UNIVERSITÀ DEGLI STUDI DI MILANO-BICOCCA
SCUOLA DI DOTTORATO

DIPARTIMENTO DI INFORMATICA, SISTEMISTICA E COMUNICAZIONE
PHD PROGRAM IN COMPUTER SCIENCE—CYCLE XXXV

# Formal analysis of information flow and control properties in Petri nets

Federica Adobbati

Matricola 764300

Supervisor: Luca Bernardinello

Tutor: Gianluca Della Vedova
Coordinator: Leonardo Mariani

Academic year 2021/2022

## Abstract

In this work we study formal techniques for analysing information flow and control properties in distributed systems modelled with Petri nets. The first problem that we tackle consists in checking whether an agent observing only part of a system is able to gather information about the hidden part. We study this problem starting from two formal relations defined in the literature on the transitions of the Petri net: *reveals* and *excludes*. *Reveals* models positive information flow, meaning that the observation of a transition gives information about the occurrence of another one; *excludes* models negative information flow, meaning that observing a transition gives information about the non-occurrence of another one. We define some generalizations of these two relations and propose several algorithms to compute them on a particular class of Petri nets.

We then consider a control problem: we assume that an agent can control only some transitions on the Petri net and observe only some places; we want to know whether the agent is able to enforce certain properties on the system. We model the problem as a two-player asynchronous game on the Petri net and study notions of observation that keep into account the concurrent nature of the system. We propose algorithms to find strategies in some particular cases, making use both of the unfolding and of the marking graph, and we study the relation between our model and concurrent game structures, introduced to define the game-based temporal logic ATL.

Finally, we start to explore the use of Petri nets to model asynchronous multi-agent system. We consider a model defined on automata and show that it can be equivalently expressed with Petri nets, possibly in a more compact way.

# Contents

Chapter 1

# Introduction

The main goal of this thesis is to develop new approaches for the formal analysis and model-checking of information flow and control abilities in distributed systems.

We assume that an agent can observe parts of the distributed system, and study two classes of goals for the agent on the system. The first class is about the information that the agent can get on the unobservable part of the system through its observations. This may happen because an unobservable event is a cause or a consequence of an observable one, therefore its occurrence can be deduced without the need of a direct observation. The analysis of *information flow* may be used to model-check security requirements, as well as for system diagnosis. Some parts of the system may need to remain secret, and the presence of information flow might violate the security of the system. Security requirements can be formalized through noninterference and opacity properties. On the contrary, we may want to be able to know that a fault happened on the system, despite the impossibility of a direct observation. This is the problem studied by diagnosis.

The second class of goals is about the *control* of a system. In this case, not only the agent can observe a part of the system, but it can also control it partially, influencing its global behaviour. We are interested to analyse which properties the agent can guarantee on the system through its control. For example, the agent may try to guarantee that a service is always available, or that the system never reaches a faulty state. We formalize the control goals on the system through temporal logic formulas.

Despite the thesis is mostly focussed on the point of view of a single agent, the considered problems may be generalized to a multi-agent context. In the last part of the thesis we reason about models that could be used for the analysis of this case.

We model the distributed systems with Petri nets. Petri nets explicitly represent concurrency, and this makes them a suitable tool to model distributed systems, in which different components may modify different local parts of the global system without any need of time synchronization. Whereas synchronous systems assume the presence of a global clock marking the moments in which all the components together select which transition to occur, in *asynchronous* systems, such as the one considered in this thesis, concurrent transitions can occur at any moment independently from each other.

The two most common tools for Petri net analysis are the marking graph and the unfolding. The marking graph is a labelled transition system representing all the global states of the system. The unfolding is a Petri net explicitly representing all the possible executions of the system. The main advantage of the unfolding with respect to the marking graph is that it preserves the information about concurrency on the system, whereas the marking graph hides it, creating several difficulties in the analysis of properties which are influenced by concurrency. On the other hand, transition systems are more studied in the literature, and the techniques developed on them can be very helpful for the analysis of some properties of Petri nets. In this thesis we will use both these tools.

## 1.1 Outline and contribution

The rest of the thesis is organized as follows. Chap. 2 provides the background on Petri nets and the notation used in the rest of the thesis. In particular, the chapter presents the classes of Petri nets used in the thesis and discusses some of their features, formalizes the notions of unfolding and marking graph, and presents the basic notions of the synthesis of Petri nets through region theory.

In Chap. 3, we study a set of formal relations for the analysis of information flow between transitions of a Petri net. We consider relations based on the notions of *reveals* and *excludes* [70, 21]. Reveals was introduced in the literature to study positive information flow between transitions; it is used to express that observing a transition reveals that another transition occurred, or will inevitably occur. On the contrary, excludes models negative information flow; it expresses that observing a transition excludes the occurrence in the past or in the future of another one. In Sec. 3.1 we recall the definitions in the literature for reveals and excludes on the transitions of a Petri net, and we introduce new definitions, in which we assume that the agent can count a certain number of repeated occurrences of a group of transitions (Sec. 3.1.2). The definitions in Sec. 3.1.2 and the study of the complexity of reveals and excludes in the class of 1-safe Petri nets in Sec. 3.2 are the first contribution of this thesis. The main contributions of this chapter is in Sec. 3.3, where we present a set of algorithms to compute all the relations defined in Sec. 3.1 on the class of bounded equal-conflict Petri nets. Finally, Sec. 3.4 discusses other works studying information flow and some applications of the relations presented in the chapter. Part of the results in this chapter are in [7, 6, 1].

Chap. 4 presents the second main contribution of this thesis. In Chap. 4 we define a two-player game to model a control problem on an asynchronous system modelled as a Petri net. The game is defined on the unfolding of the Petri net; we assume that one of the two players (the *user*) can control a subset of transitions, and we want to know whether it is able to force a desired behaviour on the system, defined by a temporal logic. The user can choose how to control the system on the basis of its observations, which are defined as equivalence classes of B-cuts (antichains formed by conditions) on the unfolding, its decisions are modelled by a *strategy*. Sec. 4.1 provides the formal definitions of the game and discusses some notions of observations on the basis of some examples. In particular, we focus the discussion on the relation between observation of local states, memory and concurrency. In addition to the classical notions of observations defined in games, we propose a new notion based on *stable parts* of markings [4] that keeps into account the impossibility of an observer to know in any moment the current global state of a system in a concurrent structure, due to the possible changes of the system in some locations from the time of their observation to the current time. In Sec. 4.2 and Sec. 4.3 we show how to find a winning strategy, if one exists, in some restricted cases of the game defined in Sec. 4.1. In Sec. 4.2, we check the existence of a strategy on a prefix of the unfolding for a reachability goal on the system. The result of this section is published in [3]. In Sec. 4.3 we study the relation of our game with the one defined on concurrent game structures [9] and provide an algorithm based on the marking graph to find a winning strategy when the goal can be expressed with a fragment of LTL. The result of this section is in [4]. Sec. 4.4 defines a notion of *implementable strategy*. A strategy is implementable if it can be encoded in the system by adding some places, so that in the system with the additional places, all the runs are forced to follow the strategy. We show how to check if a strategy is implementable in some particular cases. These results are in [5]. Sec. 4.5 concludes the chapter by discussing related works.

Finally, in Chap. 5 we discuss a model of multi-agent system based on Petri nets inspired to the asynchronous multi-agent systems (AMAS) defined in [79] with automata networks. In Sec. 5.1 we discuss related works, with a special focus on the definitions of AMAS in [79]. In Sec. 5.2 we discuss the main contribution of the chapter, namely a method to construct our model starting from an AMAS by using region theory, and the prove that the equivalence of the two models is preserved by the composition of agents. Finally, Sec. 5.3 present an algorithm to check whether any transition is not 1-live anymore after the composition. Part of the results on this paper are in [8].

Chap. 6 concludes the thesis and propose future directions for the presented research.

# Chapter 2

# Petri nets

Petri nets are formal tools for modelling concurrent and distributed systems. They were introduced by Carl Adam Petri in his PhD thesis [105], and then studied in many subsequent works, analysing their properties and applications, and proposing extensions to the original model. Among these works, [101, 104] provide a general overview, [81, 50, 44] focus on the complexity of some important properties of Petri nets, and [65, 132] discuss the use of Petri nets in automatic control and model-checking, respectively.

Unlike in automata, in Petri nets, local states and the effect of transitions on them are explicitly represented; a global state is a collection of local states. This is convenient for modelling distributed systems, where the occurrence of a transition may not affect most of the global state, but change only a few local states. During the last decades, several classes of Petri nets have been defined, that differ by the nature of their local states; for example, a local state can be a boolean condition, a counter, or a more complex algebraic structure. The kind of local state defined on the net determines how transitions can occur and modify the local states of the system.

In this thesis we consider the class of Place Transition (P/T) nets, where local states are interpreted as counters, and some of its subclasses. This chapter provides the basic definitions and notations used in the rest of the thesis, and it is structured as follows.

Sec. 2.1 introduces P/T systems, and describes some properties of two of their subclasses, namely *equal-conflict* P/T systems and *1-safe* systems. Sec. 2.2 and Sec. 2.3 present two of the most common models to describe the behaviour of a net: the *unfolding* [49] and the *marking graph*. Sec. 2.4 compares the two models, focussing in particular on the case of equal-conflict nets. Finally, Sec. 2.5 recalls the relation between labelled transition systems and Petri nets [11].

## 2.1 P/T systems

A P/T net is a graph with two kinds of nodes: *places* (or *conditions*) represented with circles, and *transitions* (or *events*), represented with squares. Let $P$ be the set of places, and $T$ be the set of transitions; the *flow relation* between the elements of the net is $F \subseteq (P \times T) \cup (T \times P)$ and it is represented with oriented arcs: let $x, y \in P \cup T$, if $(x, y) \in F$, the arc starts from $x$ and arrives in $y$. Finally, we define a *weight* function $W : (P \times T) \cup (T \times P) \to \mathbb{N}$ such that $W(x, y) = 0$ for each $(x, y) \notin F$. In the graphical representation, given a pair of elements $(x, y) \in F$, if $W(x, y) > 1$ we label the arc with the value of $W(x, y)$. Formally, a P/T net can be denoted as a quadruple $N = (P, T, F, W)$. In the special case in which $W(x, y) = 1$ for each $(x, y) \in F$, we can omit $W$ and write the net as $N = (P, T, F)$.

Let $A$ be an alphabet, a *labelled net* is a P/T net $N = (P, T, F, W)$ together with a function $\beta : T \to A$ mapping the transitions of the net on the alphabet.

A net $N' = (P', T', F', W')$ is a *subnet* of $N = (P, T, F, W)$ if $P' \subseteq P$, $T', \subseteq T$, and $F'$ and $W'$ are respectively equal to $F$ and $W$ restricted to the elements in $N'$.

For each element of the net $x \in P \cup T$, the *pre-set* of $x$ is the set ${}^\bullet x = \{y \in P \cup T \mid (y, x) \in F\}$, the *post-set* of $x$ is the set $x^\bullet = \{y \in P \cup T \mid (x, y) \in F\}$. If $x \in T$, its pre-set (resp. post-set) is also called set of *pre-conditions* (resp. *post-conditions*). Analogously, if $x \in P$, its pre-set and post-set are also called set of *pre-transitions* and *post-transitions* respectively. The previous notation can be extended to subsets of elements $A \subseteq P \cup T$: ${}^\bullet A = \bigcup_{x \in A} {}^\bullet x$ and $A^\bullet = \bigcup_{x \in A} x^\bullet$.

A net is *finite* if $P \cup T$ is finite, and *infinite* otherwise. A net is *T-restricted* if for any $t \in T$, ${}^\bullet t \neq \emptyset$ and $t^\bullet \neq \emptyset$. In this thesis we consider only T-restricted nets.

A *marking* is a function $m : P \to \mathbb{N}$ describing the global state of the net. A P/T *system* is a net with an initial marking, and it is denoted with $\Sigma = (P, T, F, W, m_{in})$. A marking is graphically represented with *tokens* (small black circles) inside places: for representing the marking $m$, for each $p \in P$, we draw $m(p)$ tokens in $p$.

**Example 1.** *Fig. 2.1 represents a P/T system with* 9 *places and* 8 *transitions. In this system,* $W(p_6, t_5) = W(p_6, t_6) = 2$. *The initial marking* $m_{in} : P \to \mathbb{N}$ *can be explicitly described as* $m_{in}(p_i) = 2$ *for* $1 \leq i \leq 4$, $m_{in}(p_j) = 0$ *for* $5 \leq j \leq 9$.

A transition $t \in T$ is *enabled* in a marking $m$, denoted with $m[t\rangle$, if, for each place $p \in {}^\bullet t$, $p(m) \geq W(p, t)$. If a marking $m$ does not enable any transition, we say that $m$ is a *deadlock*. An enabled transition can occur, or *fire*, producing a new marking $m'$ (denoted $m[t\rangle m'$), such that for each $p \in P$, $m'(p) = m(p) - W(p, t) + W(t, p)$.

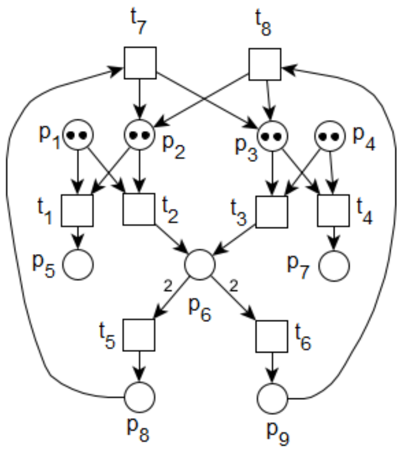


Figure 2.1: Example of P/T system.

