relevant for the parallel operator ($\underset{X}{\parallel}$). Deterministic choices follow the first event which occurs if both their decisions are part of a synchronization. On the other hand, non-deterministic choices do not depend on which event of a synchronization occurs first but are made independently.

This means that in $((a \to A) \mid (b \to B)) \underset{\{a,b\}}{\|} ((a \to A) \sqcap (b \to B))$ the deterministic choice on the left-hand side will follow the decision made by the non-deterministic choice on the right-hand side of the synchronization.

CSP allows recursion with the "="-operator by making it possible to assign a label L to a process P as in $L = P$. The label L then represents the process P and can be used in other processes to represent it. For example, the process $A = (a \to A)$ constitutes a process which emits infinitely many instances of a. The recursion operator enables us to state reused processes only once as in the example $(a \to A) \mid (b \to A)$ where the process can deterministically decide between a and b but afterwards it will behave in both cases like A. The only difference lays in the initial action whereas the following part A has to be written down only once to define it.

The binding order in CSP is simple. $\to$ binds the strongest, followed by the two choice operators ($\mid$ and $\sqcap$) binding equally strong, which in turn bind stronger than $\|$. The recursion operator $=$ binds weakest. For instance, this makes it possible to omit all brackets in the previous example about deterministic and non-deterministic choice retaining the intended meaning: $a \to A \mid b \to B \underset{\{a,b\}}{\|} a \to A \sqcap b \to B$. The three operators $\mid$, $\sqcap$, and $\|$ bracket left-associative meaning that $A \sqcap B \sqcap C \sqcap D$ represents $((A \sqcap B) \sqcap C) \sqcap D$, whereas $\to$ brackets right-associative implying that $a \to b \to c \to d$ stands for $a \to (b \to (c \to d))$.

2.3.2 Example Modeling of a Vending Machine

In this subsection, two CSP expressions are shown which behave equivalent to the example Petri net from Fig. 1 (a) and to the example Petri game from Fig. 2. The expression for the Petri net in Fig. 1 (a) looks as follows:

$$\begin{aligned} A &= (\mathit{decCoffee} \to \mathit{coffee} \to \mathit{STOP}) \sqcap (\mathit{decTea} \to \mathit{tea} \to \mathit{STOP}) \\ M &= (\mathit{coffee} \to M) \mid (\mathit{tea} \to M) \\ \mathit{Fig1} &= A \underset{\{\mathit{coffee},\mathit{tea}\}}{\|} M \end{aligned}$$

The process A represents the person who can decide between a cup of coffee and a cup of tea and the process M represents the machine which will either produce a cup of coffee or a cup of tea and then wait for the next order. The process $\mathit{Fig1}$ puts both A and M together and enforces synchronization on *coffee* and *tea*. Notice that the person will force the machine in this scenario to produce the right product because A uses the

non-deterministic choice whereas M uses the deterministic one and therefore waits for its only synchronization partner and follows the person's decision producing the desired hot beverage.

The expression for the Petri game from Fig. 2 can be defined as follows without the explicit division into places belonging to the system or to the environment:

$$\begin{aligned} A &= (\mathit{decCoffee} \to \mathit{orderCoffee} \to \bot_L \to \mathit{STOP}) \sqcap \\ &\quad (\mathit{decTea} \to \mathit{orderTea} \to \bot_R \to \mathit{STOP}) \\ M &= (\mathit{orderCoffee} \to M') \mid (\mathit{orderTea} \to M') \\ M' &= (\mathit{coffee} \to \bot_R \to \mathit{STOP}) \mid (\mathit{tea} \to \bot_L \to \mathit{STOP}) \\ \mathit{Fig2} &= A \underset{\{\mathit{orderCoffee}, \mathit{orderTea}, \bot_L, \bot_R\}}{\parallel} M \end{aligned}$$

The process A represents the person which now either decides and orders a cup of coffee or a cup of tea. Afterwards it tries to use a bad transition representing that the wrong product was served. The machine M first reacts to the order of either a cup of coffee or a cup of tea and then reaches in both cases the same state where it can either produce a cup of coffee or a cup of tea. The process $\mathit{Fig2}$ ensures synchronization between A and M on the order and on the transitions which show bad behavior because the bad behavior can only be determined when investigating the behavior of both processes. Notice that there is no representation in CSP for the bad places of a Petri game which leads to the labeling of transitions by $\bot$ to indicate bad behavior as a workaround (which was already used when the figure was introduced in Section 2.2.4). Furthermore, CSP does not recognize system or environment but the way deterministic and non-deterministic choice behave coincides with the intended meaning of the Petri game.

3 Game-Based Semantics

In this section, a new semantics is developed as the main part of this thesis. The proposed game-based semantics translates expressions following a slightly modified syntax of CSP into Petri games.

This section is structured as follows: First, non-communicating local system and environment players are introduced. Second, it is explained how the rules of the structural operational semantics (SOS) work and in which way they are applied. Third, the modifications to the original syntax of CSP are outlined. Fourth, the SOS-rules used to derive simple transitions are introduced. Fifth, system and environment are given the chance to communicate with each other via SOS-rules for synchronization. Sixth and Seventh, SOS-rules for choices of the system and of the environment are introduced. It follows an explanation of the SOS-rules for recursion.

3.1 Terminated Players

A local player that cannot perform actions anymore is represented by a corresponding token reaching a place without outgoing transitions. Such a place is called *terminal* and the player reaching it is called *terminated*. As described in the next subsection, the rules of the structural operational semantics allow the derivation of the transitions of a Petri game. Therefore, no SOS-rules for terminal places have to be provided in the presented semantics.

In CSP, the $STOP$-process is used to describe that neither external nor internal communication can occur anymore. As Petri games distinguish between system and environment players two types of $STOP$-places are required representing the non-communication of the corresponding player. This is achieved by adding the subscript S as in $STOP_S$ for the terminal place of the system player and by adding the subscript E as in $STOP_E$ for the terminal place of the environment player, respectively.

As outlined in Section 2.2, Petri games can mark bad places. There is no equivalent in CSP. Reaching a bad place represents that the environment won. This is the reason why this place has no outgoing transitions. Therefore, each bad place is terminal but not vice versa. The bad place is by default an environment place in this semantics and the additional syntax construction *FAIL* suffices to specify the case in a uniquely recognizable manner.

In Fig. 5, all new syntactic concepts defined for the corresponding Petri games are shown. Each Petri game consists of one place which is labeled by the syntactic construct and one token residing in the place. Notice how the label *FAIL* identifies a bad place and replaces $\perp$ which is used in the original

paper on Petri games to symbolize bad places.

Figure 5: The corresponding Petri games for $STOP_S$, $STOP_E$, and $FAIL$ are displayed.

3.2 Structural Operational Semantics

In the following, the rules comprising the new semantics are introduced. The goal is to start with an expression P from a slightly modified syntax of CSP (which will be shown in the next subsection) and to finish with a Petri game $\mathscr{G} = (\mathscr{P}_S, \mathscr{P}_E, \mathscr{T}, \mathscr{F}, In, \mathscr{B})$. Rules have the form $\frac{premise}{conclusion}$ (*side-condition*) which can be read as the following statement: *premise and side-condition imply conclusion.* A semantics based on rules of this type is called structural operational semantics (SOS) [8] and the corresponding rules are called SOS-rules. The presented semantics is structural in the sense that the applicable rule is based on the syntactic structure of the expression. It is operational in the sense that the conclusion results in an extension of the Petri game by a transition.

The *conclusion* consists of one transition of a Petri game including the places directly preceding it and the places directly following the transition. The transition and all places have to be labeled and there must exist at least one place before and after the transition. The *premise* consists of a number of transitions (including zero transitions) for which the same conditions regarding places and labels as for the conclusion have to hold. The transitions of the premise are strictly smaller than the conclusion, i.e., less places take part in the transition or the labels of the places contain less operators. The *side-condition* will be used for the synchronization operator and the recursion operator. For the former case, it require a transition either to be or not to be in a set of transitions. In the latter case, the side-condition requires that the derived recursion is defined.

Initially, there exists one place labeled by the given expression P. The expression is in $\mathscr{P}_S$ if its first operator is $\rightarrow_S$, $|$, or $STOP_S$, it is in $\mathscr{P}_E$ and $\mathscr{B}$ if its first operator is $FAIL$, or only in $\mathscr{P}_E$ for all other operators. The transition operator of the system ($\rightarrow_S$) and the choice operator of the system ($|$) will be introduced in Section 3.4 and Section 3.6, respectively. Initially, there are no transitions ($\mathscr{T} = \mathscr{F} = \emptyset$) and the initial marking In

is fixed to assign only one token to the place labeled by P. Notice that In will not change when deriving transitions because the created Petri game is given with its initial marking.

Next, systematically all possible transitions are derived via the SOS-rules which will be introduced in the remainder of this section. As only the reachable part of a Petri game is of interest, it is sufficient to only start the derivation of a transition when all places preceding the transition exist in the union of the places of the system and of the environment ($\mathscr{P}_S \cup \mathscr{P}_E$). The SOS-rules are applied recursively to derive a transition until all premises and side-conditions are fulfilled. An empty premise or an empty side-condition is always fulfilled. By this process, a derivation tree is constructed where the first SOS-rule applied is the lowermost part of the tree and its premise is the conclusion of the second applied SOS-rule. This procedure is continued till all premises are empty and all side-conditions are fulfilled. The tree growth upwards and can branch if a rule has more than one premise. Two examples can be found in Fig. 18 and Fig. 19 in Section 4. The first example is simpler and can be understood without in-depth knowledge of the rules introduced in the remainder of this section.

For every successful derivation, the transition and the places of the lowermost conclusion are added to the respective sets of $\mathscr{G}$. For each arrow of the transition, the corresponding pair is added to the flow relation $\mathscr{F}$ of $\mathscr{G}$. According to the first operator (after $\|_X$ and ${}_X\|$), the places can be distributed over $\mathscr{P}_S$, $\mathscr{P}_E$, and $\mathscr{B}$ as for the very first place of the Petri game. The split synchronization operators $\|_X$ and ${}_X\|$ will be introduced in Section 3.5.1. Notice the annotations $\|_X$ and ${}_X\|$ are not part of the syntax but only arise from the rules for the opening of synchronization. Therefore, the operators where not mentioned in the initial distribution of P above.

The SOS-rules induce the labels of the transition and the places. The label of the transition has as subscript a unique number because all places taking part in this transition are part of the derivation. This ensures unique pairs for the transition in the flow relation $\mathscr{F}$ and enables several transitions which are labeled equally up to the number. The number is left out in drawn Petri games for convenience resulting in Petri games with several transitions with the same label without these transitions collapsing into one huge transition subsuming all small transitions with the same label. Notice that the same does not hold for places because a place is uniquely identified by its label and it is possible to reach it via several transitions.

If a transition with the same label (without the unique number) and the same preceding and following places can be derived with more than one derivation tree then it is added for each derivation tree to the Petri game. Two derivation trees are different if at some point different rules are used.

This ensures that the derivation according to the semantics does not simplify the given Petri game and it is possible to give for example a decision where it does not matter which alternative the system chooses.

When all transitions for a place have been derived one continues with the newly added places until all places have been processed. If a place is reached more than once via several transitions then it is only processed the first time. This ensures that the derivation terminates. The derivation of transitions happens in a breadth-first search like manner.

3.3 Syntax

The syntax of an expression P on which the SOS-rules, which will be introduced in the remainder of this section, are applicable looks as follows where t is a transition, L is a label, and X is a set of transitions representing the synchronization alphabet. P, P_1, and P_2 are expressions according to the syntax. Transitions and labels are finite words of letters and numbers starting with a letter. The usage of subscripts is possible. Transitions often start with a lower case letter whereas labels mostly start with an upper case letter. For transitions and labels the words *STOP* and *FAIL* are forbidden to prohibit confusion with the syntax-constructs.