In Fig. 2.1, $t_1$ is enabled in the initial marking. Its occurrence generate the marking $m'$ such that $m'(p_1) = m'(p_2) = m'(p_5) = 1$; $m'(p) = m_{in}(p)$ for all the other places. The marking $m''$ such that $m''(p_5) = 2$, $m''(p_7) = 2$, $m''(p) = 0$ for each $p \notin \{p_5, p_7\}$ is a deadlock.

A marking $m*$ is reachable from another marking $m$, if there is a sequence of transitions $t_1 t_2 \ldots t_n$ such that $m[t_1\rangle m_1[t_2\rangle \ldots m_{n-1}[t_n\rangle m*$; this is also denoted with $m[t_1 t_2 \ldots t_n\rangle m*$. We denote with $[m\rangle$ the set of markings reachable from $m$. A marking $m$ is *reachable* if it is reachable from the initial marking $m_{in}$, namely if $m \in [m_{in}\rangle$. A transition is *1-live* if there is a marking $m \in [m_{in}\rangle$ such that $m[t\rangle$.

P/T systems allow for the explicit encoding of different relations between transitions: if two transitions $t_1, t_2 \in T$ are enabled in a marking $m$, they are said in *conflict*, denoted with $t_1 \# t_2$, if they are both enabled and the occurrence of one disable the other, otherwise they are *concurrent*. In Fig. 2.1, $t_1$ and $t_2$ are concurrent in $m_{in}$; however in the marking $m'$ reached after the occurrence of $t_1$, they are still both enabled, but in conflict.

In some situations concurrency and conflicts overlap in such a way that it is not clear if in the execution of concurrent transitions a conflict has been solved or not. This is a so called situation of 'confusion' which has been discussed in several papers as for example in [118, 121], and which can be formalised in the following way. Let $m \in [m_{in}\rangle$, and $t_1, t_2 \in T$ be two concurrent transitions, then $(m, t_1, t_2)$ is a *confusion* at $m$ if $\text{cfl}(t_1, m) \neq \text{cfl}(t_1, m_2)$, where $\text{cfl}(t_1, m) = \{t' \in T \ : \ m[t'\rangle m_1 \ \wedge \ \neg m_1[t_1\rangle\}$, and $m_2$ is such that $m[t_2\rangle m_2$.

**Example 2.** *Fig. 2.2 illustrates the two main situations of confusion, namely* asymmetric *confusion, on the left side, and* symmetric *confusion, on the*

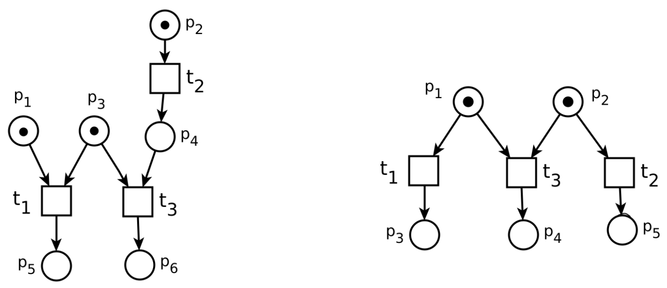


Figure 2.2: Asymmetric confusion (on the left) and symmetric confusion (on the right side).

*right side. In the net with asymmetric confusion, $t_1$ and $t_2$ are concurrently enabled, whereas $t_3$ is not. If $t_2$ fires before $t_3$, there is a conflict between $t_1$ and $t_3$ that needs to be solved; otherwise, if $t_1$ fires before the occurrence of $t_2$, then $t_3$ never becomes enabled. In the net with symmetric confusion, $t_1$ and $t_2$ are concurrently enabled, and $t_3$ is in conflict with both of them. If the conflict between $t_1$ and $t_3$ is solved in favour of $t_1$, there is no conflict anymore between $t_2$ and $t_3$, since $t_3$ has been disabled. Simmetrically if $t_2$ fires first.*

A *multiset* of transitions $U : T \to \mathbb{N}$ is *concurrently enabled* at $m \in [m_{in}\rangle$, denoted $m[U\rangle$, and called *step*, if, and only if, $\forall p \in P$, $\sum_{t \in T} U(t) \cdot W(p,t) \leq m(p)$. If $U$ is enabled at $m$, it can occur producing the new marking $m'$, denoted $m[U\rangle m'$ and defined as follows: for each $p \in P$, $m'(p) = m(p) - \sum_{t \in T} U(t) \cdot W(p,t) + \sum_{t \in T} U(t) \cdot W(t,p)$. A multiset $U : T \to \mathbb{N}$ is a *maximal step* at $m \in [m_0\rangle$ if $m[U\rangle$ and for each $t' \in T$, there is a place $p \in P$ such that: $m(p) < \sum_{t \in T} U(t) \cdot W(p,t) + W(p,t')$.

**Example 3.** *In Fig. 2.1, the multiset $U$ such that $U(t_1) = 2$, $U(t_3) = 1$, $U(t_4) = 1$, and $U(t) = 0$ for all the other elements in $T$ is a maximal step concurrently enabled in $m_{in}$. The marking produced by its occurrence is $m'$ such that $m'(p_5) = 2$, $m'(p_6) = 1$, $m'(p_7) = 1$, $m'(p) = 0$ for all the other $p \in P$.*

**Remark 1.** *To simplify the notation, in some cases a multiset $\beta : S \to \mathbb{N}$ will be denoted as $\beta = \{s^n | s \in S, \beta(s) = n > 0\}$. When $n = 1$, it will be omitted. With this notation, the multisets in Ex. 3 are denoted as $U = \{t_1^2, t_3, t_4\}$, $m' = \{p_5^2, p_6, p_7\}$.*

Analogously to occurrence sequences, a finite or infinite sequence of steps (or maximal-steps) $U_1U_2...U_k...$ is a *step sequence* (or a *maximal-step sequence*) enabled at $m_0$, denoted $m_0[U_1...U_k...\rangle$, if there are intermediate markings $m_1...m_k...$ such that: $m_0[U_1\rangle m_1...[U_k\rangle m_k...$.

Whereas in standard Petri nets semantics transitions occur asynchronously, maximal-step semantics require that in each moment as many transitions as possible occur. However, when the system allows for asymmetric confusion, using maximal-step semantics may hide information about 1-live transitions. For an example, consider the net on the left in Fig. 2.2. In the initial marking, the only enabled maximal step is $U = \{t_1, t_2\}$. After the occurrence of $U$, the new marking is $m$ defined as $m(p_5) = m(p_4) = 1$, $m(p_1) = m(p_2) = m(p_3) = m(p_6) = 0$. The marking $m$ does not enable $t_3$, that therefore is never enabled when considering maximal-step semantics. However, if instead of maximal steps we consider any possible behaviour of the net, we find a marking in which $t_3$ is enabled, namely the marking reached after the occurrence of $t_2$. In [80], the authors show that if there is no asymmetric confusion, the two semantics are equivalent, namely a transition can fire $n$ times in the initial net if, and only if, it can fire $n$ times in the net with maximal-step semantics.

The following subsection describes a subclass of P/T systems that rules out confusion, and therefore in which some of its properties can be studied with maximal-step semantics.

### 2.1.1 Equal-conflict systems

Equal-conflict Petri nets were introduced in [120] as a generalization of *free-choice* nets [46]. Some of their properties are collected in [23, 78, 125, 126].

**Definition 1.** *A Petri net $\Sigma = (P, T, F, W)$ is* free-choice *if the following properties are satisfied.*

1. *For each pair of transitions $t_1, t_2 \in T$, if ${}^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$, then ${}^\bullet t_1 = {}^\bullet t_2$.*
2. *For each $(x, y) \in F$, $W(x, y) = 1$.*

In the first definition [73], there was an additional condition imposing that, if $t_1$ and $t_2$ are in conflict, then $|{}^\bullet t_1| = |{}^\bullet t_2| = 1$. Subsequent works showed that this condition can be relaxed without losing any interesting property.

Equal-conflict nets generalise the second condition by allowing weights on arcs.

**Definition 2.** *A net $\Sigma = (P, T, F, W)$ is* equal-conflict *if, and only if, the first point of Def. 1 holds, and for each pair of transitions $t_1, t_2 \in T$, for each place $p \in {}^\bullet t_1 \cap {}^\bullet t_2$, $W(p, t_1) = W(p, t_2)$.*

**Example 4.** *Fig. 2.1 represents an example of equal-conflict system. The system in Fig. 2.4 is not equal-conflict since ${}^\bullet t_3 = \{p_2\} \neq {}^\bullet t_2 = \{p_2, p_3\}$, and ${}^\bullet t_2 \cap {}^\bullet t_3 \neq \emptyset$.*

The structure of equal-conflict nets (resp. free-choice) nets guarantees that when we consider an equal-conflict (resp. free-choice) system $\Sigma = (P, T, F, W, m_{in})$, for each marking $m \in [m_{in}\rangle$, if a transition $t \in T$ is enabled

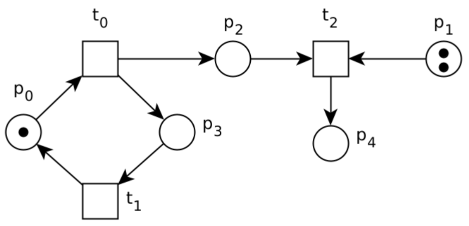


Figure 2.3: An unbounded system.

in $m$, also all the transitions in conflict with $t$ are enabled in $m$. This excludes any possible situation of confusion and can lead to more efficient algorithms. The structures of these nets is often sufficient to model business processes, therefore they are of relevant use in some application domains such as process mining [123, 127]. As a drawback, the absence of confusion prevents them to model mutual exclusion and resource sharing situations [118].

### 2.1.2 Bounded and unbounded systems

In general, a P/T system $\Sigma = (P, T, F, W, m_{in})$ may have an infinite number of global states, namely $[m_{in}\rangle$ is infinite, even if $P \cup T$ is finite. If this happens, we say that $\Sigma$ is unbounded. For an example, consider the system in Fig. 2.3. Every time that transition $t_0$ fires, it puts a token in place $p_2$; transition $t_2$ takes a token from $p_2$, but it can fire only twice, since it needs also a token from its precondition $p_1$. Since $t_0$ can fire infinitely often, place $p_2$ can accumulate an infinite number of tokens and the net is unbounded.

Unbounded systems provide a finite structure to study systems with infinite states, and therefore they are a very powerful tool. However, many problems are hard to solve on unbounded nets, and for many practical applications, models with a finite number of states are sufficient and easier to analyse [50].

A system $\Sigma = (P, T, F, W, m_{in})$ such that $|[m_{in}\rangle| < \infty$ is called *bounded*. Let $k \in \mathbb{N}$; $\Sigma$ is $k$-bounded (or *$k$-safe*) if, for each $p \in P$, for each $m \in [m_{in}\rangle$, $m(p) \leq k$.

Fig. 2.1 represents a 4-bounded system: if both $t_2$ and $t_3$ occur twice from the initial marking, the resulting marking has 4 tokens in $p_6$ and none in the other places. No other reachable marking assigns more than 4 token to a place.

#### 1-safe systems

When considering bounded systems, many studies focus on the analysis and application of 1-safe systems [41, 82, 37, 28, 131], namely systems such that

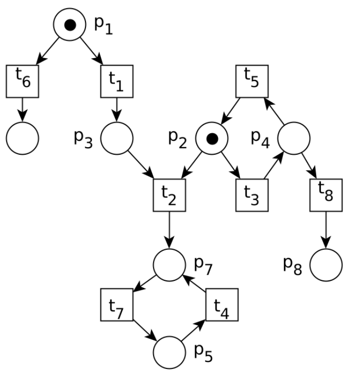


Figure 2.4: A 1-safe Petri net.

in each place of each reachable marking, either there is no token, or there is only one token. In a 1-safe and 1-live system $\Sigma = (P, T, F, W, m_{in})$, $W$ does not add any information to $F$: if $(x, y) \in F$, then $W(x, y) = 1$, otherwise $W(x, y) = 0$. Hence, we can denote a 1-safe system as $\Sigma = (P, T, F, m_{in})$, without explicitly referring to $W$.

In such systems, places can be interpreted as logical conditions: given a reachable marking $m$, a condition $p \in P$ is true if $m(p) = 1$, and false if $m(p) = 0$. Since there are no other possible values that a marking can take, we can (and will) interpret a marking as the set of conditions that are true in it.

**Example 5.** *The system in Fig. 2.4 is* 1*-safe, and its initial marking is the set* $\{p_1, p_2\}$.

## 2.2 The unfolding

For analysing a P/T system, it is often useful to consider formal models derived from it and explicitly describing its behaviour, such as the *unfolding* [49]. The unfolding is an acyclic Petri net recording all the possible executions of the system. Its elements are partially ordered, according to a causality relation. If the system is cyclic, some of its executions may be infinite, hence, in general the unfolding is an infinite object. To deal with it algorithmically, many researchers defined finite prefixes, sufficient to verify some properties of the system [52, 85, 27, 38]. In many cases, the size of these prefixes is smaller than the size of the set of reachable markings.

In order to define the unfolding of a P/T system, we need to introduce two technical relations and some preliminary definitions. The $\prec$ relation on the elements of a net $N$ is the transitive closure of $F$ and $\preceq$ is the reflexive closure of $\prec$. Let $x, y \in P \cup T$, $x \natural y$, iff there exist $t_1, t_2 \in T : t_1 \neq t_2,\ t_1 \preceq x,\ t_2 \preceq y$ and there exists $p \in {}^\bullet t_1 \cap {}^\bullet t_2$.