$P ::=$		“$STOP_S$”	// termination environment
	$\mid$	“$STOP_E$”	// termination system
	$\mid$	“$FAIL$”	// bad place
	$\mid$	t “$\rightarrow_S$” P	// transition system
	$\mid$	t “$\rightarrow_E$” P	// transition environment
	$\mid$	P_1 “$\mid$” P_2	// system choice
	$\mid$	P_1 “$\sqcap$” P_2	// environment choice
	$\mid$	P_1 “$\underset{X}{\parallel}$” P_2	// synchronization
	$\mid$	L “$=$” P	// recursion

There are three slight modification in contrast to the original syntax of CSP from Section 2.3.1. As explained in Section 3.1, the syntactical construct *STOP* is replaced by $STOP_S$ and $STOP_E$ to differentiate between a terminated system player and a terminated environment player and the construct *FAIL* is added to symbolize a bad place. In the next subsection, it will be explained why thirdly the “$\rightarrow$”-operator has to be replaced by $\rightarrow_S$ and $\rightarrow_E$. Notice that because of the replacement of *STOP* and $\rightarrow$ these two operators without the subscripts S and E are not allowed according to the syntax.

3.4 Transitions

The transitions of players in a Petri game are described with the prefix operator of CSP ($\rightarrow$). It is important to indicate which player has the chance to perform the transition since the strategy for the player may refuse to fire the transition. For the transition t and the expression P, the prefix operator for the system, $t \rightarrow_S P$, and the prefix operator for the environment, $t \rightarrow_E P$, are introduced.

A transition requires no further condition but having a token in the place the transition starts in. As the semantics only results in the structure of a Petri game with an initial marking it is obvious that the SOS-rules in the semantics for $\rightarrow_S$ and $\rightarrow_E$ have an empty premise, respectively.

The two corresponding SOS-rules for transitions of the system and of the environment are outlined in Fig. 6. The rule ST (system transition) states that if there exists a system place labeled by $x \rightarrow_S P$ then the transition x can be defined by adding x_c to $\mathscr{T}$ and the two pairs $(x \rightarrow_S P, x_c)$ and (x_c, P) to $\mathscr{F}$; c is the counter mentioned before which makes all transitions unique. P is added according to its first operator either to $\mathscr{P}_S$, $\mathscr{P}_E$, or $\mathscr{P}_E$ and $\mathscr{B}$. The rule ET (environment transition) works in an analog manner for the environment.

So far, system places are circles filled grey and environment places are circles which remain white. An additional, new type for places is introduced. The place is depicted by a dashed, white circle as can be seen for both places labeled by P in Fig. 6. When the new place follows a transition in a conclusion then it denotes that the first operator of the place after $\|_X$ and ${}_X\|$ decides which set of places it is added to and whether it is marked as a bad place. When it is used for other places in an SOS-rule (i.e. it precedes the transition in the conclusion or it stands in the premise) then it defines that the type of the place does not matter for this rule. The only imposed requirement in this case is that the corresponding places in the premise and in the conclusion have to either belong both to the set of system places or both to the set of environment places.

3.5 Synchronization

Players can interact with each other by means of the operator for synchronization ($P \underset{X}{\|} Q$). It takes as input two expressions P and Q and a possibly empty set of transitions X on which it enforces synchronization. X is called the *synchronization alphabet*. It is defined that a transition $a \in X$ can only occur if P and Q can perform it at the same time. X must not include τ as this symbol represents internal transitions.

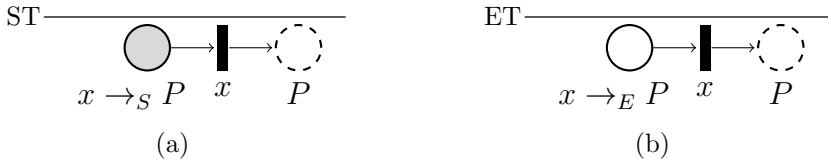


Figure 6: The rules ST (system transition; cf. panel (a)) and ET (environment transition; cf. panel (b)) for places of the form $x \to_S P$ and of the form $x \to_E P$ are displayed.

For example, $a \to_S b \to_S STOP_S \underset{\{a\}}{\|} c \to_E a \to_E STOP_E$ only licenses one order of transitions, namely first c (because the system player has to wait for synchronization on a), next a (because both players can only perform the transition they synchronize on), and at last b (because the environment player already terminated). As outlined in Section 2.2.3, the system is deadlock avoiding and the environment tries to reach a bad place. In this example, no player is allowed to or will stop, the former holds for the system, the later for the environment.

The SOS-rules for synchronization consist of four parts: (1) one rule for splitting in the beginning, (2) one rule for closing the synchronization in the end, (3) one rule for the actual synchronization (in two versions), and (4) a set of rules for performing local transitions which are not part of the synchronization alphabet.

3.5.1 Opening and Closing of a Synchronization

The opening of the synchronization operator takes place by splitting the operator. It enables us to define Petri games which incorporate more than one token during the play of the game. Splitting a parallel operator aims at obtaining two independent players. Nevertheless, each player provides information about with which player it has to synchronize on which synchronization alphabet. This is done by splitting the one player $P \underset{X}{\|} Q$ into the following two players $P\|_X$ and ${}_X\|Q$ [7]. Notice the different positions of X in $\underset{X}{\|}$, $\|_X$, and ${}_X\|$ representing the single parallel operator before the opening and the two produced operators after the opening. Brackets of several parallel operators ensure that at any time only one pair can synchronize. The synchronization operator takes exactly two expressions as arguments. This implies that it builds a binary tree which represents the synchronization structure.

For bracketing, ${}_X\|$ and $\|_X$ bind equally strong but weaker than the choice operators and stronger than the synchronization operator $\underset{X}{\|}$.

The SOS-rules of the semantics produce some transitions which are needed to represent internal steps. The rule in Fig. 7 is the first one to do so and shows that the transitions are labeled by τ in these cases. Internal transitions are not visible to other local players. This implies that it is not possible to perform synchronization on them.

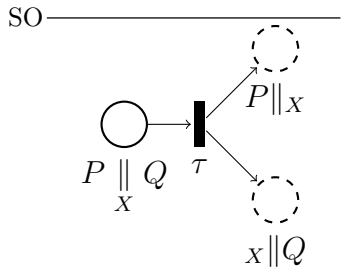


Figure 7: The rule SO (synchronization open) for the opening of the parallel composition in places of the form $P \underset{X}{\|} Q$ is displayed.

The place before the transition is assigned to the environment because the place and the transition are produced in order to represent internals of the semantics. Environment players do not get more power to win the Petri game by an additional place with only one outgoing transition where they can stop moving because they want to reach a bad place. The decision where to add $P\|_X$ is based on the first operator of P. An analog statement holds for ${}_X\|Q$.

Rule SC (synchronization close) deals with the situation when both players for one split up "$\|$"-operator have terminated, i.e., they have the form $STOP_S\|_X$ or $STOP_E\|_X$ and ${}_X\|STOP_S$ or ${}_X\|STOP_E$. In this case, the two players can be merged into one player which cannot perform any action ($STOP_E$) but by removing the split parallel operator it is shown that both players terminated. The rule SC is outlined in Fig. 8. The closing of local players works for them belonging to the same global player as well as to different global players. Only the produced player is an environment player as the player was which opened the synchronization.

3.5.2 Deriving a Synchronous Transition

The following SOS-rules deal with the actual synchronization of two players. In order to model the general case of synchronization, lets us start with a specifically easy case characterized as the synchronization of exactly two

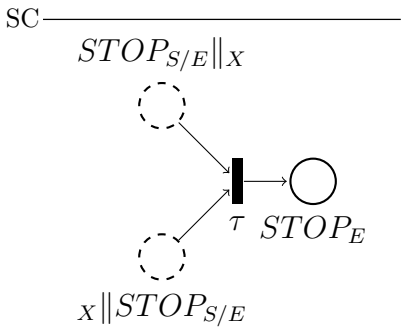


Figure 8: The rule SC (synchronization close) for closing the synchronization of pairs of places of the form $STOP_{S/E}\|_X$ and $_X\|STOP_{S/E}$ is displayed. It can be chosen independently between $STOP_S$ or $STOP_E$ for any occurrence of $STOP_{S/E}$.

places. The two places are labeled by $P\|_X$ and $_X\|Q$ and can synchronize on a transition t if t is in the synchronization alphabet ($t \in X$) and P and Q can perform t on their own, respectively. Note that $P\|_X$ and $_X\|Q$ must have the same synchronization alphabet. This ensures that only related pairs of places can synchronize with each other. Furthermore, the split parallel operators $\|_X$ and $_X\|$ stay attached to the produced places to ensure future synchronization. The rule SY-PRE is delineated in Fig. 9.

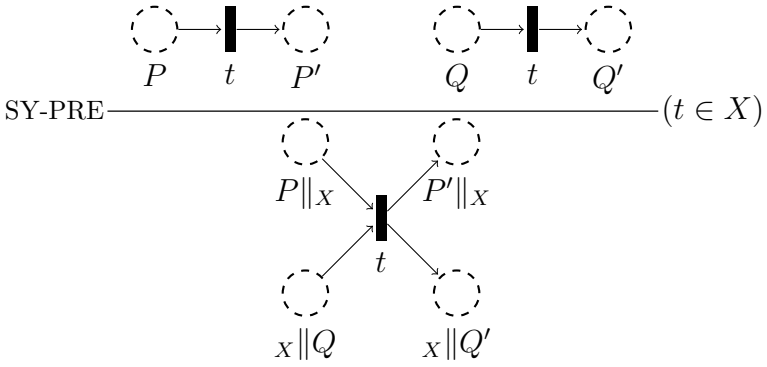


Figure 9: The rule SY-PRE (preliminary version of the rule for synchronization) for the synchronization between two environment players of the form $P\|_X$ and $_X\|Q$ on a transition $t \in X$ is displayed.

In order to model the general case of the synchronization on a transition t the rule SY-PRE has to be extended. The outer synchronization operator requires the left-hand side and the right-hand side to perform the same action. In the general case, there may be several places $P_i\|_X$ on the left-hand side

side because after the initial outer opening of a synchronization further inner openings may have taken place. Thus several $P_i\|_X$ synchronize on the same t which leads to all places, participating in the inner synchronization, taking also part in the outer synchronization. This holds in an analog manner for the places ${}_X\|Q_i$. This implies that the general rule for synchronization deals with the synchronization of more than two places.

Accordingly, the rule SY-PRE (cf. Fig. 9) is extended to the rule SYNC which is introduced in Fig. 10. Instead of a single place P and a single place Q participating in the synchronization exactly two sets of places $\{P_1, ..., P_m\}$ and $\{Q_1, ..., Q_n\}$ take part. All members of these sets have to be part of a possible derivation of the transition without the restriction of the outer synchronization. The sets represent a minimal but sufficiently large group of places which can perform the transition t when not restricted by $\|_X$ or ${}_X\|$, respectively. If those two sets exist then both sets can perform the transition together under the restriction of the outer synchronization. Notice that if one of the conditions $m = 1$ or $n = 1$ holds then no inner synchronization happens for the corresponding set but only one place is needed for the transition. For both subsets there must exist a derivation of the symbolized transition t.

When two local players synchronize with each other on a transition then it is not a problem at the time processing the first player that the second player might not be ready for the synchronization because the second player has to perform another transition before the synchronous transition. The following example illustrates this situation. Let us assume there exist the two places $(a \rightarrow_S STOP_S)\|_{\{a\}}$ and ${}_{\{a\}}\|(b \rightarrow_S a \rightarrow_S STOP_S)$. When these places are processed in sequence it is not possible to derive a transition for the first place because it lacks a partner for the synchronization on a. In the described case, the synchronous transition will be derived when processing the place which will be produced when the second player performs the transition b locally. The first place is processed without deriving a transition and then the second place is processed. Here, the transition b is derived leading to the place ${}_{\{a\}}\|(a \rightarrow_S STOP_S)$. When this place is processed the synchronous transition a is derived which synchronizes with the already processed place $(a \rightarrow_S STOP_S)\|_{\{a\}}$ resulting in the two places $STOP_S\|_{\{a\}}$ and ${}_{\{a\}}\|STOP_S$.

This example shows that nothing can happen before both players are ready for the transition but at the time the second player becomes ready the first player may be processed entirely and then it suffices to only derive transitions for the second player and not to reactivate an already processed place for derivation.

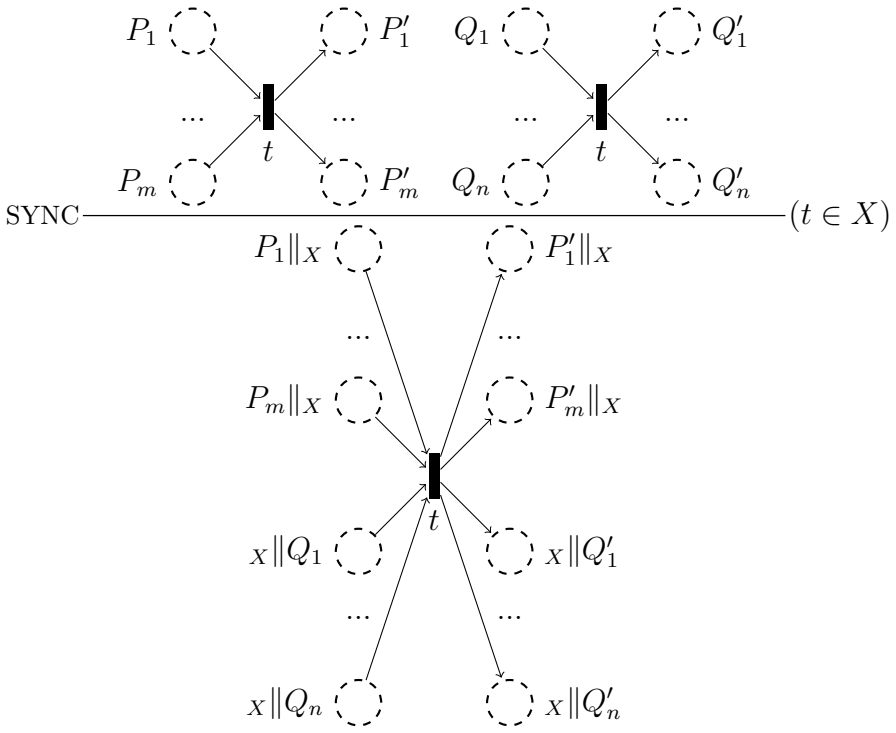


Figure 10: The rule SYNC for the synchronization between players of the form $P_i\|_X$ and $_X\|Q_i$ on a transition $t \in X$ is displayed. The set $\{P_1, ..., P_m\}$ is a sufficiently large but minimal set of places to derive the transition t without the restriction of $\|_X$ as is the set $\{Q_1, ..., Q_n\}$ without the restriction of $_X\|$.

3.5.3 Local Transitions in a Synchronization

Now, SOS-rules for transitions on which no synchronization happens are specified. These transitions are called *local*. The first four rules are needed for the cases when opening and closing of another parallel operator takes place. These cases are called the inner opening of another synchronization and the inner closing of another synchronization. The transitions are annotated by τ which is by definition not part of the synchronization alphabet. This makes the rules rather simple. Places of the form $(P \underset{X}{\|} Q)\|_X$ and $_X\|(P \underset{X}{\|} Q)$ can always perform their transition. The two produced places both keep the split parallel operator of the outer synchronization and get the split operator for the inner synchronization. The corresponding pair of rules SOL (synchronization open left) and SOR (synchronization open right) are outlined in Fig. 11. It is important to note that the places $(P \underset{Y}{\|} Q)\|_X$ and $_X\|(P \underset{Y}{\|} Q)$ in this case will always be environment places by the definition of the semantics. In

contrast, their successor places can be system or environment places.

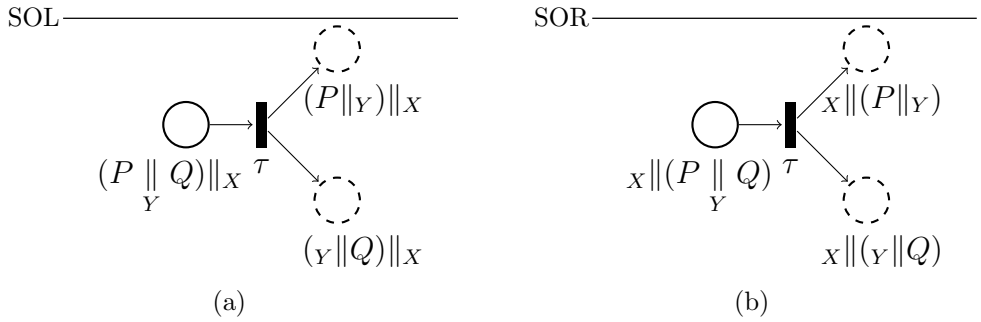


Figure 11: The rule SOL (synchronization open left; cf. panel (a)) for places of the form $(P \underset{Y}{\|} Q)\|_X$ is displayed. $P \underset{Y}{\|} Q$ engages on the left-hand side of a synchronization over X and can open a synchronization to the places $P\|_Y$ and $_Y\|Q$. The rule defines the τ-transition with the resulting places $_X\|(P\|_Y)$ and $_X\|(_Y\|Q)$. The rule SOR (synchronization open right; cf. panel(b)) works in an analog fashion for places which engage on the right-hand side of a synchronization.

The next two SOS-rules deal with the closing of an inner synchronization. They correspond directly to the opening rules. Again it holds that the transition will be labeled by τ meaning that it will not be part of the synchronization alphabet. The rules SCL (synchronization close left) and SCR (synchronization close right) are given in Fig. 12. $STOP_E\|_X$ and $_X\|STOP_E$ will always be environment players by definition. The rules state that if two places are terminated and therefore can be closed to one place than this can also happen if the two places took part in an additional outer synchronization. Notice that the resulting place must keep the synchronization operator in order to enable closing of the outer synchronization.

The last two SOS-rules for synchronization represent the case when a place performs a transition which is not part of the synchronizations alphabet. The rules SLS (synchronization left step) and SRS (synchronization right step) show that if P can perform a transition t to the place P' and $t \notin X$ then $P\|_X$ can do the same to $P'\|_X$ and $_X\|P$ can do to $_X\|P'$. The place following the transition keeps $\|_X$ or $_X\|$ because the opened synchronization operator applies for the life time of a player. The rules are displayed in Fig. 13. It is important to notice that the types of the places P and $P\|_X$ as well as of the places P' and $P'\|_X$ have to be either system or environment, respectively.

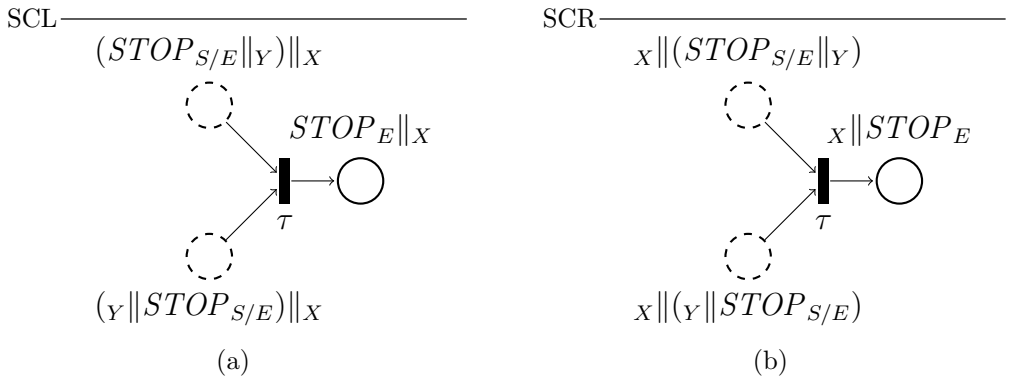


Figure 12: The rule SCL (synchronization close left) is displayed. It shows that two places of the form $(STOP_{S/E}\|_Y)\|_X$ and $(_Y\|STOP_{S/E})\|_X$, which are part of the left-hand side of a synchronization, have a τ-transition to a place $STOP_E\|_X$ representing the closing of an inner synchronization. For each $STOP_{S/E}$, it can be chosen independently between $STOP_S$ or $STOP_E$. The rule SCR (synchronization close right) works in an analog manner for places which are part of the right-hand side of a synchronization.

3.6 System Choice

So far, the outcome of a constructed Petri game is determined by whether a system player decides to stop at a place (only possible if another system player has a fireable transition) and whether the synchronization deadlocks. In addition, the system can also decide between P and Q via the "|"-operator representing a deterministic choice in CSP. Two expressions P and Q are taken as input and the transitions for both choices are added to the place labeled by $P \mid Q$. The corresponding rules are delineated in Fig. 14.

In order to make a choice, P and Q are required to have at least one transition. This results in two pairs of SOS-rules, depending on whether the outgoing transition has one or two successor places. Notice that in the semantics a transition with only one preceding place will have at most two direct successor places. Two successors can only occur if the first operator after the choice is a parallel operator. This constitutes a special case which will be dealt with soon. The premise ensures that when the left-hand side can do a transition on its own for example via the rule ST for transitions of the system (cf. Fig. 6 (a)) then the place for the choice can perform this transition to the same destination and behaves like the destination afterwards. The destination is added based on its first operator to the respective set(s).

It is important to note that P' and Q' belong to the same player in the premise and in the conclusion, respectively. Furthermore, it is required

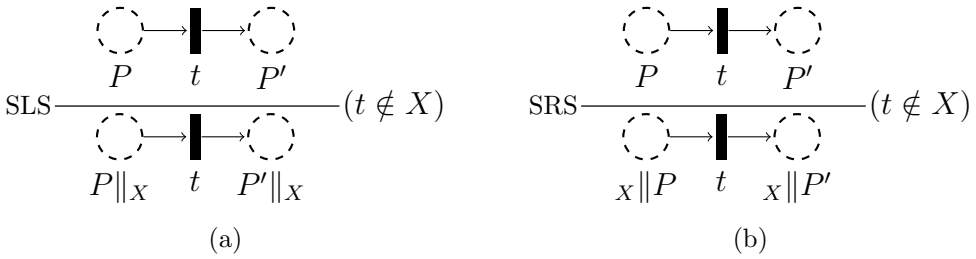


Figure 13: The rules SLS (synchronization left step) and SRS (synchronization right step) for places of the form $P\|_X$ and of the form ${}_X\|P$ are displayed.

that P and Q are system players. This prohibits any derived transitions where it is unclear whether a place belongs to the system or to the environment. This is the case for example for $a \rightarrow_E STOP_E \mid b \rightarrow_E STOP_E$. One might except that two transitions are derivable for this example but, in fact, no transition is derivable because neither the rule SL nor the rule SR is applicable.

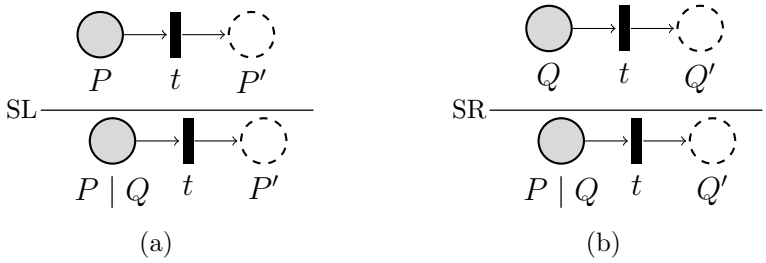


Figure 14: The rules SL (system left) and SR (system right) for places of the form $P \mid Q$ are displayed where P is chosen in the left rule and Q is chosen in the right rule.

The second pair of SOS-rules deals with the special case when P or Q in a place labeled by $P \mid Q$ have a transition with more than one successor place. This can only happen if the first operator of P or Q is the synchronization operator $\underset{X}{\|}$, i.e., P has the form $P_1 \underset{X}{\|} P_2$ or Q has the form $Q_1 \underset{X}{\|} Q_2$. In this case, the decision of the system for either P or Q is represented by a transition labeled by τ for the opening of the synchronization. The rules rules are displayed in Fig. 15.

The additional rules constitute a special case because a place belonging to the environment in the premise is transformed into a place belonging to the system in the conclusion. The place before the split of a parallel operator is

defined to be an environment place (cf. rule SO in Fig. 7) but this cannot hold here since the system made the decision which resulted in a split of players. Therefore, $P \mid Q$ is a system place despite P or Q being an environment place because the first operator is the parallel operator, respectively. The rules SLO (system left open) and SRO (system right open) state that the decision of the system can be a split of players.