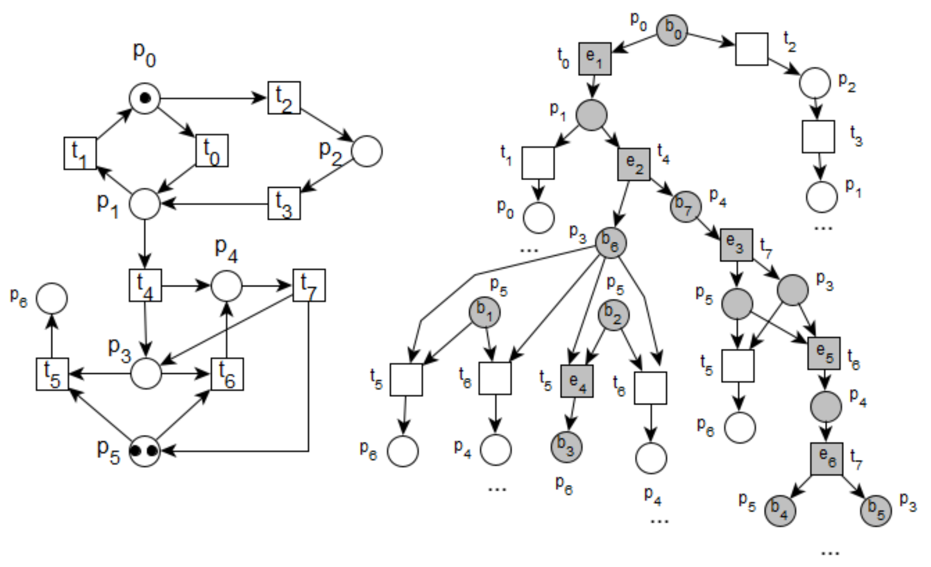


Figure 2.5: An equal-conflict P/T net (on the left) and a prefix of its unfolding (on the right).

A net $N = (B, E, F)$, possibly infinite, is an *occurrence net* if the following restrictions hold:

1. $\forall x \in B \cup E : \neg(x \prec x)$.
2. $\forall x \in B \cup E : \neg(x \natural x)$.
3. $\forall e \in E : \{x \in B \cup E \mid x \preceq e\}$ is finite.
4. $\forall b \in B : |{}^\bullet b| \leq 1$.

An occurrence net can represent different histories of a net. The first point guarantees the acyclicity of the net, the second and fourth points guarantee that each time that there is a conflict, the branches of the net following that conflict will never be reunited. The third point ensures the existence of minimal element with respect to the $\preceq$ relation.

In an occurrence net, the elements of $B$ are called *conditions* and the elements of $E$ are called *events*; the transitive and reflexive closure of $F$, $\preceq$, forms a *partial order*. The set of minimal elements of an occurrence net $N$ with respect to $\preceq$ will be denoted by $\min(N)$. Since we only consider T-restricted nets the elements of $\min(N)$ are conditions.

**Example 6.** *The net on the right in Fig. 2.5 is a prefix of an occurrence net. In this net,* $\min(N) = \{b_0, b_1, b_2\}$.

A *configuration* of an occurrence net $N = (B, E, F)$ is a, possibly infinite, set of events $C \subseteq E$ which is causally closed (for every $e \in C$, $e' \preceq e \Rightarrow e' \in C$) and free of conflicts ($\forall e_1, e_2 \in C$, $\neg(e_1 \natural e_2)$). $C$ is *maximal* if it is maximal with respect to set inclusion.

In the net in Fig. 2.5 the set of grey events is a maximal configuration.

A configuration $C$ is *local* if there is an event $e \in C$ such that $C = \{e' \in E \mid e' \leq e\}$, with $E$ set of events of the unfolding. Such a local configuration is denoted $\lfloor e \rfloor$. Any local configuration is a finite set, and this is due to condition 3. of the definition of occurrence nets.

On the elements of an occurrence net the relation of concurrency, **co**, is defined as follows: let $x, y \in B \cup E$, $x$ **co** $y$, if neither $(x \prec y)$ nor $(y \prec x)$ nor $(x \natural y)$.

A *B-cut* of $N$ is a maximal set of pairwise concurrent elements of $B$, and can be intuitively seen as a global state of the net in a certain moment. An *E-cut* of $N$ is a maximal set of pairwise concurrent elements of $E$, that corresponds to a maximal step on $N$. By analogy with net systems, we will sometimes say that an event $e$ of an occurrence net is enabled at a B-cut $\gamma$, denoted $\gamma[e\rangle$, if ${}^\bullet e \subseteq \gamma$. We will denote by $\gamma{+}e$ the B-cut $(\gamma \setminus {}^\bullet e) \cup e^\bullet$. A B-cut is a deadlock if no event is enabled at it. We will denote by $C(\gamma)$ the set of all events in the causal closure of a B-cut $\gamma$, i.e., $C(\gamma) = \{e \in E \mid e \prec b,\ b \in \gamma\}$. We say that an event $e \in E$ precedes a B-cut $\gamma$, and write $e < \gamma$, iff there is $y \in \gamma$ such that $e \prec y$. In this case, each element of $\gamma$ either follows $e$ or is concurrent with $e$ in the partial order induced by the occurrence net. Given an event $e$, we denote with $\gamma(\lfloor e \rfloor)$ the minimal B-cut in which $e$ is enabled, namely, if $\downarrow e = \{e' \in E : e' \leq e\}$, then $\gamma(\lfloor e \rfloor) = ((\downarrow e^\bullet \setminus {}^\bullet \downarrow e) \setminus \{e^\bullet\}) \cup \{{}^\bullet e\}$.

Let $\Gamma$ be the set of B-cuts of $N$. A partial order on $\Gamma$ can be defined as follows: let $\gamma_1, \gamma_2$ be two B-cuts. We say $\gamma_1 \leq \gamma_2$ iff

1. $\forall y \in \gamma_2\ \exists x \in \gamma_1 : x \preceq y$
2. $\forall x \in \gamma_1\ \exists y \in \gamma_2 : x \preceq y$

In words, $\gamma_1 \leq \gamma_2$ if any condition in the second B-cut is or follows a condition of the first B-cut and any condition in the first B-cut is or comes before a condition of the second B-cut. $\gamma_1 < \gamma_2$ if the conditions above hold and $\exists x \in \gamma_1, \exists y \in \gamma_2 : x \prec y$.

A sequence of B-cuts, $\delta = \gamma_0\gamma_1 \ldots \gamma_i \ldots$ is *increasing* if $\gamma_i < \gamma_{i+1}$ for all $i \geq 0$. A cut $\gamma$ is *compatible* with an increasing sequence of B-cuts $\delta$ iff either there are two cuts $\gamma_i, \gamma_{i+1} \in \delta$ such that $\gamma_i \leq \gamma \leq \gamma_{i+1}$ or $\gamma < \gamma_0$.

Given an increasing sequence of B-cuts $\delta$, we define a *refinement* $\delta'$ of $\delta$ as an increasing sequence of B-cuts such that for each $\gamma \in \delta$, $\gamma$ is also in $\delta'$. A *maximal* refinement $\delta'$ is an increasing sequence of B-cuts such that there is no $\gamma \notin \delta'$ compatible with $\delta'$.

**Example 7.** *In the occurrence net in Fig. 2.5, the set* $\gamma = \{b_3, b_4, b_5\}$ *is a B-cut, the set* $\{e_4, e_5\}$ *is an E-cut. The set* $\gamma' = \{b_1, b_2, b_6, b_7\}$ *is also a*

*B-cut, $C(\gamma') = \{e_1, e_2\}$, $\gamma'[e_4\rangle$, and $\gamma' + e_4 = \gamma'' = \{b_1, b_3, b_7\}$. The sequence $\delta = \gamma', \gamma$ is an increasing sequence of B-cuts. The cut $\gamma''$ is compatible with $\delta$.*

A *branching process* of a bounded 1-live PT system $\Sigma = (P, T, F, W, m_{in})$, whose underlying net is T-restricted, is a pair $(N, \lambda)$, where $N = (B, E, F)$ is an occurrence net, and $\lambda$ is a map from $B \cup E$ to $P \cup T$ such that:

1. $\lambda(B) \subseteq P$; $\lambda(E) \subseteq T$
2. $\forall e \in E$, $\forall p \in P$, $W(p, \lambda(e)) = |\lambda^{-1}(p) \cap {}^\bullet e|$ and $W(\lambda(e), p) = |\lambda^{-1}(p) \cap e^\bullet|$
3. $\forall p \in P$ $m_{in}(p) = |\lambda^{-1}(p) \cap \min(N)|$
4. $\forall x, y \in E$, if ${}^\bullet x = {}^\bullet y$ and $\lambda(x) = \lambda(y)$, then $x = y$

We extend the definition of $\lambda$ to the set of configurations of the branching process: for each configuration $C$, $\lambda(C)$ is the multiset of the transitions whose occurrences are recorded in $C$ and is called the *footprint* of $C$; formally $\lambda(C) = \sum_{e_i \in C} \lambda(e_i)$. We denote the number of occurrences of a transition $t$ in the footprint as $\lambda(C)(t)$. If a transition $t$ belongs to the support set of $\lambda(C)$, i.e.: at least an occurrence of $t$ is in $C$, $\lambda(C)(t) \geq 1$, then we use the notation $t \in \lambda(C)$. If $C$ is infinite, it records the infinite occurrence of some transitions, and the multiset $\lambda(C)$ is such that the multiplicity of those transitions is infinite, whereas its support set is obviously finite, being a subset of $T$.

Let $t \in T$ be any transition of a P/T system, we denote with $E_t = \{e \in E : \lambda(e) = t\}$ the set of occurrences of $t$ on the branching process.

A branching process $(N_1, \lambda_1)$ is a *prefix* of a branching process $(N_2, \lambda_2)$ if $N_1$ is a subnet of $N_2$ containing all minimal elements ($\min(N_2)$) and such that: if $e \in E_1$ and $(b, e) \in F_2$ or $(e, b) \in F_2$ then $b \in B_1$; if $b \in B_1$ and $(e, b) \in F_2$ then $e \in E_1$; and $\lambda_1$ is the restriction of $\lambda_2$ to $B_1 \cup E_1$.

Any finite P/T system $\Sigma = (P, T, F, W, m_{in})$ has a unique branching process which is maximal with respect to the prefix relation. This maximal branching process, called the *unfolding* of $\Sigma$, will be denoted by $\textsc{unf}(\Sigma) = (B, E, F, \lambda)$, where $\lambda$ is the map from $(B, E, F)$ to $(P, T, F)$.

A *run* records a possible non sequential behaviour of the system, it is a branching process, whose occurrence net is free of conflicts, i.e., its set of events is a configuration. A run is *maximal* if the corresponding configuration is maximal. Let $\rho$ be a run, its footprint $\lambda(\rho)$ is equal to the footprint of its set of events.

**Example 8.** *The net in Fig. 2.5 on the right is a prefix of the unfolding $\textsc{unf}(\Sigma)$ of the P/T system $\Sigma$ in the same figure on the left. For each element $x$ in $\textsc{unf}(\Sigma)$ represented in the prefix, the outside label is the name of $\lambda(x)$*

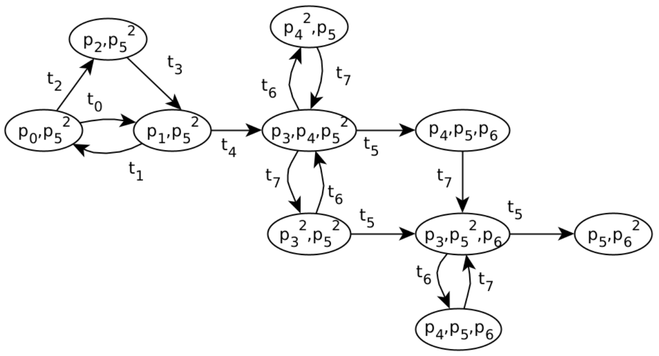


Figure 2.6: Sequential marking graph of the system on the left in Fig. 2.5.

*in $\Sigma$. Since the same label can occur multiple times, some elements have also a second label inside the node, identifying their occurrence on the unfolding. Let $C$ be the configuration formed by the events coloured in grey in the figure. Its footprint is $\lambda(C) = \{t_0, t_4, t_5, t_7^2\}$, where $t_7^2$ denotes that $\lambda(C)(t_7) = 2$.*

## 2.3 The marking graph

The *marking graph* is an initialized labelled transition system (LTS) used to analyse the behaviours of a P/T system. The formal definition of LTS follows.

**Definition 3.** *An* LTS *is a quadruple* TS $= (Q, U, A, q_0)$, *where:*

- *$Q$ is a set of states.*
- *$U$ is a set of labels.*
- *$A \subseteq Q \times U \times Q$ is a set of arcs or transitions labelled with elements of $U$.*
- *$q_0$ is an initial state*

Transitions in LTS and transitions in Petri nets are two different notions, for this reason, if the context can create ambiguity, the term 'arcs' will be preferred when referring to LTS.

In a marking graph, the set of states coincides with the set of reachable markings, and the arcs are labelled as the transitions in the net.

Formally, let $\Sigma = (P, T, F, W, m_{in})$ be a P/T system. The *sequential marking graph* (or simply marking graph) of $\Sigma$ is an LTS MG$(\Sigma) = ([m_{in}\rangle, T, A, m_{in})$, such that for each pair of markings $m_1, m_2 \in [m_{in}\rangle$, $(m_1, t, m_2) \in A$ if, and only if, $m_1[t\rangle m_2$.