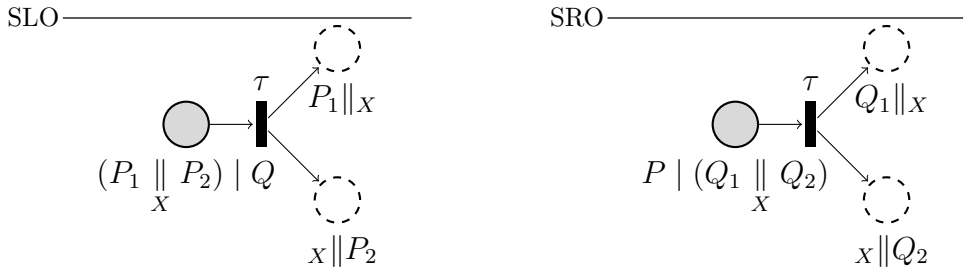


Figure 15: The rules SLO (system left open) and SRO (system right open) are displayed. SLO enables the left-hand side of a system place representing the decision between $P_1 \underset{X}{\|} P_2$ and Q to perform its decision via a transition opening the synchronization of $P_1 \underset{X}{\|} P_2$. Notice that $P_1 \underset{X}{\|} P_2$ on its own is an environment place but becomes a system place in $(P_1 \underset{X}{\|} P_2) \mid Q$. SRO works in an analog manner.

3.7 Environment Choice

The environment is supposed to behave non-deterministically. So far, the environment could only stop its progress which does not enable it to actively try to reach a bad place. In order to enable the environment to make choices the non-deterministic choice operator $P \sqcap Q$ is introduced. It takes as input two arguments P and Q and symbolizes the situation that the environment can choose between these two arguments.

The key difference between the deterministic and the non-deterministic choice operators is that the environment must finalize its decision by a transition labeled by τ before performing further transitions which might synchronize. Note that τ is not allowed to be part of the synchronization alphabet (cf. Section 3.5). The reason for defining obligatory τ-steps here is that the system cannot enforce the environment to follow its direction but the environment has its free choice with which the system has to deal when

synchronization takes place. The corresponding rules EL (environment left) and ER (environment right) are depicted in Fig. 16.

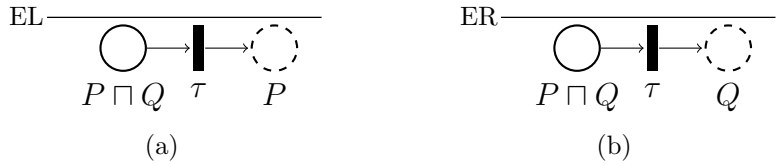


Figure 16: The rules EL (environment left) and ER (environment right) for places of the form $P \sqcap Q$ are displayed where P is chosen in panel (a) and Q is chosen in panel (b).

The environment must progress in the game as long as it has not reached a bad place in order to have a chance to reach one. Assuming the choice operator of the environment would work in the same way as it does for the system the environment would always decide for the fireable transition if only one of the two transitions is fireable. The system would control the environment in some cases where it is not intended.

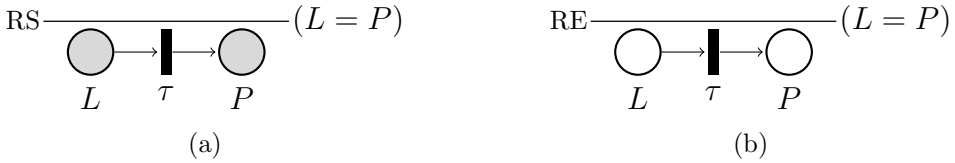
3.8 Recursion

There are two reasons to extend the semantics by the operator for recursion which is represented by $L = P$. It assigns a Petri game P to the label L. The label L can then be used instead of P in other Petri games. Firstly, it makes the semantics more practical because formulae get large for large Petri games. Recursion allows splitting the whole formula into smaller subformulae. Moreover, recursion allows defining finite Petri games with infinite paths of transitions by introducing loops.

The derivations in the semantics should terminate and every place should belong either to the system or to the environment. This requires two restrictions for the recursion operator. The first one is that the recursion has to be *action-guarded*. This is the case when for P a transition is derivable which is not τ, i.e., the first operator of P after a possibly empty number of system choices is either $\rightarrow_S$ or $\rightarrow_E$. The operator for system choices $|$ does not produce transitions but behaves like the decision which is taken and $\rightarrow_S$ or $\rightarrow_E$ are the only possibility to produce transitions which are not τ. $\rightarrow_E$ is only possible if no system choice occurs (cf. Section 3.6). The second restriction is that (nested-)self recursion including the "$\|$"-operator is not allowed. This restriction will be illustrated by an example in Section 4.3.

The SOS-rules are called RS (recursion system) and RE (recursion environment). They are delineated in Fig. 17. The rules describe that the place L

can perform an internal transition to the place P when $L = P$ is defined. Note that in RS and RE both places L and P have to be system places or environment places, respectively. This ensures that it is possible to find the type of the place L. Moreover, it is not possible to define labels for essential places like $STOP_S$, $STOP_E$, or $FAIL$ because the recursion is required to be action-guarded. The rules result in one place marked with the label L and one place labeled by P, respectively.



(a) (b)

Figure 17: The rules RS (recursion system) and RE (recursion environment) are delineated. They state that for a place L a τ-transition to a place P can be derived when $L = P$ is defined. L and P are either both system places (cf. panel (a)) or both environment places (cf. panel (b)).

The condition of action-guardedness prohibits as a side effect Petri games which have an infinite loop of τ-transitions. An example is the Petri game P with $P = P \sqcap P$. It persists of two environment places and has three transitions which are all labeled by τ. The first transition leads from the place P to the place $P \sqcap P$ and is derived with the rule for recursion of the environment. The next two transitions go from the place $P \sqcap P$ to the place P and are caused by the rules for the left and right decision of the environment, respectively. Only the condition that $P = P \sqcap P$ has to be action-guarded prohibits the existence of this futile Petri game.

4 Examples

In this section, examples illustrating the semantics are explained. The three main topics derivation trees, synchronization and recursion, and large Petri games will be discussed. The first subsection shows and explains a simple derivation tree. The next subsection deals with a synchronization of three places on one transition leading to a more advanced derivation tree. It is shown how different synchronization alphabets and changed bracketing lead to different derived transitions. In the third subsection, it is illustrated why recursion is restricted not to include the operator for synchronization recursively. The fourth subsection is about how the semantics handles synchronization in a Petri game which includes loops of different length. The fifth subsection shows how the semantics can be used to describe a meaningful Petri game. The Petri game describes a distributed alarm system which has to react correctly to a burglar by displaying the burglar's break-in position at all distributed components of the alarm system. The last subsection shows the description of another meaningful Petri game. Two local system players have to mimic the behavior of one environment player, respectively.

4.1 First Derivation Tree

In this example, the transition of the place $P\|_\emptyset$ is derived where P is defined as $P = a \rightarrow_S (P \underset{\emptyset}{\|} STOP_S)$. The transition will be used later in Fig. 20. No synchronization is possible for $P\|_\emptyset$ because of the empty synchronization alphabet. This implies that the only applicable rule is SLS (cf. Fig. 13) meaning that the left-hand side of a synchronization performs a transition which is not part of the synchronization alphabet. The premise of the rule requires that a transition for $P\|_\emptyset$ without the restriction of $\|_\emptyset$ is derivable. Since P is a label the only possible transition is caused by the opening of the recursion. The first operator of the right-hand side of the recursion is $\rightarrow_S$ implying that the place belongs to the system. Therefore, the only rule to derive the opening of the recursion is RS (cf. Fig. 17 (a)). The side-condition of the rule is true because it is defined that $P = a \rightarrow_S (P \underset{\emptyset}{\|} STOP_S)$. The rule RS has an empty premise meaning that the derivation tree is finished and it defines that the transition leads to the place $a \rightarrow_S (P \underset{\emptyset}{\|} STOP_S)$ and the rule SLS then defines that the derived transition reaches the place $(a \rightarrow_S (P \underset{\emptyset}{\|} STOP_S))\|_\emptyset$. The resulting derivation tree is depicted in Fig. 18.

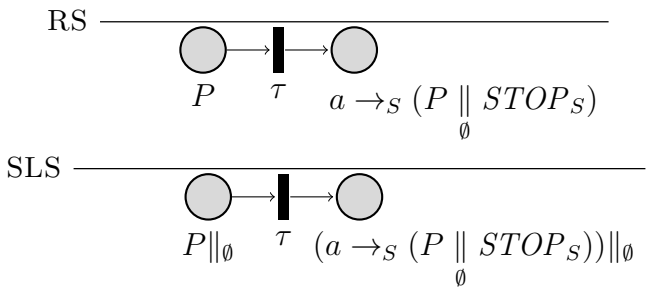


Figure 18: The derivation tree for the τ-transition of the place $P\|_\emptyset$ from the example from Fig. 20 is delineated. The rule RS requires as side-condition that $P = a \rightarrow_S (P \underset{\emptyset}{\|} STOP_S)$ is defined.

4.2 Three Places Synchronizing in Different Ways

The second example shows the derivation of a synchronization requiring three local players. The expression is $a \rightarrow_S A \underset{\{a\}}{\|} (a \rightarrow_E B \underset{\{a\}}{\|} a \rightarrow_S C)$ where A, B, and C are labels for arbitrary Petri games. After two internal transitions in order to open the initial synchronizations, three places with the following labels can still derive transitions: $(a \rightarrow_S A)\|_{\{a\}}$, ${}_{\{a\}}\|((a \rightarrow_E B)\|_{\{a\}})$, and ${}_{\{a\}}\|({}_{\{a\}}\|(a \rightarrow_S C))$. At first, $(a \rightarrow_S A)\|_{\{a\}}$ is processed and it is tried to derive a transition. The only candidate is the transition a and the only applicable SOS-rule is the rule SYNC (Fig. 10). This leads to two sets of places for which a transition needs to be derived according to the premise of SYNC. The first set contains places which can be part of the left-hand side of the outer synchronization. The first operator of each element in this set is $\|_{\{a\}}$. This implies that the set only consists of the first element of our previous list of three places. The second set contains places for the right-hand side of the outer synchronization. In this case, the first operator of each element is ${}_{\{a\}}\|$. This leads to the remaining two places being in the set. The goal is to find sufficiently large, non-empty subsets of both sets which can perform a without the restriction of $\|_{\{a\}}$ and ${}_{\{a\}}\|$, respectively.

There is only one non-empty subset of the set with one element and therefore it is necessary to derive the transition a for $a \rightarrow_S A$. This is possible via the rule TS (cf. Fig. 6 (a)). Next, it is required to find a subset for the second set. Since the elements both have an inner parallel operator including a in the synchronization alphabet the only choice is the set itself

as its own non-empty subset. This implies that the transition a has to be derived for the places $(a \to_E B)\|_{\{a\}}$ and ${}_{\{a\}}\|(a \to_S C)$. The simplified SOS-rule for synchronization SY-PRE from Fig. 9 is applicable. It constitutes a special case of the rule for general synchronization SYNC (cf. Fig. 10). The premise can be fulfilled by applications of the rules ET for $a \to_E B$ and ST for $a \to_S C$. The two rules can be found in Fig. 6. The resulting derivation tree is depicted in Fig. 19.