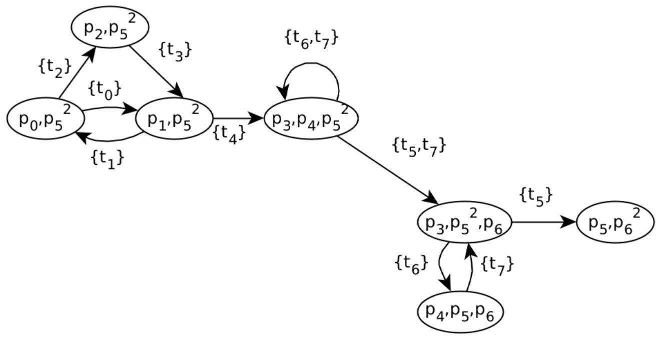


Figure 2.7: Marking graph of maximal steps of the system in Fig. 2.5.

**Example 9.** *Fig. 2.6 represents the marking graph of the P/T system in Fig. 2.5. Each state is labelled with a reachable marking, and each arc with the name of a transition of the system. The initial state on the marking graph is labelled* $\{p_0, p_5^2\}$*, where the notation* $p_5^2$ *means that, in the initial marking* $m_{in}$*,* $m_{in}(p_5) = 2$*, and analogously for the other nodes where an exponent appears.*

The marking graph provides a description of the system based on the sequences of transitions that can occur from the initial marking. If $\Sigma$ is bounded, MG($\Sigma$) is finite. However, its size can be exponentially larger than the size of $\Sigma$. This happens because in each marking, concurrent transitions can fire in any order, and since the marking graph uses interleaving semantics, it needs to record all of them. For this reason the marking graph is said to suffer of the *state explosion* problem.

### 2.3.1 Maximal-step semantics in the marking graph

The maximal-step semantics discussed in Sec. 2.1 for P/T systems can also be used in the construction of the marking graph. Let $\Sigma = (P, T, F, W, m_{in})$, its *marking graph of maximal steps*, denoted mmg($\Sigma$) or, if $\Sigma$ is clear from the context, just mmg, is an LTS where the arcs are labelled with maximal steps of transitions, and the states are the markings that can be reached on the system through the occurrence of maximal steps from the initial marking. The marking graph of maximal steps is at most large as the sequential marking graph, and it may be smaller in P/T systems with concurrent transitions, since the maximal steps may hide some reachable markings. However, as discussed in Sec. 2.1, in case of asymmetric confusion in the P/T system, analysing the system with maximal-step semantics may provoke loss of information also with respect to the set of transitions that can be enabled in a system.

**Example 10.** *The* LTS *in Fig. 2.7 represents the marking graph of maximal* *steps of the system in Fig. 2.5. Compared to the sequential marking graph in Fig. 2.6, this* LTS *has two states less, that can never be reached by considering only maximal steps. However, since the P/T system is equal-conflict, all the transitions that can be enabled in the P/T system are represented in the marking graph of maximal steps.*

### 2.3.2 Tree-unfolding of the marking graph of maximal steps

Let TS $= (Q, U, A, q_0)$ be an LTS on alphabet $T$, namely in which $U$ is the set of labels of the transitions in $A$, and it is the set of multisets of the elements in $T$. A path on mmg is a possibly infinite sequence $\pi = q_0 u_0 q_1 u_1 ... q_n u_n ...$ such that for each $i \in \mathbb{N}$, $q_i \in Q$, $u_n \in U$, and $(q_i, u_i, q_{i+1}) \in A$. Given a path $\pi$, we can define a *footprint* of $\pi$, analogous to the footprint of a configuration defined in Sec. 2.2. With an abuse of notation, we define $\lambda(\pi)$ the footprint of $\pi$, namely the multiset union of all the labels occurring in $\pi$.

In order to represent explicitly all the paths of an LTS, we can unfold it in a tree. A first version of this tree was introduced in [7] for representing the marking graph of maximal steps in 1-safe free-choice systems, and then adapted to the class of bounded equal-conflict PT systems in [6]. In [1] the tree is generalised to any labelled transition system, therefore also including those representing the maximal-step marking graph of bounded P/T systems. Here we consider this last formal definition.

**Definition 4.** *Let* TS $= (Q, U, A, q_0)$ *be a labelled transition system on alphabet $T$. The* unfolding-tree *of* TS, *denoted as* TS*-tree, is a tree defined as follows.*

- *Each node of the tree is labelled with a state in $Q$.*
- *The root of the tree is labelled with the initial state $q_0$.*
- *From each node labelled with state $q$ in the tree, for each transition $(q, u, q')$ in* TS, *there is exactly an arc in the tree, labelled with $u$, and leading to a node labelled with $q'$.*

Analogously to the *path* in a TS, a path in a TS-tree is a sequence $\pi = q_1 u_1 q_2 u_2 \cdots$, such that, for each $i$, there is an arc from $q_i$ to $q_{i+1}$, labelled with $u_i$. A path is *initial* if it starts in the root of the tree. A path can be either finite or infinite. Let $v$, $r$ be two nodes on the same path $\pi$ in a TS-tree, $v < r$ iff $v$ is closer to the root than $r$. The footprint $\lambda(\pi)$, is the (multiset) union of all the labels occurring in $\pi$.

By construction, for each maximal path $\pi$ in a TS, there is a maximal path $\pi'$ on the TS-tree with the same footprint and vice versa.

We define an inclusion relation between footprints: let $\pi_1$ and $\pi_2$ be paths on a TS-tree; then $\lambda(\pi_1) \subseteq \lambda(\pi_2)$ iff $\forall t \in T$, $\lambda(\pi_1)(t) \leq \lambda(\pi_2)(t)$, i.e.,

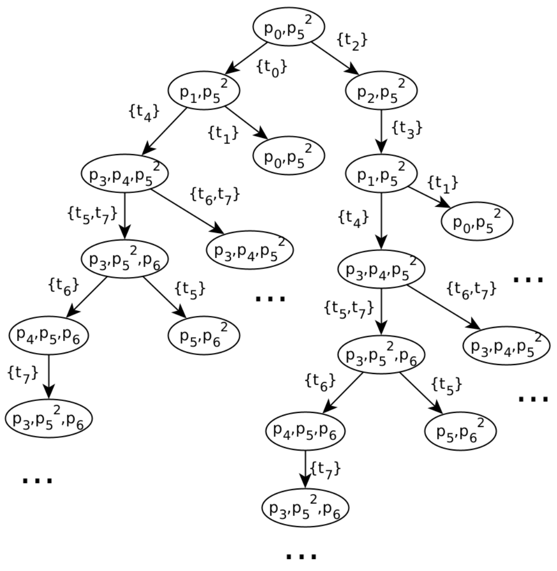


Figure 2.8: The tree-unfolding of the mmg in Fig. 2.7.

for each element $t$, the number of occurrences in the footprint of $\pi_1$ is less than or equal to that in the footprint of $\pi_2$.

To simplify the notation, we will write $t \in \lambda(\pi)$ when $\lambda(\pi)(t) \geq 1$, namely when $t$ occurs in some step of path $\pi$.

**Example 11.** *Fig. 2.8 represents a prefix of the tree of the* LTS *and* mmg *in Fig. 2.7. A non-maximal initial path on the tree is for example*

$$\pi = \{p_0 p_5^2\}\{t_0\}\{p_1, p_5^2\}\{t_1\}\{p_0, p_5^2\}.$$

*The footprint of $\pi$ is $\lambda(\pi) = \{t_0, t_1\}$.*

The following lemma states a property of TS-trees that will be useful for the results presented in Chap. 3.

**Lemma 1.** *Let* $\text{TS} = (Q, U, A, q_0)$ *be an* LTS*, and $v_i$ and $v_k$ be two nodes of the* TS*-tree labelled with the same elements of $Q$. The subtree with $v_i$ as root and the one with $v_k$ as root are isomorphic.*

*Proof.* Since $v_i$ and $v_k$ are associated to the same element in $Q$, the arcs leaving from $v_i$ and from $v_k$ have the same labels. Then, by construction the children of $v_i$ are equivalent to the children of $v_k$, and the same reasoning can be applied to each of them. □

Let $\pi$ be a path on a TS-tree, and $v$ be a node on $\pi$; we denote with $v{\downarrow}\pi$ the set of nodes $r \in \pi$ such that $v \geq r$, and symmetrically, with $v \uparrow \pi$ the set of nodes $r \in \pi$ such that $v \leq r$. On the maximal paths of the msc-tree, we can define a *peeling* operation as follows [7, 72].

**Definition 5.** *Let* TS $= (Q, U, A, q_0)$ *be an* LTS*,* $\pi = v_1 U_1 v_2 U_2 \cdots$ *be a maximal path on the* TS*-tree,* $v_i$ *and* $v_k$ *be two nodes in* $\pi$ *associated to the same element of* $Q$ *and such that* $i < k$*. Let* $\pi_{i,k}$ *be the subpath between* $v_{i+1}$ *and* $v_k$*. The* peeling *of* $\pi$ *with respect to* $\pi_{i,k}$ *is the path* $\pi' = \text{peel}(\pi, \pi_{i,k})$ *such that* $v_i \uparrow \pi = v_i \uparrow \pi'$*,* $\pi_{n,k}$ *is deleted from* $\pi$*, and* $v_k \downarrow \pi$ *is equivalent to* $v_i \downarrow \pi'$*.*

In words, peeling $\pi$ means to consider the execution in which the cycle $\pi_{i,k}$ has not been executed. The path $\pi'$ constructed in this way is also maximal in the TS-tree.

**Some Notation for the tree-unfolding**

In the following we will introduce some notations which will be used throughout Chap. 3. In all the following items, $A$ will denote a tree-unfolding, or any sub-tree of it.

- $\Pi_A$ is the set of all the maximal paths of $A$.
- Given a node $v$ in a tree $A$, we denote $A_v$ the largest sub-tree of $A$ whose root is $v$.
- Given a transition $t$, we write
$$\Pi_A^t := \{\pi \in \Pi_A \mid t \in \lambda(\pi)\}.$$
- Given a finite path $\pi$ of $A$, let leaf$(\pi)$ denote the single leaf in this path.

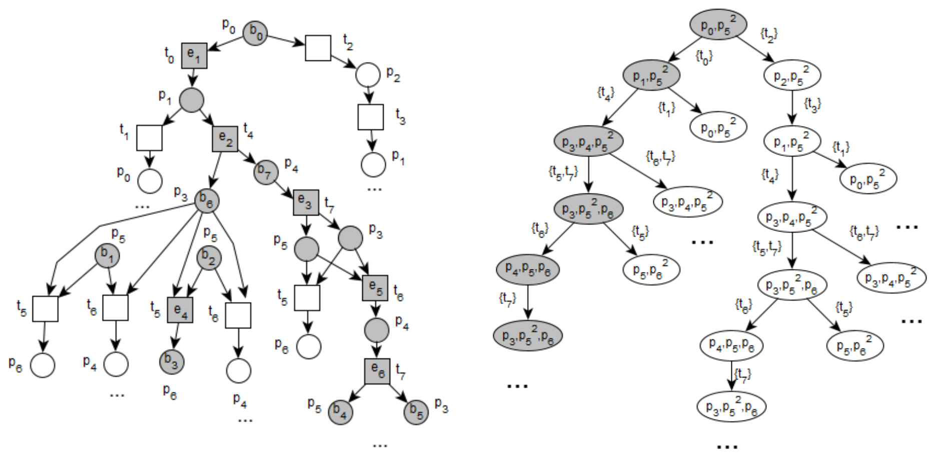
## 2.4 Unfoldings and marking graphs of maximal steps in equal-conflict P/T systems

The unfolding presented in Sec. 2.2, and the marking graph presented in Sec. 2.3 are the two most common tools for the analysis of P/T systems.

The use of the unfolding is sometimes convenient, since prefixes on which some properties can be verified are often significantly smaller than the marking graph [52]. In addition, the unfolding has an explicit representation of concurrency and partial order between elements, whereas the marking graph represents the system using sequential semantics.

On the other hand, the large body of works for finding properties on LTS and the explicit representation of the global states make the marking graph a practical tool to verify properties of P/T systems.

When we focus on equal-conflict P/T systems and maximal-step semantics, there is a strong relation between the unfolding and the marking graph of maximal steps.



$p_0$ $b_0$ $t_0$ $e_1$ $t_2$ $p_2$ $p_1$ $t_1$ $e_2$ $t_4$ $t_3$ $b_7$ $p_4$ $p_0$ $p_1$ $p_3$ $b_6$ $e_3$ $t_7$ $p_5$ $b_1$ $p_5$ $b_2$ $p_5$ $p_3$ $t_5$ $t_6$ $t_5$ $e_4$ $t_6$ $t_5$ $e_5$ $t_6$ $p_4$ $p_6$ $p_4$ $b_3$ $p_6$ $p_4$ $e_6$ $t_7$ $p_5$ $b_4$ $b_5$ $p_3$ …
$p_0,p_5^2$ $\{t_0\}$ $\{t_2\}$ $p_1,p_5^2$ $p_2,p_5^2$ $\{t_4\}$ $\{t_1\}$ $\{t_3\}$ $p_0,p_5^2$ $p_3,p_4,p_5^2$ $p_1,p_5^2$ $\{t_6,t_7\}$ $\{t_5,t_7\}$ $\{t_1\}$ $p_0,p_5^2$ $\{t_4\}$ $p_3,p_5^2,p_6$ $p_3,p_4,p_5^2$ $p_3,p_4,p_5^2$ $\{t_6\}$ $\{t_5\}$ $\{t_5,t_7\}$ $\{t_6,t_7\}$ $p_4,p_5,p_6$ $p_5,p_6^2$ $p_3,p_5^2,p_6$ $p_3,p_4,p_5^2$ $\{t_7\}$ $\{t_6\}$ $\{t_5\}$ $p_3,p_5^2,p_6$ $p_4,p_5,p_6$ $p_5,p_6^2$ $\{t_7\}$ $p_3,p_5^2,p_6$ …

Figure 2.9: A prefix of the unfolding of the P/T system in Fig. 2.5, and a prefix of tree-unfolding of its marking graph.

In [25], the authors studied the relationships between various classes of non-sequential processes (runs of an unfolding), occurrence sequences and step sequences. They showed that, in the special case of countable, T-restricted P/T systems of finite synchronisation (i.e., where any transition has a finite set of pre- and post-places) with finite initial marking, and then also in the case of the bounded equal-conflict P/T systems, the distinction between occurrence sequences and step sequences disappears. Moreover, for the same class of P/T systems, they showed that it is possible to consider equivalence relations both on the set of occurrence sequences and on the set of processes such that there is a bijection between the induced equivalence classes.

In the case of occurrence sequences, the equivalence relation abstracts w.r.t. the total order arbitrarily chosen when transitions are concurrent; in the case of processes, the equivalence abstracts from the distinction among several tokens on the same place.

From the construction of the equivalence classes both of occurrence sequences and of processes, it is possible to observe that the elements in each class have the same footprints, i.e., the same multiset of transitions which can be observed through the corresponding behaviour, and that elements of an equivalence class in bijective correspondence with a class of the other sort have the same footprint too.

**Example 12.** *Fig. 2.9 recalls the unfolding of the equal-conflict system in Fig. 2.5, and the tree-unfolding of the marking graph of maximal steps of the same P/T system.*

*The run coloured in grey in the unfolding has the same footprint of the path coloured in grey in the tree-unfolding. We can verify that we can find* *a path for all the maximal runs of the unfolding.*

*Given a path on the tree, we may find more that a maximal run of the unfolding, for example, to the grey path on the tree, we can associate both the grey run on the unfolding, and the run without $e_4$, but with the other occurrence of $t_5$. This is not bothering, since the two runs are associated to indistinguishable executions on the P/T system.*

## 2.5 Synthesis with region theory

Let TS $= (Q, U, A, q_0)$ be an initialized labelled transition system. The synthesis problem consists in finding a P/T system $\Sigma$, if one exists such that TS is the marking graph of $\Sigma$. A method to address this problem is given by *region theory.*

Regions were introduced in [48], and in [100] they were extended to the synthesis of P/T systems. Intuitively, a region corresponds to a potential place of the P/T system. An LTS is synthesizable into a P/T system if its set of regions satisfies some constraints. In this section, we will first consider region theory in general P/T systems, and then focus on the special case of the synthesis of a 1-safe system. For a detailed overview on region theory and the synthesis problem, refer to [11].

We assume that in all the LTS under consideration all the states are reachable from the initial state, and that every label appears in at least one arc. The marking graph of a 1-live P/T system fulfills these properties.

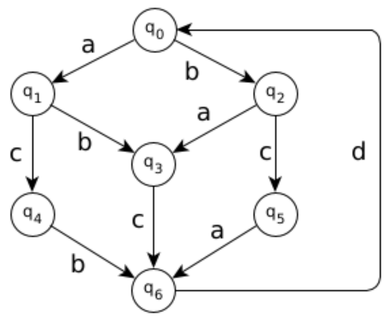
**Definition 6.** *Let* TS $= (Q, U, A, q_0)$ *be an* LTS*. A* region *of* TS *is a pair of maps $(r, w_r)$, with $r : Q \to \mathbb{N}$, and $w_r : U \to \mathbb{N} \times \mathbb{N}$, such that, for each $(q, t, q') \in A$, if $w_r(t) = (x, y)$, the following conditions hold.*

1. $r(q) \geq x$;
2. $r(q') = r(q) - x + y$

If we think of $r$ as a map giving the number of tokens in a potential place in each state, then Def. 6 states that every firing of $t$ has the same effect on the region $r$. For each region, the function $w_r$ does not depend on the choice of the arc, but only on its label. For each label $t \in U$, the value $w_r(t)$ expresses the change in the value of $r$ produced by any arc labelled by $t$.

**Example 13.** *Consider the labelled transition system in Fig. 2.10, and the two maps $r_1$ and $r_2$ tabulated on the right of the picture. Let $w_{r_1}(a) = (0, 1)$, $w_{r_1}(b) = (0, 1)$, and $w_{r_1}(c) = (1, 0)$; and let $w_{r_2}(a) = (1, 0)$, $w_{r_2}(b) = (0, 0)$ $w_{r_2}(c) = (0, 0)$. The pairs $(r_1, w_{r_1})$ and $(r_2, w_{r_2})$ are regions of* TS.

In order to have a solution for the synthesis problem, the LTS needs to satisfy some separation requirements.



| | $q_0$ | $q_1$ | $q_2$ | $q_3$ | $q_4$ | $q_5$ | $q_6$ |
|---|---|---|---|---|---|---|---|
| $r_1$ | 0 | 1 | 1 | 2 | 0 | 0 | 1 |
| $r_2$ | 1 | 0 | 1 | 0 | 0 | 1 | 0 |

Figure 2.10: An LTS and two of its regions.

Let $q_1, q_2 \in Q$ be two distinct states of TS $= (Q, U, A, q_0)$. A region $(r, w_r)$ *separates* $q_1$ and $q_2$ if $r(q_1) \neq r(q_2)$. In this case, $r$ solves the *state separation problem* $(q_1, q_2)$. Let $t$ be a label in $U$, and $q$ a state such that $t$ is not enabled in $q$; a region $(r, w_r)$ separates $t$ from $q$ if in $w_r(t) = (x, y)$, $x > 0$ holds, and $r(q) < x$.

Intuitively, we can think of $r$ as a place of a P/T system, with $r(q)$ the number of tokens in $r$ in state $q$. The occurrence of $t$ in $r$ takes $x$ token from $r$, hence $r$ is an input place of $t$; then $r(q) < x$ means that in state $q$, $r$ does not contain enough tokens to allow for the firing of $t$. We will then say that $r$ solves the state-event separation problem $(q, t)$.

A labelled transition system TS is *separated* if, for each pair of distinct states, there is a region solving the corresponding separation problem, and, for each state $q$ and for each label $t$ not enabled in $q$, there is a region solving the corresponding separation problem.

If the transition system is not separated, we cannot synthesize a P/T system such that its marking graph is isomorphic to the transition system.

We will say that a set $D$ of regions *separates* TS $= (Q, U, A, q_0)$, if it contains enough regions to solve all separation problems. If this is the case, we can define a P/T system as follows. Let $R = \{r : (r, w_r) \in D\}$ be the set of functions describing the states in the regions, we identify this set with the set of places in the P/T system. In the P/T system $\Sigma = (R, U, F, W, m_{in})$, the flow relation and the weights between elements are defined in this way: for each region $r$ in $R$, for each $t \in U$, let $w_r(t) = (x, y)$; then $W(r, t) = x$ and $W(t, r) = y$. For all $r$ in $R$, $m_{in}(r) = r(q_0)$.

If $R$ separates TS, then the marking graph of $\Sigma$ is isomorphic to TS (see [11] for a proof).

For each separation problem, there is an algorithm, polynomial in the size of TS, which decides if there is a region solving it, and, if so, effectively finds a separating region. For details on the algorithm, again see [11].

**Example 14.** *Consider again the transition system* TS *shown in Fig. 2.10. The rows of the table on the left of Fig. 2.11 correspond to the function*

|  | $q_0$ | $q_1$ | $q_2$ | $q_3$ | $q_4$ | $q_5$ | $q_6$ |
|---|---|---|---|---|---|---|---|
| $r_1$ | 0 | 1 | 1 | 2 | 0 | 0 | 1 |
| $r_2$ | 1 | 0 | 1 | 0 | 0 | 1 | 0 |
| $r_3$ | 1 | 1 | 0 | 0 | 1 | 0 | 0 |
| $r_4$ | 0 | 1 | 0 | 1 | 1 | 0 | 1 |
| $r_5$ | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| $r_6$ | 0 | 0 | 0 | 0 | 1 | 1 | 1 |
| $r_7$ | 0 | 0 | 1 | 1 | 0 | 1 | 1 |

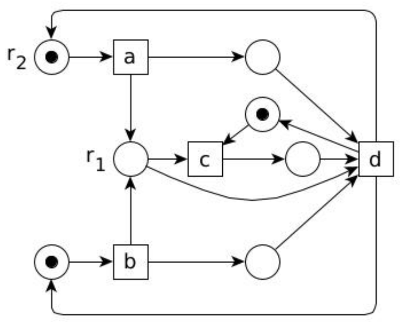


Figure 2.11: A P/T system synthsizing the LTS in Fig. reff:ptlts.

*associated to places in the net, which are part of a separating set of regions for* TS*. The P/T system on the right is the corresponding is constructed from them as described above.*

In order to clarify the correspondence between regions of a transition system and places of a net system, consider a bounded P/T system $\Sigma = (P, T, F, W, m_{in})$ and its marking graph $\text{MG}(\Sigma) = ([m_{in}\rangle, T, A, m_{in})$. Choose a place $p$ and define a map $f_p : [m_{in}\rangle \to \mathbb{N}$ as follows:

$$\forall m \in M : f_p(m) = m(p).$$

As a consequence of the firing rule of P/T nets, such a map satisfies a uniformity property: for each transition $t$, and for each edge $(m, t, m')$ in $A$, the difference $f_p(m) - f_p(m')$ is the same, and is equal to $W(p, t) - W(t, p)$. The marking graph $\text{MG}(\Sigma)$ is separated by the set of regions corresponding to the places of $\Sigma$.

**Remark 2.** *Let* $\text{TS} = (Q, U, A, q_0)$ *be an* LTS*; it may not exist any set of regions separating* TS *(see Ex. 15). However, we can always synthesize a labelled net by splitting transitions with the same label (in the worse case, element of A is associated to a different transition on the net). The problem of deciding how to split transitions by keeping their number as small as possible and preserving the behaviour described by* TS *has been tackled under different assumptions in [35, 114].*

**Example 15.** *Consider the* LTS *in Fig. 2.12; from the state* $q_1$*, transition c can occur followed by transition d, which restore the state* $q_1$*. In the P/T system, this means that the effect of c in a given marking is cancelled by the occurrence of d. However, this is contradicted by the structure of the* LTS *starting in* $q_3$*: the sequence cd can also occur, but it does not return to* $q_3$*. In order to obtain a P/T system with the behaviour described in the* LTS *we need to split some labels, for example by considering different transitions labelled with c in the P/T system.*

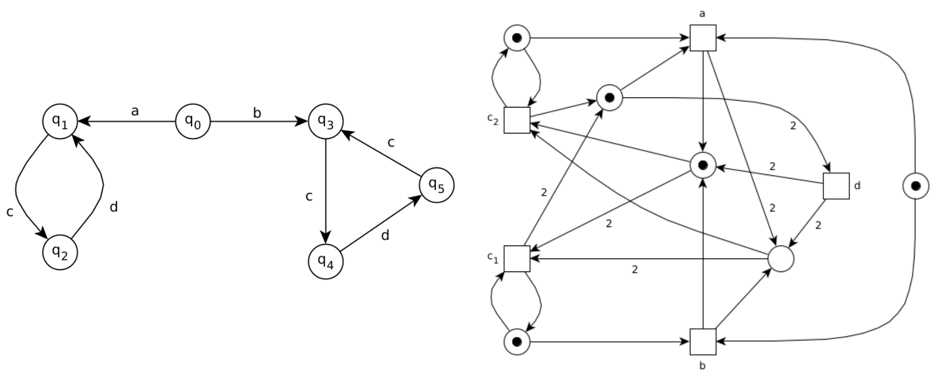


Figure 2.12: An LTS that can be synthesized only with a labelled net on the left, and a labelled P/T system synthesizing it on the right.

*The labelled P/T system on the right of the same figure is obtained through the synthesis of the* LTS *on the left, where the transitions labelled with c starting from $q_1$ has been considered as a different transition from the ones starting from $q_3$ and from $q_5$ (on the P/T system they are labelled with $c_1$ and $c_2$ respectively).*

### 2.5.1 Synthesis of 1-safe systems

In the special case in which we need to synthesize a 1-safe system, we can consider a region as a set of states in an LTS. In this section we show how regions are characterized when we see them as sets, and how the separation problems are formalized in this case. This notions could be equivalently expressed as a restriction of the general case, by imposing that for each region $(r, w_r)$, for each state $q$, $r(q) \in \{0,1\}$, and for each label $t$, $w_r \in \{0,1\} \times \{0,1\}$. However, considering regions as subset of states is more convenient for this case, and translation into this formalism is not immediate, therefore we explicitly discuss it in this section.

Let TS $= (Q, U, A, q_0)$ be a transition system from which we want to synthesize a 1-safe system.