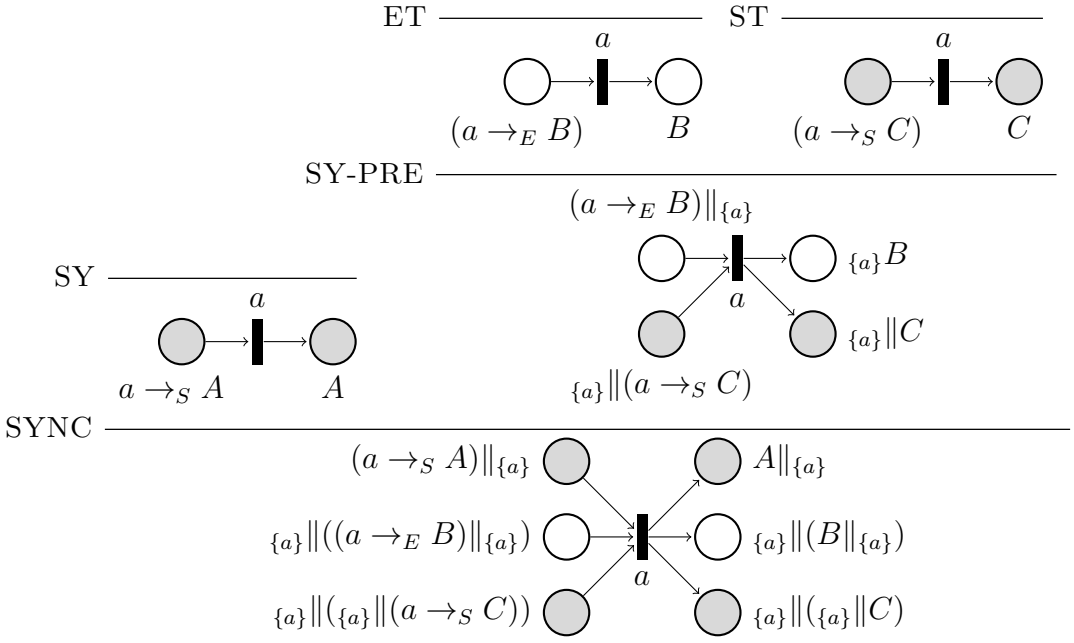


Figure 19: An example derivation of a transition which requires three tokens to fire is displayed (cf. root of the derivation tree depicted on the bottom of the tree). The first rule used is SYNC, followed by ST for the left-hand side; on the right-hand side first SY-PRE is applied; the resulting two premises are fulfilled by ET and ST.

No additional transitions are derivable without unfolding the recursion of the Petri games A, B, or C. Each derivation tree for ${}_{\{a\}}\|((a \to_E B)\|_{\{a\}})$ and ${}_{\{a\}}\|(_{\{a\}}\|(a \to_S C))$ will use the exact same SOS-rules as the previous example.

If we consider $a \to_S A \underset{\{a\}}{\|} (a \to_E B \underset{\emptyset}{\|} a \to_S C)$ with the subtle change that the second synchronization happens over the empty synchronization alpha-

bet, then we can derive two transitions labeled by a before unfolding the recursion of A, B, or C. Two places take part in both transitions, respectively. The place labeled by $(a \rightarrow_S A)\|_{\{a\}}$ is part of both transitions, i.e., for the first transition it synchronizes with the place labeled by ${}_{\{a\}}\|((a \rightarrow_E B)\|_{\{\emptyset\}})$ and for the second one with the place labeled by ${}_{\{a\}}\|({}_{\{\emptyset\}}\|(a \rightarrow_S C))$. At the position of using the rule SY-PRE in Fig. 19, we can instead derive a transition for the outer synchronization in two ways. The first way uses SLS instead of SY-PRE to make a step on the left-hand side inside the synchronization and then TE to derive the transition a for $(a \rightarrow_E B)\|_{\{\emptyset\}}$. For ${}_{\{\emptyset\}}\|(a \rightarrow_S C)$, the second way uses the rule SRS instead of SY-PRE to make a step on the right-hand side inside a synchronization and then the rule ST to derive the transition a.

The brackets for the parallel operator have a big impact on the resulting Petri game. For example, the Petri game $a \rightarrow_S A \underset{\{a\}}{\|} (a \rightarrow_E B \underset{\emptyset}{\|} a \rightarrow_S C)$ has as we have seen two transitions labeled by a where both transitions have two players taking part. This means that the player, which will behave like A after performing the transition a, has to decide for one of the two transitions and only two of the three places labeled by $A\|_{\{a\}}$, ${}_{\{a\}}\|(B\|_{\emptyset})$, and ${}_{\{a\}}\|({}_{\emptyset}\|C)$ are reached. On the other hand $(a \rightarrow_S A \underset{\{a\}}{\|} a \rightarrow_E B) \underset{\emptyset}{\|} a \rightarrow_S C$ also has two transitions but the transition leading to ${}_{\{a\}}\|({}_{\emptyset}\|C)$ is not a synchronization. This means that both transitions can occur during the same run of the game and all three places $(A\|_{\{a\}})\|_{\emptyset}$, $({}_{\{a\}}\|B)\|_{\emptyset}$, and ${}_{\emptyset}\|C$ can be reached.

4.3 Restriction to Recursion

The second restriction to recursion postulates that (nested-)self recursion including the synchronization operator is prohibited. The reason for this restriction is that the "$\|$"-operator might be collected infinitely often. An illustration for the collection of the operator when ignoring the restriction is given by the example Petri game P with $P = a \rightarrow_S (P \underset{\emptyset}{\|} STOP_S)$. Notice that the recursion is guarded by the transition a which implies that the recursion accords to the first restriction to recursion.

The resulting Petri game is depicted in Fig. 20 up to a recursion depth of two. All places belong to the system. Transitions are counted starting from the place P. The first transition shows the unfolding of the recursion and is reasoned by the rule RS. The second transition is labeled by a and derived by the rule ST. The third transition is the opening of the synchronization. It is derived by the rule SO and shows that the infinity of the Petri game comes from the attached opened synchronization operator $\|_{\emptyset}$. It leads to a

new place labeled by $P\|_\emptyset$. From this place, analog transitions can be derived resulting in new places which all have $\|_\emptyset$ attached. All these derivations start with the rule SLS followed by the same rules as before. The place $(P\|_\emptyset)\|_\emptyset$ is reached. From this place, analog transitions and places can be derived which have $\|_X$ attached twice. This procedure can be repeated infinitely often.

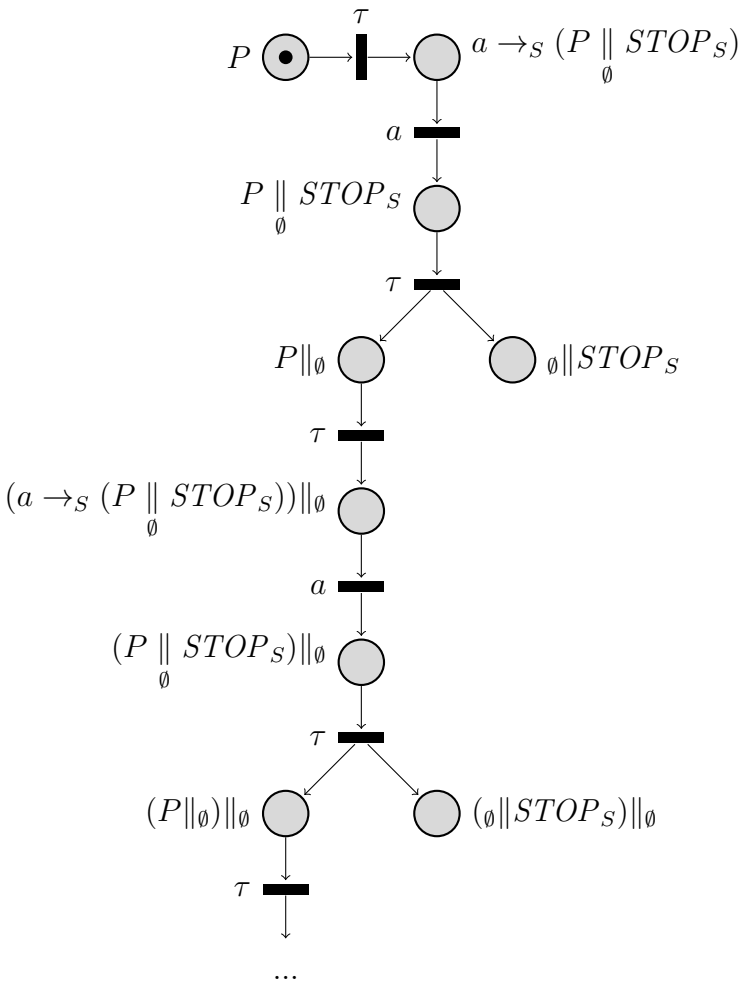


Figure 20: The derived Petri game P for $P = a \to_S (P \underset{\emptyset}{\|} STOP_S)$ up to a recursion depth of two is depicted. Notice that the definition of P is not allowed according to the restrictions to recursion.

4.4 Synchronization of Recursive Petri Games

In this subsection, an example is presented that illustrates that it suffices for the semantics to unfold recursion only once. If the recursion contains a loop then the place starting the recursion is reached again after the first unfolding. In this case, both places have equal labels which implies that they fall together and therefore the place will not be processed a second time. In order for the semantics to deal with this case, the derived transitions have to cover all cases which can be reached if the recursion would be unfolded infinitely often. This is explained best using an example.

The example consists of the Petri games $A_1 = a \rightarrow_E a \rightarrow_E a \rightarrow_E A_1$ and $A_2 = a \rightarrow_E a \rightarrow_E A_2$. The first Petri game A_1 can perform the transition a three times before the recursion is unfolded again whereas A_2 can perform it only two times per recursion. A_1 and A_2 are forced to synchronize on a in the Petri game $A_1 \underset{\{a\}}{\|} A_2$.

A possible approach which is not taken by the presented semantics would unfold A_1 two times and A_2 three times in order to achieve six synchronous transitions of a before both games would reach again the places after the opening of the initial synchronization. This approach would lead to 18 places (twelve places are caused by the preceding places of the six synchronous transitions labeled by a. One place is used for the initial opening of the synchronization. A_1 causes two places where it needs to be unfolded whereas A_2 causes three such places). The first transition would open the synchronization. Then, two places would synchronize on a leading to the next two places. Each place would have exactly one outgoing transition. The recursion of A_1 and A_2 would be unfolded when necessary. The last two places would return to the places representing the opened synchronization. This approach is difficult to perform in a structural operational semantics because the least common multiple for the unfolding of recursion needs to be calculated.

The presented semantics incorporates another approach. The result of this approach is depicted in Fig. 21. The minimal number of places is produced but additional transitions are derived, i.e., places will have more than one outgoing transition in this example. The first place is labeled by $A_1 \underset{\{a\}}{\|} A_2$ and has one outgoing transition. It is labeled by τ and represents the opening of the synchronization. The transition results in the two places $A_1\|_{\{a\}}$ and ${}_{\{a\}}\|A_2$ which represent A_1 and A_2 before the opening of the recursion. Both places have one outgoing transition labeled by τ representing the unfolding of the recursion, respectively. The places abbreviated by A_{13} and A_{22} are reached. The abbreviation A_{13} stands for $(a \rightarrow_E a \rightarrow_E a \rightarrow_E A_1)\|_{\{a\}}$

meaning that process A_1 can still perform three times a before the next unfolding of recursion is necessary. According abbreviations are used for all remaining places implying that all outgoing transitions of A_{13} have to reach the place A_{12}. In an analog fashion, A_{22} represents ${}_{\{a\}}\|(a \rightarrow_E a \rightarrow_E A_2)$.

All remaining transitions will be labeled by a and the derivation is based on the rule SY-PRE each time. Transitions are derived for A_{13} first. The only partner existing so far is A_{22}. A transition is derived resulting in the places A_{12} and A_{21}. The abbreviation A_{12} describes that A_1 can perform two instances of a before the recursion needs to be unfolded again, i.e., it has the form $(a \rightarrow_E a \rightarrow_E A_1)\|_{\{a\}}$. The place A_{21} represents ${}_{\{a\}}\|(a \rightarrow_E A_2)$. Now another transition is derivable for A_{13} because the new partner A_{21} was produced. The transition leads to A_{12} and ${}_{\{a\}}\|A_2$. No additional places are produced and all transitions for A_{13} have been derived.

Now, transitions are derived for A_{22}. The transition together with A_{13} has been derived already. A_{12} is a possible partner and the resulting transition leads to the newly created place A_{11} and the already existing place A_{21}. The newly created place A_{11} gives A_{22} another partner for synchronization. The transition leads to $A_1\|_{\{a\}}$ and A_{21}.