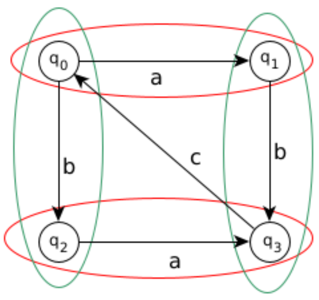
**Definition 7.** *A set $r \subseteq Q$ is a* region *if, and only if, for each label $t \in U$, one of the following conditions hold.*

1. *For each arc $(q,t,q') \in A$, $q \in r$ and $q' \notin r$.*

2. *For each arc $(q,t,q') \in A$, $q' \in r$ and $q \notin r$.*

3. *For each arc $(q,t,q') \in A$, $q \in r$ and $q' \in r$, or $q \notin r$ and $q' \notin r$.*

Let TS $= (Q, U, A, q_0)$ be an LTS. TS is synthesizable with a 1-safe net if there is a set of regions $D$ satisfying the following separation properties.



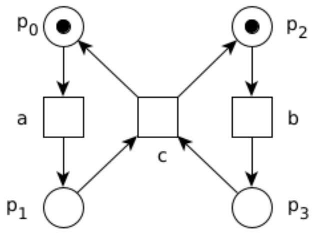


Figure 2.13: An LTS and the synthesised 1-safe system

- For each pair of states $q_1 \neq q_2$ in $Q$, there is a region $r \in D$ such that $q_1 \in r$, $q_2 \notin r$.
- For each state $q \in Q$, and each label $t \in U$ such that there is no arc $(q, t, q') \in A$, there is a region $r \in D$ such that $q \notin r$, and either $r$ satisfies condition (1) in Def. 7, or for each arc $(q_1, t, q_2) \in A$, $q_1 \in r$ and $q_2 \in r$.

If such a set $D$ exists, a 1-safe system $\Sigma = (D, U, F, m_{in})$ can be synthesized as follows. For each place $r \in D$, and for each transition $t \in U$, if condition 1 in Def. 7, $(r, t) \in F$; if condition 2 holds, then $(t, r) \in F$; if condition 3 holds, and in particular, for each arc $(q, t, q') \in A$, $q, q' \in r$, then $(r, t), (t, r) \in F$.

A region $r$ is *minimal* if there is no region $r'$ such that $r \subseteq r'$. In [19], the author proved that, if a set of regions separating an LTS exists, the set of minimal regions of the LTS is sufficient to solve all the separation problems.

**Example 16.** *Consider the transition system on the left of Fig. 2.13, with $q_0$ as initial state. The green and red circle correspond to four different regions on it. This regions are sufficient to solve all the state separation problems, and state-transition separation problems. The synthesis process with these four regions produces the 1-safe system on the right of the same figure. In particular, place $p_0$ is the region $\{q_0, q_2\}$: since it contains the initial state in the LTS, it belongs to the initial marking in the net; $p_0$ is a precondition of transition a, for both the arcs labelled with a, the region satisfies condition 1 of the Def. 7; it is a postcondition of transition c since it satisfies condition 3 of the definition, and has no relation with b, since $q_1$ and $q_3$ are outside the region.*

## Implicit places

In a 1-safe system $\Sigma = (P, T, F, m_{in})$, each marking is a set of places, therefore, also each state of the marking graph can be seen as a set of places.

We can associate to each place its *extension*, namely the set of reachable markings to which it belongs:

$$\forall p \in P \quad r(p) = \{m \in m_{in} \mid p \in m\}.$$

The extension of a place satisfies a few properties, and in particular it is a region of the 1-safe system.

We have already discussed in Sec. 2.5 that regions can be associated to places of the net. In general, in the marking graph $\text{MG}(\Sigma)$ of a 1-safe net $\Sigma$, not all the region also extensions of places of $\Sigma$. However, if $r$ is a region of $\text{MG}(\Sigma)$, then we can add a corresponding place to $\Sigma$ without changing its behaviour: the marking graph of the extended net system is isomorphic to the marking graph of $\Sigma$. The place $r \in P$ is connected to the transitions of $\Sigma$ according to the synthesis rules presented in Sec. 2.5.1.

**Definition 8.** *Let $\Sigma = (P, T, F, m_{in})$ be a 1-safe system, and $\text{MG}(\Sigma)$ its marking graph. An* implicit place *is a region of $\text{MG}(\Sigma)$ not associated to any extension of places in $P$.*

Given a set of regions on the marking graph, we can find new regions by considering combinatorial properties on LTS. Two regions, $r_1$ and $r_2$, are *compatible* if there exist three, pairwise disjoint, regions, $u_1, u_2, u_3$ such that $r_1 = u_1 \cup u_2$ and $r_2 = u_2 \cup u_3$. If $r$ is a region, then its set-theoretic complement $Q \setminus r$ is a region. In general, regions are not closed by union and intersection, but the union of compatible regions is always a region. In particular, the union of disjoint regions is a region. Since each region can be associated to a (potential) place on the net, these same properties can be equivalently stated between places of $\Sigma$. In particular, for each place $p \in P$ with extension $r(p)$, its complement $p^c$ is the place with extension $Q \setminus r(p)$; two places $p_1, p_2 \in P$ are *compatible*, denoted with $p_1 \$ p_2$, iff their extensions $r(p_1)$ and $r(p_2)$ are compatible; the union of two compatible places $p_1$ and $p_2$ is the place with extension $r(p_1) \cup r(p_2)$ and is denoted with $p_1 \vee p_2$.

Let $H$ be a set of implicit places of $\Sigma$; then $\Sigma_H$ denotes the net system obtained from $\Sigma$ by adding the implicit places in $H$. For each place $h$ in $H$ we add an arc to the flow relation from $t$ to $h$ for each $t$ in ${}^\bullet h$ and an arc from $h$ to $t'$ for each $t'$ in $h^\bullet$. An implicit place belongs to the initial marking of $\Sigma_H$ if $m_0$ belongs to the corresponding region.

**Example 17.** *The left of Fig. 2.14 represents the 1-safe system of Fig. 2.4 with an implicit place, coloured in grey. Its marking graph is on the right of the same figure. The region delimited by the red line on the marking graph is the extension of place $p_8$. The implicit place derives from the region coloured in green on the transition system. This region is not represented in Fig. 2.4, and it is not necessary to solve any separation problem. It is the union of the extensions of places $p_4$ and $p_8$.*

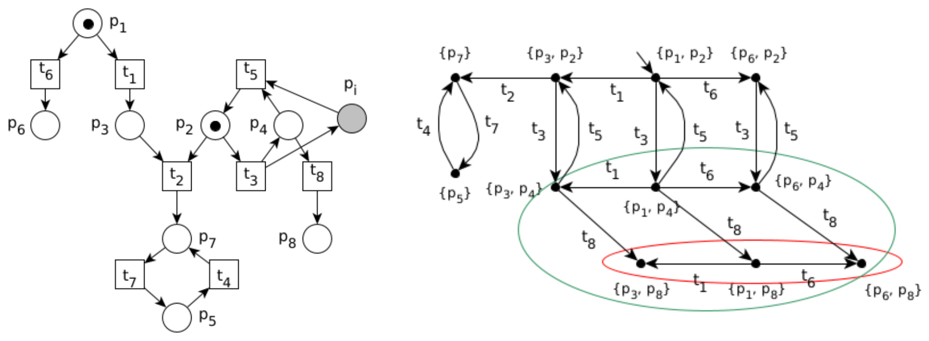


Figure 2.14: An implicit place on the 1-safe system in Fig. 2.4.

# Chapter 3

# Information flow between transitions of a Petri net

In distributed systems, the actions happening in one of the components may not be totally observable by another one, for example for physical distance or for security reasons, but the two components may still be able to interact with each other. If the structure of the system is known, even if an action is not directly observable, sometimes a user can deduce its occurrence by the observation of other parts of the system. This information flow between events of a system may be wanted, for example to discover a fault in a system, or unwanted, if private information can be deduced from public one.

Formal methods can help to analyse the information flow in a distributed system. This has several applications, among them, the *diagnosis* [70], where we want to know if a set of observations is enough to detect an invisible fault in the system, the *noninterference* [32], where we need to guarantee that the observation of public events does not reveal information about the occurrence of private events, and the *opacity* [31], where we require that for any execution of the system satisfying a secret property, there is another execution with the same observations and such that the property is not satisfied.

In this chapter, we focus on a particular method for the analysis of information flow on distributed systems modelled with Petri nets, based on a *reveals* and an *excludes* relation between transitions of the P/T system [70, 21].

Sec. 3.1 recalls the definitions already introduced in the literature, and introduces some new relations based on reveals; Sec. 3.2 studies the complexity of checking some of these relations in 1-safe systems. Sec. 3.3 proposes an algorithm for checking the relations on bounded equal-conflict systems. Part of the results presented in these sections are published in [7, 6, 1]. Finally, Sec. 3.4 discusses different notions of diagnosis, noninterference, and opac-

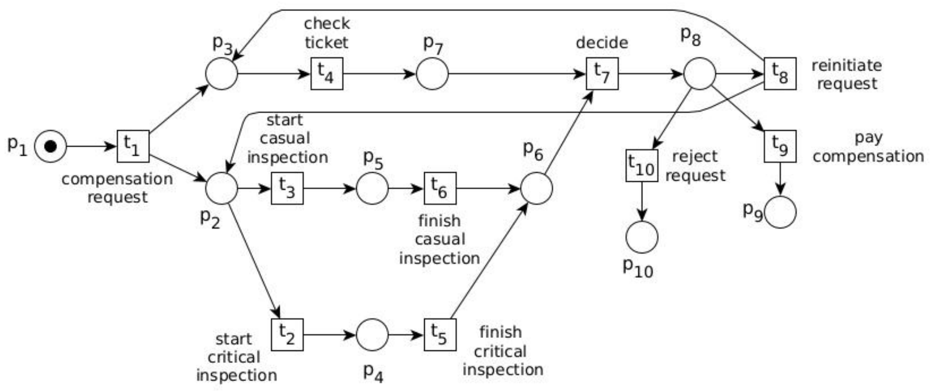


Figure 3.1: Petri net model of a business process.

ity, and shows how reveals have been used or can be used in these contexts.

## 3.1 Reveals and excludes relations

In the analysis of information flow, we are interested in studying whether an agent allowed to observe only some transitions of a P/T system is able to gain information about the occurrence of other hidden transitions. The information may be *positive*, if the agent discovers the occurrence of a hidden transition through its observations, or *negative*, if the user discovers the impossibility of a hidden transition to occur. The *reveals* relation was introduced in [70], with the name of covering relation, to model positive information flow between the events in the unfolding of a 1-safe system. In [12], the authors propose an extended reveals relation, generalising the relation in [70]. In this thesis we focus on reveals relations on the transitions of the P/T system, rather than on the unfolding of the net; this version was defined in [21]. In the same paper, the authors formalized an excludes relation between transitions of a P/T system, to model the negative information flow.

Before proceeding with formal definitions, we give an intuition of the reveals and excludes relations with an example.

**Example 18.** *The P/T system in Fig. 3.1 is adapted from a model in Chap. 2.3 of "Process Mining" book by Aalst [124]. The net models the handling of compensation requests within an airline. A customer may place a request for various reasons. After a request, the ticket is checked and an inspection is performed in parallel. However, inspection can be in two different ways, which is modelled via conflicting activities. Transition $t_2$ and $t_5$ model a critical inspection activity, which is performed for suspicious* *or complex requests. Transition $t_3$ and $t_6$ model a casual inspection activity, which is performed for regular requests. A decision is made only after that the check of the ticket and the inspection have been performed.*

*There are three possible outcomes of the decision: the requested compensation is paid, the request is declined, or further processing is needed. In the first two cases the process is finalised. In the latter case the process returns to the marking $\{p_2, p_3\}$.*

*Since each execution of the P/T system must start with transition $t_1$, even if $t_1$ is not directly observable, observing the occurrence of any other transition* reveals *that $t_1$ must have occurred. This is not a security concern, since we can assume that for each component involved in the analysis of the request, it is not a secret that a request has been sent. We may be concerned if a user that can observe only the outcome of a request, can deduce on which kind of inspection the request went through. This is not the case in this model, since before the occurrence of $t_8$, $t_9$, and $t_{10}$, both $t_2$ and $t_3$ may have occurred; hence, $t_8$, $t_9$, and $t_{10}$ do not reveal neither $t_2$ nor $t_3$.*

*In this P/T system we can detect also some negative information flow: the occurrence of $t_{10}$* excludes *the occurrence of $t_9$, and vice versa.*

In Sec. 3.1.1 we recall some of the definitions in [21], and in Sec. 3.1.2 we propose new parametric relations based on reveals.

### 3.1.1 Reveals and excludes relations in the literature

In [21] reveals, extended-reveals and excludes relations are defined for the transitions of a Petri net with the aim of modelling information flow in concurrent systems. For all this relations, we assume *progress* of the system, namely, an enabled transition either fires or gets disabled by another transition in conflict with it.

#### Reveals relation

The reveals relation was originally introduced for events of an occurrence net in [70]. In [72] the authors applied it in the field of fault diagnosis. In [21] the reveals relation is redefined for the transitions of a 1-live P/T system.

Transition $t_1$ reveals transition $t_2$ if each maximal configuration which contains an occurrence of $t_1$ also contains at least one occurrence of $t_2$. This means that, from the occurrence of $t_1$, we can deduce that $t_2$ has already occurred or will inevitably occur. In other words, the occurrence of $t_1$ implies the occurrence of $t_2$ either in the past or in the future. The formal definition follows.