After this, transitions for the place A_{12} are derived. The transition synchronizing with A_{22} has already been derived. The only new transition results from a synchronization with A_{21} leading to the already existing places A_{11} and ${}_{\{a\}}\|A_2$. Next, for A_{21} a transition together with A_{11} is derived leading to $A_1\|_{\{a\}}$ and ${}_{\{a\}}\|A_2$. For A_{11}, no further transitions can be derived and all transitions for all places have been derived.

The transitions labeled by a in the resulting Petri game (cf. Fig. 21) are depicted together with the number which makes them unique in order to show the order in which they have been derived. This order does not coincide with the order in which the transitions are fired when the game is played. The produced Petri game results in the following path of transitions labeled by a: $(a_1,\ a_5,\ a_4,\ a_2,\ a_3,\ a_6)^\omega$. This shows that the derived Petri game symbolizes the intended game where the play is repeated after six synchronous transitions despite only producing eight places.

For completeness the abbreviations for places are stated:

$$\begin{aligned}
A_{11} &= (a \rightarrow_E A_1)\|_{\{a\}} \\
A_{12} &= (a \rightarrow_E a \rightarrow_E A_1)\|_{\{a\}} \\
A_{13} &= (a \rightarrow_E a \rightarrow_E a \rightarrow_E A_1)\|_{\{a\}} \\
A_{21} &= {}_{\{a\}}\|(a \rightarrow_E A_2) \\
A_{22} &= {}_{\{a\}}\|(a \rightarrow_E a \rightarrow_E A_2)
\end{aligned}$$

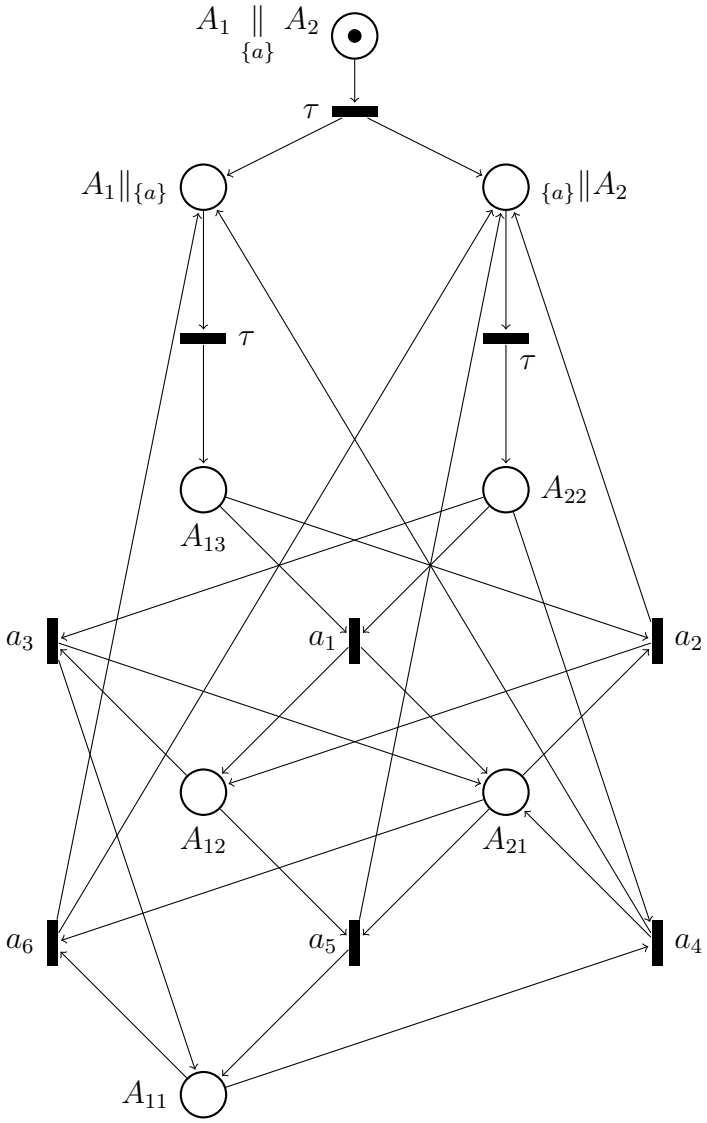


Figure 21: The Petri game $A_1 \underset{\{a\}}{\|} A_2$ for $A_1 = a \rightarrow_E a \rightarrow_E a \rightarrow_E A_1$ and $A_2 = a \rightarrow_E a \rightarrow_E A_2$ is depicted. Transitions are annotated with the number which marks the order of the derivation. The example illustrates that the semantics handles recursion correctly.

4.5 Distributed Alarm System

This and the next example show how more complex Petri games can be expressed in the semantics. The examples are introductory examples from the paper by Finkbeiner and Olderog introducing Petri games [2]. The first example results in the Petri game from Fig. 1 of [2] modeling a distributed alarm system. Distributed alarm system is abbreviated by DAS in figures and expressions in the following.

The burglar is represented by the environment. It can decide to break in at position A or B with the transitions t_A and t_B, respectively. Afterwards it may synchronize on transitions to bad places. The alarm system consists of the two distributed parts A and B which are placed at the positions of the same name. The goal of the alarm system is to signal the correct location of a break-in at both locations if and only if a break-in occurs. A can either decide that it recognized a break-in via the transition A_2, get information from B via t_{BB}, or recognize a break-in of the burglar at A with a synchronization on the transition t_A. After the detection, A can decide to either keep the information about the break-in (transition A_1) or to share it with B via t_{AA}. In all four cases, a place is reached where A can signal a break-in at position B via transition AB or at position A via transition AA. B works in an analog manner except that BB signals a break-in at position B and BA at A.

Two transitions to bad places are given. The transition $\bot_R$ shows that the burglar broke in at A but the alarm system at B signaled a break-in at B. The transition $\bot_L$ represents the opposite situation when the burglar broke in at B but the alarm system at A signaled a break-in at A. In the original paper, the example is explained further and additional bad behaviors are defined.

The following expression produces the described situation:

$$
\begin{aligned}
Env &= t_A \to_E EA \sqcap t_B \to_E EB \\
EA &= \bot_R \to_E EA \\
EB &= \bot_L \to_E EB \\
A &= t_A \to_S (A_1 \to_S pA \mid t_{AA} \to_S pA) \mid A_2 \to_S pA \mid t_{BB} \to_S pA \\
pA &= AB \to_S STOP_S \mid AA \to_S \bot_L \to_S FAIL \\
B &= t_B \to_S (B_1 \to_S pB \mid t_{BB} \to_S pB) \mid B_2 \to_S pB \mid t_{AA} \to_S pB \\
pB &= BB \to_S \bot_R \to_S FAIL \mid BA \to_S STOP_S \\
DAS &= Env \underset{\{t_A, t_B, \bot_L, \bot_R\}}{\parallel} (A \underset{\{t_{AA}, t_{BB}\}}{\parallel} B)
\end{aligned}
$$

The derived Petri game according to the semantics is depicted in Fig. 22. It begins with two initial transitions from its initial place to the places repre-

senting Env, A, and B. After these transitions, all environment places have the form $e\|_{\{t_A,t_B,\perp_L,\perp_R\}}$, all places belonging to the player A have the form ${}_{\{t_A,t_B,\perp_L,\perp_R\}}\|(a\|_{\{t_{AA},t_{BB}\}})$ and all place belonging to the player B have the form ${}_{\{t_A,t_B,\perp_L,\perp_R\}}\|({}_{\{t_{AA},t_{BB}\}}\|b)$ where e, a, and b stand for the changing parts of the label.

The opened synchronization operators stay attached for the lifetime of a local player. This implies that two bad places are produced where the original figure has only one. Furthermore, one transition labeled by τ followed by another transition labeled by τ is added in the beginning in order to open the synchronization. One τ-transition is added to close the synchronization of the system players when they reach S_7 and S_{15}, respectively. Notice, that this represents the situation that the alarm system at A signals a burglar at B and the alarm system at B signals a burglar at A, meaning that this transition exists only because of the omitted additional transitions to bad places. The remaining transitions labeled by τ are caused by the unfolding of recursion. The structural similarity to the original Petri game is obvious while keeping the number of added places because of the semantics low.

The following abbreviations are used in Fig. 22:

$$
\begin{array}{rcl}
E_5 & = & EA\|_{\{t_A,t_B,\perp_L,\perp_R\}} \\
E_6 & = & \perp_R \rightarrow_E EA\|_{\{t_A,t_B,\perp_L,\perp_R\}} \\
E_7 & = & (t_B \rightarrow_E EB)\|_{\{t_A,t_B,\perp_L,\perp_R\}} \\
E_8 & = & EB\|_{\{t_A,t_B,\perp_L,\perp_R\}} \\
E_9 & = & \perp_R \rightarrow_E EA\|_{\{t_A,t_B,\perp_L,\perp_R\}} \\
S_1 & = & {}_{\{t_A,t_B,\perp_L,\perp_R\}}\|(A \underset{\{t_{AA},t_{BB}\}}{\|} B) \\
S_2 & = & {}_{\{t_A,t_B,\perp_L,\perp_R\}}\|(A\|_{\{t_{AA},t_{BB}\}}) \\
S_3 & = & {}_{\{t_A,t_B,\perp_L,\perp_R\}}\|((t_A \rightarrow_S (A_1 \rightarrow_S pA \mid t_{AA} \rightarrow_S pA) \mid \\
 & & A_2 \rightarrow_S pA \mid t_{BB} \rightarrow_S pA)\|_{\{t_{AA},t_{BB}\}}) \\
S_4 & = & {}_{\{t_A,t_B,\perp_L,\perp_R\}}\|((A_1 \rightarrow_S pA \mid t_{AA} \rightarrow_S pA)\|_{\{t_{AA},t_{BB}\}}) \\
S_5 & = & {}_{\{t_A,t_B,\perp_L,\perp_R\}}\|(pA\|_{\{t_{AA},t_{BB}\}}) \\
S_6 & = & {}_{\{t_A,t_B,\perp_L,\perp_R\}}\|((AB \rightarrow_S STOP_S \mid AA \rightarrow_S \perp_L \rightarrow_S FAIL)\|_{\{t_{AA},t_{BB}\}}) \\
S_7 & = & {}_{\{t_A,t_B,\perp_L,\perp_R\}}\|(STOP_S\|_{\{t_{AA},t_{BB}\}}) \\
S_8 & = & {}_{\{t_A,t_B,\perp_L,\perp_R\}}\|((\perp_L \rightarrow_S FAIL)\|_{\{t_{AA},t_{BB}\}}) \\
S_9 & = & {}_{\{t_A,t_B,\perp_L,\perp_R\}}\|({}_{\{t_{AA},t_{BB}\}}\|B) \\
S_{10} & = & {}_{\{t_A,t_B,\perp_L,\perp_R\}}\|({}_{\{t_{AA},t_{BB}\}}\|(t_B \rightarrow_S (B_1 \rightarrow_S pB \mid t_{BB} \rightarrow_S pB) \\
 & & \mid B_2 \rightarrow_S pB \mid t_{AA} \rightarrow_S pB) \\
S_{11} & = & {}_{\{t_A,t_B,\perp_L,\perp_R\}}\|({}_{\{t_{AA},t_{BB}\}}\|(B_1 \rightarrow_S pB \mid t_{BB} \rightarrow_S pB)) \\
S_{12} & = & {}_{\{t_A,t_B,\perp_L,\perp_R\}}\|({}_{\{t_{AA},t_{BB}\}}\|pB) \\
S_{13} & = & {}_{\{t_A,t_B,\perp_L,\perp_R\}}\|({}_{\{t_{AA},t_{BB}\}}\|(BB \rightarrow_S \perp_R \rightarrow_S FAIL \mid BA \rightarrow_S STOP_S)) \\
S_{14} & = & {}_{\{t_A,t_B,\perp_L,\perp_R\}}\|({}_{\{t_{AA},t_{BB}\}}\|(\perp_R \rightarrow_S FAIL)) \\
S_{15} & = & {}_{\{t_A,t_B,\perp_L,\perp_R\}}\|({}_{\{t_{AA},t_{BB}\}}\|STOP_S) \\
\perp_1 & = & {}_{\{t_A,t_B,\perp_L,\perp_R\}}\|(FAIL\|_{\{t_{AA},t_{BB}\}}) \\
\perp_2 & = & {}_{\{t_A,t_B,\perp_L,\perp_R\}}\|({}_{\{t_{AA},t_{BB}\}}\|FAIL)
\end{array}
$$