**Definition 9.** *Let $\Sigma = (P, T, F, W, m_{in})$ be a 1-live P/T system,* $\text{UNF}(\Sigma) = (B, E, F, \lambda)$ *be its unfolding, and $C_{max}$ be the set of all its maximal configurations. Let $t_1, t_2 \in T$ be two transitions;* $t_1$ reveals $t_2$*, denoted $t_1 \rhd t_2$, iff*

$\forall\ C \in C_{max}$

$$t_1 \in \lambda(C) \implies t_2 \in \lambda(C).$$

Reveals is a reflexive and transitive relation. However, it is neither symmetric nor anti-symmetric.

**Example 19.** *As already discussed in Ex. 18, in Fig. 3.1 transition $t_9 \rhd t_1$, but $t_1 \not\rhd t_9$, since there is a maximal configuration in the unfolding with $t_1$ and without $t_9$. Reveals is also not anti-symmetric, for example in Fig. 3.1 $t_3 \rhd t_6$ and $t_6 \rhd t_3$.*

### Extended-Reveals relation

In some cases, one transition alone does not give much information about the occurrence of another transition. However, the occurrence of a set of transitions might give information about the occurrence of others. For example, the occurrence of all the transitions in a set $Y$ can imply the occurrence of some transitions in set $Z$. This relation was originally defined in [12] for the events of an occurrence net and used in the field of fault diagnosis. Later the relation was expressed in terms of the transitions of a Petri net [21], and used to analyse noninterference for information-flow security.

**Definition 10.** *Let $\Sigma = (P, T, F, W, m_{in})$ be a 1-live P/T system,* $\textsc{unf}(\Sigma) = (B, E, F, \lambda)$ *be its unfolding, $Y, Z \subseteq T$, and $C_{max}$ be the set of all maximal configurations. If there is at least a $C \in C_{max}$ where all the element of $Y$ occur, then $Y$* extended-reveals *$Z$, denoted $Y \twoheadrightarrow Z$ iff* $\ \forall\ C \in C_{max}$

$$\bigwedge_{t_i \in Y} t_i \in \lambda(C) \implies \bigvee_{t_j \in Z} t_j \in \lambda(C).$$

Extended-reveals relation is reflexive and nontransitive. It is neither symmetric nor anti-symmetric. The reveals relation coincides with the extended-reveals relation between singletons, i.e., $t_1 \rhd t_2$ can be written as $\{t_1\} \twoheadrightarrow \{t_2\}$.

**Example 20.** *In the P/T system in Fig. 3.2, although $t_4$ reveals both $t_1$ and $t_2$, neither $t_1$ nor $t_2$ reveals $t_4$ alone. However, if both $t_1$ and $t_2$ occur, then $t_4$ will inevitably occur because of the progress assumption. So we can write $\{t_1, t_2\} \twoheadrightarrow \{t_4\}$. Similarly, both $t_5$ and $t_6$ reveal $t_9$, but $t_9$ reveals neither $t_5$ nor $t_6$. However, the occurrence of $t_9$ implies that either $t_5$ or $t_6$ will inevitably occur. This can be expressed by the extended-reveals relation $\{t_9\} \twoheadrightarrow \{t_5, t_6\}$.*

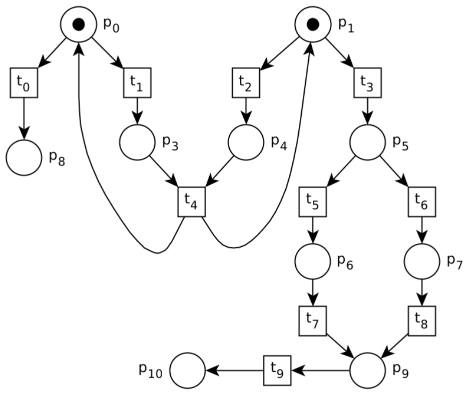


Figure 3.2: A P/T system.

## Excludes relations

Excludes relation was also introduced in [21] for transitions of a 1-live Petri net in order to express negative information flow. Two transitions exclude each other if they never occur in the same run. This means that the occurrence of one implies the nonoccurrence of the other.

**Definition 11.** *Let $\Sigma = (P, T, F, W, m_{in})$ be a 1-live P/T system,* $\textsc{unf}(\Sigma) = (B, E, F, \lambda)$ *be its unfolding, $C_{max}$ the set of all its maximal configurations, and let $t_1, t_2 \in T$. $t_1$* excludes *$t_2$, denoted $t_1$ **ex** $t_2$, iff $\forall\ C \in C_{max}\ \ t_1 \in \lambda(C) \implies t_2 \notin \lambda(C)$.*

By definition, excludes is a symmetric relation. It is neither transitive nor reflexive. Moreover, it does not coincide with the conflict relation. Transitions which are in conflict at a reachable marking can still appear in the same maximal run, so they may not exclude each other.

**Example 21.** *In the P/T system in Fig. 3.1, $t_9$ **ex** $t_{10}$. Although $t_8$ is in conflict with $t_9$, $t_8 \not{ex}\ t_9$, since there is a configuration on the unfolding in which an event labelled with $t_8$ occurs, and after that, an event labelled with $t_9$ does.*

In [21] the authors proposes two variants of excludes, namely excludes *in the past* and excludes *in the future*, that respectively hold if observing a transition excludes the occurrence of another transition in the past, but that transition may still occur after the observation, and if observing a transition excludes that another transition will ever occur from that moment on, but it could have occurred in the past.

**Definition 12.** *Let $\Sigma = (P, T, F, W, m_{in})$ be a 1-live P/T system,* $\textsc{unf}(\Sigma) = (B, E, F, \lambda)$ *its unfolding, $C_{max}$ the set of all its maximal configurations, and let $t_1, t_2 \in T$. $t_1$* excludes in the past *$t_2$, denoted $t_1$ $\boldsymbol{ex}_p$ $t_2$, iff $\forall\ C \in C_{max}$ , $\forall e_1 \in C \cap E_{t_1}$ , $\nexists e_2 \in E_{t_2} \cap C$ such that ($e_2 \preceq e_1$ or $e_2$ concurrent with $e_1$).*

**Definition 13.** *Let* $\Sigma = (P, T, F, W, m_{in})$ *be a 1-live P/T system,* $\text{UNF}(\Sigma) = (B, E, F, \lambda)$ *its unfolding,* $C_{max}$ *the set of all its maximal configurations, and let* $t_1, t_2 \in T$. $t_1$ excludes in the future $t_2$, *denoted* $t_1$ $\boldsymbol{ex}_f$ $t_2$, *iff* $\forall\ C \in C_{max}$ , $\forall e_1 \in C \cap E_{t_1}$ , $\nexists e_2 \in E_{t_2} \cap C$ *such that* ($e_1 \preceq e_2$ *or* $e_1$ *concurrent with* $e_2$.

Unlike excludes, excludes in the past and excludes in the future are not symmetric relations.

**Remark 3.** $t_1$ $\boldsymbol{ex}_p$ $t_2$ *if, and only if,* $t_2$ $\boldsymbol{ex}_f$ $t_1$.

**Example 22.** *We have already observed in Ex. 21 that in the P/T system in Fig. 3.1* $t_8$ $\not{e}x$ $t_9$. *However,* $t_9$ $\boldsymbol{ex}_f$ $t_8$, *since after that the compensation has been paid, the request cannot be initiated. In the same P/T system,* $t_4$ $\boldsymbol{ex}_p$ $t_9$, *since if the ticket need to be checked, it is not possible that the compensation was already paid.*

### 3.1.2 Reveals relations counting occurrences

In some cases, knowing that a set of transitions occurred in the system does not give any information about the occurrences of hidden transitions, but observing the same transitions multiple times does. In this section, we define three relations based on reveals that keep into account the possibility for the observer to count. These relations generalise reveals and extended reveals, and give the opportunity to specify security requirements of a distributed system in a way that is more tailored to the needs of the system. The definitions here proposed are published in [6, 1].

#### Repeated reveals

The repeated reveals relation extends reveals by checking whether several occurrences of a transition reveal the occurrence of another one.

Intuitively, $t_1$ $n$-repeated reveals $t_2$ if each maximal configuration which has at least $n$ occurrences of $t_1$, also has at least one occurrence of $t_2$.

A first version of repeated reveals was defined in [21] without progress assumption, thus considering all the configurations. In that version, expressing reveals as special case of repeated reveals was not possible. Here, as for the other reveals relations, we consider only maximal configurations; in our definition, reveals is a particular case of repeated reveals, where the observer can count only one occurrence of the transitions.

**Definition 14.** *Let* $\Sigma = (P, T, F, W, m_{in})$ *be a 1-live P/T system,* $\text{UNF}(\Sigma) = (B, E, F, \lambda)$ *be its unfolding,* $C_{max}$ *be the set of all its maximal configurations,* $t_1, t_2 \in T$, *and* $n$ *be a positive integer. If there exists a maximal configuration in which* $t_1$ *occurs at least* $n$ *times, then we say* $t_1$ $n$-repeated reveals $t_2$, *denoted* $n.t_1 \rhd t_2$, *iff:*

$$\forall C \in C_{max} \quad |C \cap E_{t_1}| \geq n \implies C \cap E_{t_2} \neq \emptyset.$$

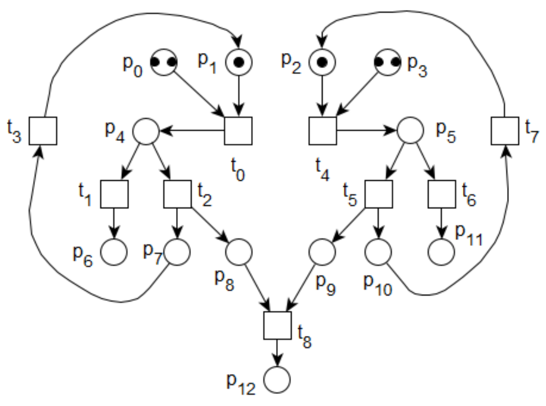


Figure 3.3: A bounded equal-conflict P/T system.

Note that $n$-repeated reveals is not defined if there is no maximal configuration in which $t_1$ occurs at least $n$ times.

The following statements are direct consequences of Def. 14.

**Remark 4.** *Reveals relation (as in Def. 9) coincides with 1-repeated reveals.*

**Remark 5.** *If $n.t_1 \rhd t_2$ and $\exists C \in C_{max}$ such that $|C \cap E_{t_1}| \geq n+1$ then $(n+1).t_1 \rhd t_2$.*

**Remark 6.** $n.t_1 \not\rhd t_2 \implies (n-1).t_1 \not\rhd t_2$.

**Example 23.** *In the P/T system in Fig. 3.1, knowing that the ticket has been checked (transition $t_4$) does not give any information about the outcome of the request ($t_8$, $t_9$, $t_{10}$), but if the tickets is checked twice, then we are sure that the request has been re-initiated, namely $2.t_4 \rhd t_8$.*

*In the P/T system in Fig. 3.3, only one occurrence of $t_0$ does not give information about $t_3$, but if $t_0$ occurs twice, this implies the occurrence of $t_3$, i.e., $2.t_0 \rhd t : 3$. Simmetrically, two occurrences of $t_4$ imply the occurrence of $t_7$, i.e., $2.t_4 \rhd t_7$.*

## Extended-repeated reveals

The following variant of reveals relation is parametric and it generalises both the reveals variants in Sec. 3.1.1 and repeated reveals. Extended-repeated reveals allows one to specify the expected security requirements in a more tailored fashion.

**Definition 15.** *Let $k \geq 1$ and $\{n_1, ..., n_k\}$ be positive integers. Let $\{t_1, ..., t_k\}$, $Y \subseteq T$. If there is at least one maximal configuration in which, for each $i \in \{1, ..., k\}$, each transition $t_i$ of $\{t_1, ..., t_k\}$ occurs at least $n_i$ times, then*

$\{n_1.t_1, ..., n_k.t_k\}$ extended-repeated reveals $Y$, *denoted* $\{n_1.t_1, ..., n_k.t_k\} \twoheadrightarrow Y$, *iff* $\forall\, C \in C_{max}$

$$\bigwedge_{t_i \in \{t_1,...,t_k\}} (|C \cap E_{t_i}| \geq n_i) \implies \bigvee_{t \in Y} (C \cap E_t \neq \emptyset).$$

Note that $\{n_1.t_1, ..., n_k.t_k\}$ extended-repeated reveals $Y$ is not defined if there is no maximal configuration in which each transition $t_i$ of $\{t_1, ..., t_k\}$ occurs at least $n_i$ times.

**Remark 7.** *Reveals, extended reveals and repeated reveals can be expressed by Def. 15.*

$$\begin{aligned} t_1 \rhd t_2 &\iff \{1.t_1\} \twoheadrightarrow \{t_2\}, \\ \{t_1, t_2\} \twoheadrightarrow \{t_3, t_4\} &\iff \{1.t_1, 1.t_2\} \twoheadrightarrow \{t_3, t_4\}, \\ n.t_1 \rhd t_2 &\iff \{n.t_1\} \twoheadrightarrow \{t_2\}. \end{aligned}$$

**Example 24.** *In the P/T system illustrated in Fig. 3.3, let us examine the relation between the transitions $t_0, t_4$ and $t_8$. Neither $t_0$ nor $t_4$ reveals $t_8$ alone. There is no extended reveals or repeated reveals relation between them either. However, there is still some information flow. Two occurrences of $t_0$ together with two occurrences of $t_4$ extended-repeated reveal $t_8$, i.e., $\{2.t_0, 2.t_4\} \twoheadrightarrow \{t_8\}$.*

**Collective reveals**

In the last case that we consider, the total number of occurrences of a set of transitions gives information about another set of transitions. In other words, if the total number of occurrences of the transitions in the first set is more than a certain number, this implies that at least one transition of the second set must have occurred or will occur inevitably. The next relation defines such situation.