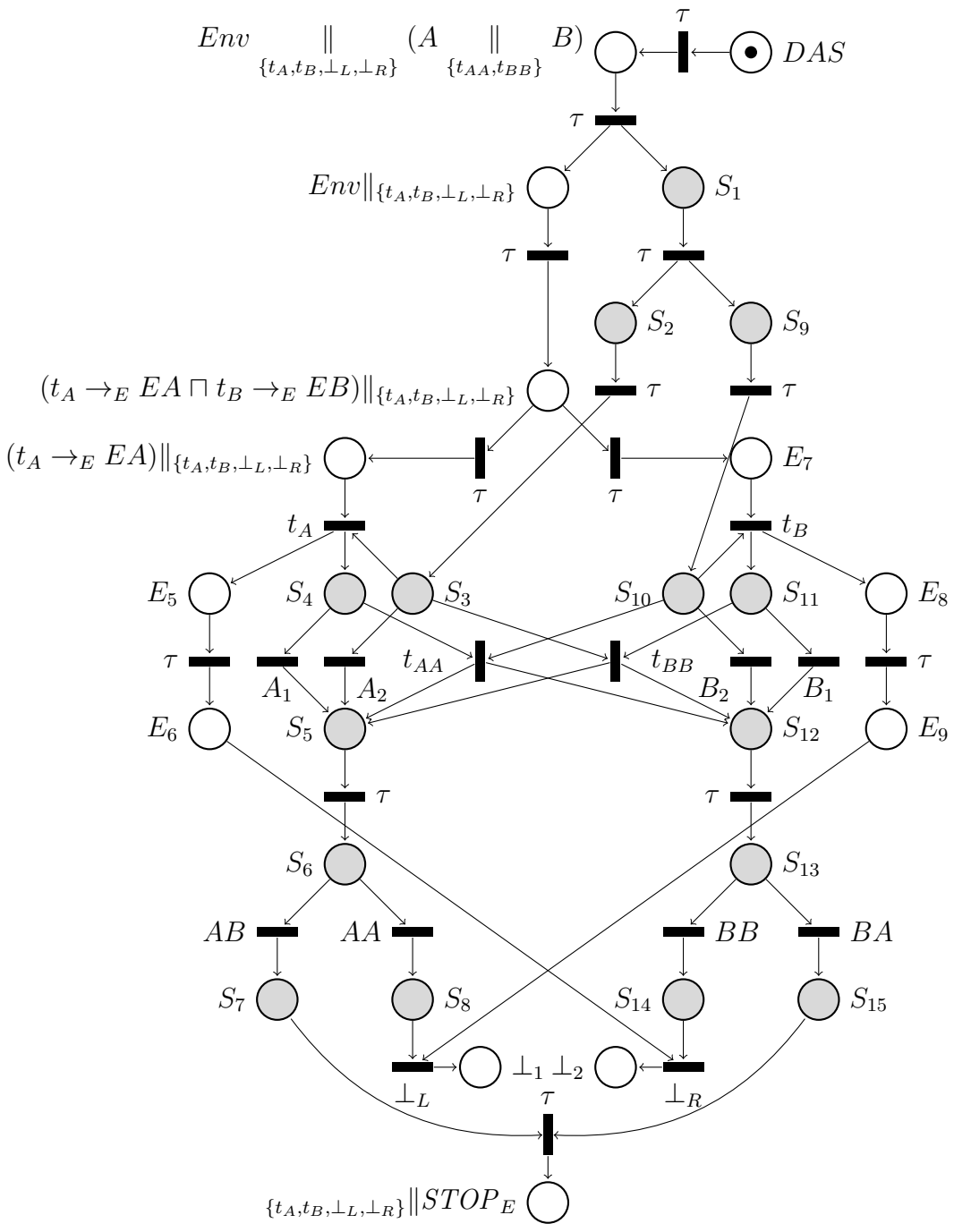


Figure 22: The derived Petri game according to the semantics for the expression DAS is depicted representing the first example from the introductory paper on Petri games. The outgoing edges from the transition $\perp_L$ to the place E_8 and from $\perp_R$ to E_5 are omitted to increase readability.

4.6 Mimicking the Environment

The Petri game from Fig. 3 of the original paper about Petri games [2] depicts the situation where two local system players have to mimic the decision of one environment player. All transitions to bad places are omitted in this example.

The environment makes a decision between two options which is represented by the "$\sqcap$"-operator. Afterwards, it allows for testing its decision via the synchronous transitions $test_1$ and $test_2$. There exist two local system players. Both have four options: they can try to test the environment's decision with $test_1$ and $test_2$, they can mimic the first decision with t_1', or they can mimic the second decision with t_2'. Synchronization only occurs on $test_1$ and $test_2$ ensuring that all three places take part in these transitions. This means that either none or both of the local system players have the information how to mimic the environment. Nevertheless, the transitions of each system player mimicking the environment occur independently of each other. The abbreviation MTE stands for mimicking the environment. The following expressions produce the intended Petri game:

$$
\begin{aligned}
Env &= test_1 \rightarrow_E STOP_E \sqcap test_2 \rightarrow_E STOP_E \\
Sys_1 &= test_1 \rightarrow_S Sys_1 \mid t_1' \rightarrow_S STOP_S \mid t_2' \rightarrow_S STOP_S \mid test_2 \rightarrow_S Sys_1 \\
Sys_2 &= test_1 \rightarrow_S Sys_2 \mid t_1' \rightarrow_S STOP_S \mid t_2' \rightarrow_S STOP_S \mid test_2 \rightarrow_S Sys_2 \\
MTE &= Env \underset{\{test_1,test_2\}}{\parallel} Sys_1 \underset{\{test_1,test_2\}}{\parallel} Sys_2
\end{aligned}
$$

The omitting of bad places leads to additional transitions which close the synchronization. The semantics will produce differently labeled places for each player meaning that both system players have their own places despite behaving equally. It also holds that without outgoing transitions all terminal places per player fall together whereas originally the system might end in EA or EB and the environment in A' or B', respectively (A and B representing the first and the second decision, respectively.). This does not happen as soon as the transitions to bad places are added. The derived Petri game is delineated in Fig.23.

The following abbreviations for labels on places are used:

$$
\begin{aligned}
E_4 &= ((test_1 \rightarrow_E STOP_E \sqcap test_2 \rightarrow_E STOP_E)\|_{\{test_1,test_2\}})\|_{\{test_1,test_2\}} \\
E_5 &= ((test_1 \rightarrow_E STOP_E)\|_{\{test_1,test_2\}})\|_{\{test_1,test_2\}} \\
E_6 &= ((test_2 \rightarrow_E STOP_E)\|_{\{test_1,test_2\}})\|_{\{test_1,test_2\}} \\
E_7 &= (STOP_E\|_{\{test_1,test_2\}})\|_{\{test_1,test_2\}} \\
S_2 &= (_{\{test_1,test_2\}}\|Sys_1)\|_{\{test_1,test_2\}} \\
S_3 &= (_{\{test_1,test_2\}}\|(test_1 \rightarrow_S Sys_1 \mid t_1' \rightarrow_S STOP_S \mid t_2' \rightarrow_S STOP_S \mid \\
&\quad test_2 \rightarrow_S Sys_1))\|_{\{test_1,test_2\}} \\
S_4 &= (_{\{test_1,test_2\}}\|STOP_S)\|_{\{test_1,test_2\}} \\
S_5 &= {}_{\{test_1,test_2\}}\|(test_1 \rightarrow_S Sys_2 \mid t_1' \rightarrow_S STOP_S \mid t_2' \rightarrow_S STOP_S \mid \\
&\quad test_2 \rightarrow_S Sys_2) \\
S_6 &= {}_{\{test_1,test_2\}}\|STOP_S)
\end{aligned}
$$

The relation between the two Petri games may not be as obvious as in the previous example. There are two reasons for that. The first one is that the two system players do not fall together resulting in more places. The second reason is caused by the recursion of the two system players. After the transitions $test_1$ and $test_2$ they have to return to the place where a transition is necessary to unfold recursion.

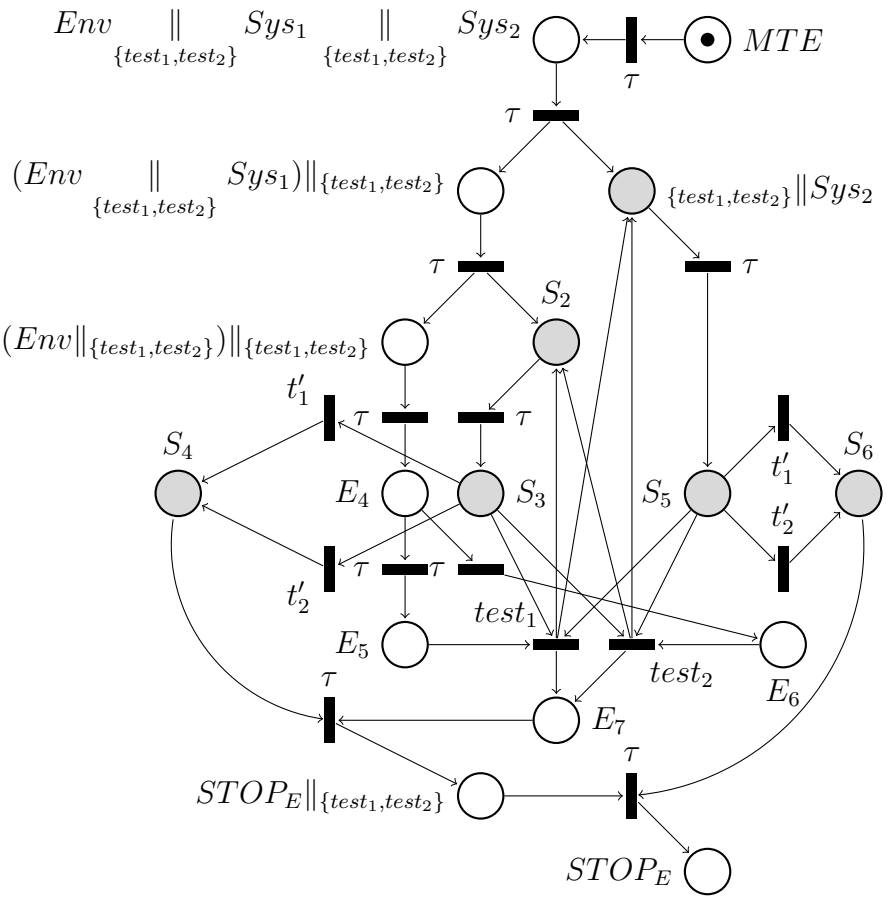


Figure 23: The derived Petri game according to the semantics for the expression MTE is depicted representing the third example from the introductory paper on Petri games.

5 Related Work

This section about related work is structured in the following manner. First, we sketch work on synthesis in general. In turn, we delineate alternative semantic concepts connecting CSP and Petri nets.

5.1 Synthesis

Raskin, Samuelides, and Van Begin [12] present an alternative definition of games based on Petri nets. In their definition, Petri nets are used to define two-player monotonic games which are turn-taking. Transitions of the underlying Petri net are divided either to belong to the system or to the environment. The game is played on configurations which are represented by a marking of the underlying Petri net and an annotation whether the current player is the system player or the environment player. It is shown that for all games which are constructed this way the coverability problem is decidable and the corresponding winning strategy can be synthesized [12]. The coverability problem is a subclass of the reachability problem where a certain ordering for the underlying game model exists. In a reachability game, the system's goal is to reach certain configurations. The presented games differ from Petri games as they are turn-taking and distinguish *transitions* to belong either to the system or to the environment. In contrast, Petri games do not enforce a turn-taking game development and distinguish places to belong either to the system or to the environment.

Originating from their work on synthesis of reactive systems [9], Pnueli and Rosner introduce synthesis in the setting of distributed reactive systems [10]. Their work aims at a strategy to fulfill a given specification for a distributed system. Here, the strategy is supposed to be spread over several parts of a system. In the general case, this problem remains undecidable. However, it is decidable for the restricted subclass of hierarchical architectures [11]. An architecture is hierarchical if there exists a pipeline between the components of the architecture. This pipeline can be used to exchange information. Petri games tackle this distributed setting although the system and the environment are distributed into several parts. So far, it is only shown for Petri games that they are decidable for a bounded number of system players and a single environment player [2]. On an intuitive level, this result goes together with Pneuli and Rosner's result because the system players have to exchange as much information as possible to have a chance to make the correct decision or to be certain that there exists no correct decision, respectively.

Mohalik and Walukiewicz [6] introduce so called distributed games. A dis-

tributed game consists of several system players and one environment player. According to this definition, each system player can communicate with the environment player only. A strategy controls each system player locally, i.e., without using knowledge of other system places. The only information, system player can share implicitly, is that they have performed a turn in the game. In the general case, these games are undecidable but it has been shown that two theorems allow the simplification of certain distributed games like pipelines [6]. The simplification reduces for these distributed games the number of system players to one which makes the games decidable, respectively. Furthermore, it is shown that the corresponding strategy can be synthesized. This game model is the opposite to Petri games where players exchange all their history on a synchronous transition.

5.2 Alternative Semantics

Olderog [7] presents a semantics that relates CSP to Petri nets. This approach is also based on SOS-rules. Notice that the semantics presented in this thesis is closely related to Olderog's. He tries to minimize the number of transitions which are labeled by τ. Olderog's semantics is based on sets of processes which initially are unfolded via a decomposition and expansion function dex. This set theoretic approach always preserves a set of processes for which further transitions have to be derived. The semantics also opens the synchronization operator into $\|_X$ and $_X\|$ and unfolds recursion until all parts are guarded. In contrast, the semantics of this thesis uses τ more often and deals with all operators via SOS-rules resulting in some overhead to ensure the termination of the derivation.

Llorens and his colleagues [5] present another semantics from CSP to Petri nets. It is used to build a software tool. The authors claim that the tool transforms a CSP specification into a closely related Petri net. In this context, closely related means that the connection between the CSP specification and the Petri net is clearly visible. This goal is achieved by determining that one process is represented by one token and its progress through the Petri net is clearly visible despite any overhead produced by the automatic generation. The close relationship between the CSP expression and its resulting Petri game is a goal the semantics of this thesis achieves as well. Furthermore, the idea of systematically deriving all transitions in a breath-first search manner is also used in the stated paper.

6 Conclusion

This section first provides a summary of the presented material. Moreover, future work is discussed. Especially, we focus on possible extensions and changes to the semantics as well as further applications of the semantics.

6.1 Summary

We have modified a subset of the syntax of CSP in order to derive Petri games. Petri games define games on an underlying Petri net. The transformation follows a structural operational semantics for the derivation of transitions. For each used syntactical construction, a set of SOS-rules is presented. The semantics shows that it is possible to define safety games based on CSP.

$STOP_S$, $STOP_E$, and *FAIL* represent places in a Petri game without outgoing transitions. Therefore, no SOS-rules exist for these cases. The subscripts S and E define that the places belong to the system or to the environment, respectively, whereas *FAIL* symbolizes a bad place. The bad places are bad from the point of view of the system, i.e., the environment aims at reaching a bad place whereas the system wants to prevent this. A winning strategy for the system prevents the reaching of a bad place for all possible behaviors of the environment. Whether such a strategy exists is called the realizability question and the automatic generation of a winning strategy is called the synthesis problem.

The syntactic constructions $a \rightarrow_S P$ and $a \rightarrow_E P$ are used to describe that a system or an environment place can fire the transition a and behave like P afterwards, respectively. A clear distinction between system and environment, respectively, enables the system to stop at a place in order to prevent reaching a bad place. Here, the assumption is made that the system's strategy is deadlock avoiding in order to prevent Petri games from being trivial.

Furthermore, the synchronization operator $P \underset{X}{\parallel} Q$ is introduced. It enforces synchronization on all transitions of P and Q which are in the synchronization alphabet X. A transition $a \in X$ can only occur if it occurs in P and Q at the same time. The opened synchronization operators $P\|_X$ and ${}_X\|Q$ characterize that synchronization on a process is defined for its lifetime. SOS-rules for the opening and the closing of the synchronization are presented. Additional rules are given for the derivation of transitions during a synchronization which are not labeled by τ. These transition can be synchronous or local depending on whether the transition is in X or not.

The system and the environment can perform choices, respectively. The system choice operator $a \rightarrow_S P \mid b \rightarrow_S Q$ enables the system to decide between performing the transition a or b. The decision for a results in a behavior according to P afterwards whereas a decision for b leads to Q, respectively. The system's decision is finalized by performing either a or b. In contrast, the choice operator of the environment in the example $a \rightarrow_E P \sqcap b \rightarrow_E Q$ basically can perform the same decision as the system before. However, the environment finalizes its decision with a τ-transition, respectively, before performing the transition a or the transition b. Thus during a synchronization the system cannot dominate the environment. This specific construction ensures meaningful Petri games.

The deterministic choice operator ($|$) of CSP corresponds to the choice operator of the system in a Petri game, i.e., the strategy controls which decision is taken. This decision can be seen as deterministic from a game-theoretic point of view. The non-deterministic choice operator ($\sqcap$) of CSP represents a choice of the environment. The environment is assumed to behave non-deterministically.

In order to allow for loops in Petri games, the recursion operator is introduced. The derivation of transitions for the place L with the definition of $L = P$ leads to one outgoing τ-transition from L to the starting place of the Petri game P. Two SOS-rules are introduced depending on whether the resulting place P belongs to the system or to the environment. This ensures that the place labeled by L can be assigned to the system or to the environment.

In various examples, derivation trees for single transitions are illustrated and explained as well as the resulting Petri games of larger expressions are outlined. The derivation trees (cf. Fig. 18 and Fig. 19) illustrate why the semantics terminates and produces the intended transitions. Furthermore, the example in Fig. 20 illustrates in which respect the recursion operator has to be restricted. The Petri game in Fig. 22 shows how CSP can be used to define a distributed alarm system for which a winning strategy can be synthesized.

The second example in Fig. 23 illustrates how a distributed system can be synthesized in which two local system players have to mimic the decision of an environment player. The latter example shows a weakness of the semantics in the handling of places with more than one token. If two tokens can perform the same transitions they could reside in the same place before each transition. The presented semantics, however, results in separate places for each token. We come back to this issue in the next subsection about future work.

6.2 Future Work

In this subsection, future work is discussed. The first part deals with additional operators for the semantics. In the second part, optimizations to the semantics are discussed regarding the reduction of occurrences of τ-transitions. Afterwards, it is explained why an equivalence for Petri games is helpful in order to analyze the semantics and to define rules which extend and simplify CSP-expression and the corresponding Petri games.

6.2.1 Additional Operators

An obvious extension of the semantics is the sequential operator $P\,;Q$ which makes it possible to serialize two Petri games P and Q, i.e., the Petri game P has to terminate before the Petri game Q is started. The closing of synchronization is defined with the sequential operator in mind permitting to define a Petri game produced by the presented semantics as terminated when a place labeled by $STOP_{S/E}$ is reached. Unfortunately, the sequential operators requires unexpectedly high number of rules for the case before the first Petri game terminates. In fact, nearly every rule would require a modified counterpart for this case. The sequential operator is left out in order to increase readability of the thesis. Notice that all examples could circumvent the lack of the sequential operator without loss of expressibility. The sequential operator is most useful when analyzing concurrency since it requires a notion of termination.

The example about the two system players mimicking one environment player elicits the wish for an operator which produces places with more than one token in it. The presented semantics produces different places for each token making the produced Petri game less readable. The goal of the operator is to give the opportunity for each transition of the places with more than one token to have either all or only one token take part in a transition.

Another interesting idea is to define a hiding operator from CSP in a meaningful fashion for Petri games. The key question about how to handle the hiding of a synchronous transition between the system and the environment remains unclear. This operator would make it possible to answer questions about which transitions of a Petri game are necessary for the system or the environment to win the game. The operator would be useful to tackle game-theoretic questions.

6.2.2 Reduction of Occurrences of τ-Transitions

In the semantics, τ-transitions are produced for the recursion operator, the choice operator of the environment, and the synchronization operator. For

the choice operator, the τ-transitions enable an indication of the decision of the environment in order to prevent it from being forced to follow the system. The recursion operator can work without τ-transitions whereas the synchronization operator can safe some τ-transitions when more than one synchronization operator occurs in a series. However, the reduction of occurrences of τ-transitions has purely cosmetic consequences making the produced Petri games more handy while keeping the expressiveness.

6.2.3 Equivalence between Petri Games

It is desirable to proof the compositionality of the presented semantics. For that purpose, it is necessary to define under which circumstances two Petri games are equivalent. Intuitively, a third Petri game should not be able to distinguish between two equivalent Petri games using a certain kind of test, i.e., a testing scenario is required. This idea is inspired by Hennessy's testing equivalence about how processes may or must behave [3]. A possible testing scenario is the synchronization on an arbitrary synchronization alphabet of the third Petri game with the two possibly equivalent Petri games, respectively. For this case, all basic equivalences which were tested so far failed. This leads to the question whether the testing scenario gives the third Petri game too much power or whether a more sophisticated equivalence might exist. We want to conjecture the second case and will address this topic as future work as it gives rise to interesting research questions we summarize in the next subsection.

6.2.4 Analysis of the Semantics

The equivalence addressed in the previous subsection would enable to check whether rules for CSP relating two CSP-expressions also hold for Petri games and the defined equivalence. This allows checking existing rules for extension and simplification in order to broaden the understanding of Petri games and to simplify Petri games. It might also be possible to find additional rules which do not hold for CSP but are in fact true for Petri games.

7 References

[1] J. Esparza and K. Heljanko. *Unfoldings - A Partial-Order Approach to Model Checking*. EATCS Monographs in Theoretical Computer Science. Springer-Verlag, 2008.

[2] B. Finkbeiner and E.-R. Olderog. Petri Games: Synthesis of Distributed Systems with Causal Memory. In *Proceedings Fifth International Symposium on Games, Automata, Logics and Formal Verification, GandALF 2014, Verona, Italy, September 10-12, 2014.*, volume 161 of *EPTCS*, pages 217–230, 2014.

[3] M. Hennessy. *Algebraic Theory of Processes*. The Foundations Of Computing Series. MIT press, 1988.

[4] C.A.R. Hoare. *Communicating Sequential Processes*, volume 178. Prentice-Hall Englewood Cliffs, 1985.

[5] M. Llorens, J. Oliver, J. Silva, and S. Tamarit. Generating a Petri net from a CSP specification: A semantics-based method. *Advances in Engineering Software*, 50:110–130, 2012.

[6] S. Mohalik and I. Walukiewicz. Distributed Games. In *FST TCS 2003: Foundations of Software Technology and Theoretical Computer Science*, pages 338–351. Springer, 2003.

[7] E.-R. Olderog. *Nets, Terms and Formulas: Three views of Concurrent Processes and Their Relationship*, volume 23. Cambridge University Press, 2005.

[8] G.D. Plotkin. A Structural Approach to Operational Semantics. *DAIMI FN–19*, 1981.

[9] A. Pnueli and R. Rosner. On the Synthesis of a Reactive Module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pages 179–190. ACM, 1989.

[10] A. Pnueli and R. Rosner. On the Synthesis of an Asynchronous Reactive Module. In *Automata, Languages and Programming*, pages 652–671. Springer, 1989.

[11] A. Pnueli and R. Rosner. Distributed Reactive Systems are Hard to Synthesize. In *Foundations of Computer Science, 1990. Proceedings., 31st Annual Symposium on*, pages 746–757. IEEE, 1990.

[12] J.-F. Raskin, M. Samuelides, and L. Van Begin. Petri Games are Monotonic but Difficult to Decide. Technical report, Université Libre De Bruxelles, 2003.

[13] A.W. Roscoe. *The Theory and Practice of Concurrency*, volume 1. Prentice Hall Englewood Cliffs, 1998.

[14] W. Thomas. Infinite Games and Verification. In *Computer Aided Verification*, pages 58–65. Springer, 2002.