**Definition 16.** *Let $n \geq 1$ and $X, Y \subseteq T$. If there is at least one maximal configuration in which the total number of occurrences of the transitions in set $X$ is at least $n$, then we say $X$ $n$-*collective reveals *$Y$, denoted $n.X \twoheadrightarrow Y$, iff $\forall\, C \in C_{max}$*

$$\sum_{t \in X} |C \cap E_t| \geq n \implies \bigvee_{t \in Y} (C \cap E_t \neq \emptyset).$$

Note that $X$ $n$-collective reveals $Y$ is not defined if there is no maximal configuration in which the total number of occurrences of the transitions in set $X$ is at least $n$.

**Remark 8.** *Reveals and repeated reveals can be expressed by Def. 16.*

$$\begin{aligned} t_1 \rhd t_2 &\iff 1.\{t_1\} \twoheadrightarrow \{t_2\}, \\ n.t_1 \rhd t_2 &\iff n.\{t_1\} \twoheadrightarrow \{t_2\}. \end{aligned}$$

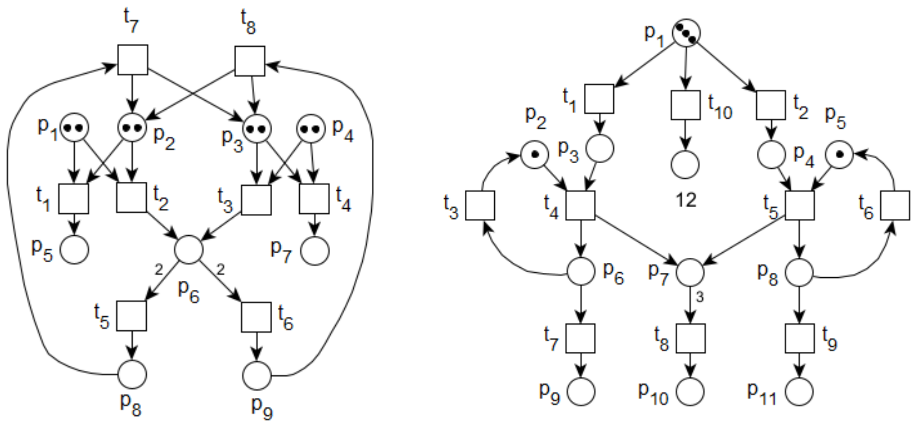


Figure 3.4: Two equal-conflict P/T systems.

**Example 25.** *The P/T system in Fig. 2.1 is recalled on the left of Fig. 3.4. In it, if the total number of occurrences of $t_2$ and $t_3$ is at least 2 ($t_2$ occurs twice, $t_3$ occurs twice or both $t_2$ and $t_3$ occur at least once), then either $t_5$ or $t_6$ must occur, therefore $2.\{t_2, t_3\} \rightarrowtriangle \{t_5, t_6\}$.*

*In the P/T system on the right of Fig. 3.4, if the total number of occurrences of $t_4$ and $t_5$ is at least 3, this implies the occurrence of $t_8$, hence $3.\{t_4, t_5\} \rightarrowtriangle \{t_8\}$.*

## 3.2 Complexity of reveals and excludes in 1-safe systems

In this section we discuss the complexity of computing reveals, repeated reveals, and excludes on 1-safe systems.

The complexity of the other relations based on reveals, and the generalization to general P/T system is left as future work. However, since 1-safe systems are a special case of P/T systems and reveals is generalized by all the relations defined in the previous section for positive information flow, the result on reveals gives a lower bound for the complexity of the other cases. Fig. 3.5 recalls the hierarchy between the relations based on reveals defined in the previous section. In particular, an arrow from a relation $a$ to a relation $b$ denotes that each instance of $b$ can be equivalently expressed as an instance of $a$.

The relations *excludes* has the same complexity of *excludes in the past* and *excludes in the future*, therefore we do not need to study them separately. The equivalence between $\boldsymbol{ex}_f$ and $\boldsymbol{ex}_p$ is stated in Remark 3; for excludes, we observe that given two transitions $t_1$ and $t_2$, $t_1$ $\boldsymbol{ex}$ $t_2$ if, and only if, $t_1\ \boldsymbol{ex}_p\ t_2 \wedge t_1\ \boldsymbol{ex}_f\ t_2$.

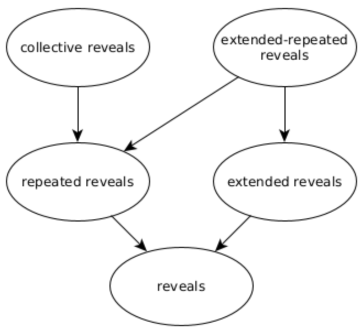


Figure 3.5: Diagram showing the generalizations between relations based on reveals

As first step, we show that deciding reveals and excludes is at least PSPACE, even when the system is 1-safe or free-choice. This provides a lower bound for all the relations based on reveals and excludes. When we consider reveals and excludes, we do not need to check whether the relation is defined on the 1-safe system, since these relations are always defined in 1-live systems. However, checking whether a transition is 1-live in a 1-safe system is also a PSPACE-complete problem, hence removing the 1-liveness assumption would not change the complexity.

**Lemma 2.** *Deciding whether an instance of the reveals or excludes relation is verified on a* 1*-safe system is at least* PSPACE *in the size of the net.*

*Proof.* First, we show that a reachability problem on a 1-safe system can be reduced to checking an instance of a reveals relation on a 1-safe system.

Let $\Sigma = (P, T, F, m_{in})$ be a 1-safe system, and $m \subseteq P$ a marking. Suppose that we need to verify if $m$ is reachable in $\Sigma$. First, we construct a net $\Sigma_1 = (P_1, T, F_1, m_1)$ by adding to $\Sigma$ all the complements of the elements in $P$, if they are not yet in $P$. For each place $p \in P$, we denote with $p^c$ its complement; we define $m' = m \cup \{p^c \in P_1 : p \in P \setminus m\}$; in addition, we add to $\Sigma_1$ a transition $t_m$, and a place $p_m$ such that ${}^\bullet t_m = m'$, $t_m^\bullet = \{p_m\}$. Let $\Sigma_2 = (P_2, T_2, F_2, m_1)$ be such a modification of $\Sigma_1$. The following statements are equivalent: (1) $m$ is reachable in $\Sigma$; (2) $m'$ is reachable in $\Sigma_2$; (3) $\{p_m\}$ is reachable in $\Sigma_2$; (4) $t_m$ is 1-live in $\Sigma_2$.

As last step we add one place $p_0$ and two transitions $t_1, t_2$ to $\Sigma_2$ such that ${}^\bullet p_0 = \emptyset$, $p_0^\bullet = \{t_1, t_2\}$, ${}^\bullet t_1 = {}^\bullet t_2 = \{p_0\}$, $t_1^\bullet = m_1$, $t_2^\bullet = m'$. Let $\Sigma_f = (P_f, T_f, F_f, \{p_0\})$ the net obtained through the described transformation. $\Sigma_f$ is a 1-safe net system, and it is such that after the occurrence of $t_1$ a marking $m$ is reachable iff it is reachable in $\Sigma_2$; in particular this property holds for $\{p_m\}$. By construction, $\{p_m\}$ is reachable after the occurrence of $t_2$. Hence, $m$ is reachable in $\Sigma$ iff $\{p_m\}$ is reachable in $\Sigma_2$. This can be expressed both with reveals and with excludes: $m$ is reachable in $\Sigma$ iff (1) $t_m \not\triangleright t_2$, or equivalently $t_m$ ***ex*** $t_2$. $\square$

**Remark 9.** *Often problems are computationally easier when we consider free-choice systems. This is not the case for the reveals and excludes relations: the reachability problem in free-choice* 1*-safe system is* PSPACE*-complete, since there is a polynomial transformation from a* 1*-safe system* $\Sigma$ *to a* 1*-safe free-choice system* $\Sigma'$ *such that each marking* $m$ *is reachable in* $\Sigma$ *iff* $m$ *is reachable in* $\Sigma'$ *[81, 41, 74]; the construction used to prove Lemma 2 can be slightly modified in order to obtain a free-choice system at the end, where reveals and excludes can be checked.*

In [71] the authors proved that computing reveals is PSPACE-complete also when the relation is defined on events of the unfolding.

Some of the upper bounds for the verification of the relations based on reveals and excludes can be proved by transforming the relations in LTL formulas. In [50], Esparza proved that model-checking LTL in $k$-bounded P/T systems is a PSPACE-complete problem; hence, if a relation can be checked with LTL, verifying it must be in PSPACE.

**Theorem 1.** *Deciding a reveals relation in a 1-safe system is* PSPACE*-complete with respect to the size of the net.*

*Proof.* Lemma 2 proves that deciding reveals is at least a PSPACE problem. We need to show that reveals for 1-safe nets is in PSPACE. This can be done by translating the problem of checking a reveals relation into the model checking of an LTL formula. Since the latter is a PSPACE-complete problem on a 1-safe Petri net [50], computing reveals cannot be harder. Let $\Sigma = (P, T, F, m_{in})$ be a 1-safe Petri net, $a, b \in T$. Assume we need to check if $a \triangleright b$. We first transform the net $\Sigma$ into a net $\Sigma' = (P', T', F', m_{in})$ by replacing each transition $t$ with two transitions $t_1$ and $t_2$ such that ${}^\bullet t_1 = {}^\bullet t$ and $t_2^\bullet = t^\bullet$, and adding a new place $p_t$ such that $t_1^\bullet = {}^\bullet t_2 = \{p_t\}$, ${}^\bullet p_t = \{t_1\}$, and $p_t^\bullet = \{t_2\}$. This transformation is linear in the size of the net. Computing $a \triangleright b$ on $\Sigma$ is equivalent to model-check the formula $Fp_a \rightarrow Fp_b$ on the set of maximal runs of $\Sigma'$, namely in all the runs in which no transition is permanently enabled. For each transition $t \in T'$, let $\mathit{enabled}(t) = \bigwedge_{p \in {}^\bullet t} p$ be the formula denoting that transition $t$ is enabled. Then, checking if $a \triangleright b$ is equivalent to model-check $\psi = (\bigwedge_{t \in T'} \neg FG(\mathit{enabled}(t))) \rightarrow (Fp_a \rightarrow Fp_b)$. The length of $\psi$ is polynomial with respect to $\mid P \mid + \mid T \mid$, and since the model-checking is PSPACE in the size of the formula, it is also PSPACE in the size of the net.

We now need to check that checking if $a \triangleright b$ is at least PSPACE, even when we consider free-choice nets. We prove it by reducing a reachability problem on a 1-safe net to a reveals problem on a 1-safe free-choice net.

$\square$

**Theorem 2.** *Deciding an excludes relation in a 1-safe Petri net is* PSPACE-*complete with respect to the size of the net.*

*Proof.* To show that checking an excludes relation is in PSPACE, we reduce it to a LTL formula. Let $\Sigma = (P, T, F, m_0)$ be a 1-safe Petri net, $a, b \in T$. Assume we need to check if $a$ ***ex*** $b$. We split each transition into two, as described in the proof of Theorem 1, obtaining the net $\Sigma'$. Computing $a$ ***ex*** $b$ is equivalent to model-check the formula $Fp_a \rightarrow G(\neg p_b)$ on the set of maximal runs of $\Sigma'$, that is equivalent to model-check the formula $\psi = (\bigwedge_{t \in T'} \neg FG(\mathit{enabled}(t))) \rightarrow (Fp_a \rightarrow G(\neg p_b))$ on the entire net. The length of $\psi$ is polynomial with respect to $\mid P \mid + \mid T \mid$, and since the model-checking is PSPACE in the size of the formula, it is also PSPACE in the size of the net. This and Lemma 2 conclude the proof. hence . □

## Repeated reveals in 1-safe nets

The repeated reveals relation $n.a \rhd_r b$ holds iff there is at least a maximal run with $n$ occurrences of $a$, and each of them has at least an occurrence of $b$.

Since reveals is a special case of repeated reveals ($1.a \rhd_r b \equiv a \rhd b$), checking repeated reveals is PSPACE-hard. To show that it is PSPACE-complete we prove that both checking if there is a maximal run with at least $n$ occurrences of $a$, and checking if in each of them there is at least an occurrence of $b$ can also be done in PSPACE, when $n$ is a given constant.

**Lemma 3.** *Let* $\Sigma = (P, T, F, m_{in})$ *be a* 1*-safe net. Checking the existence of a maximal run with* $n$ *occurrences of a transition* $a$ *can be done in* PSPACE*.*

*Proof.* We modify $\Sigma$ by adding a place $p$ precondition of $a$ with $n$ tokens in the initial marking. The net $\Sigma_n$ obtained in this way is $n$-safe. There is a maximal run in $\Sigma$ with at least $n$ occurrences of $a$ iff the CTL formula $EF\neg p$ is verified on $\Sigma_n$. Since the model-checking of CTL is PSPACE-complete for bounded nets [40], the thesis follows. □

**Lemma 4.** *Checking if each maximal run with at least* $n$ *occurrences of* $a$ *has at least an occurrence of* $b$ *is at most* PSPACE*.*

*Proof.* We construct a net $\Sigma' = (P', T', F, m_{in})$ so that each transition $t \in T$ is split into two and there is a place $p_t$ in the middle denoting its occurrence. We can translate the problem of checking if each maximal run of $\textsc{unf}(\Sigma)$ with at least $n$ occurrences of $a$ has at least an occurrence of $b$ in a LTL formula on the places of $\Sigma'$. If $n = 2$, we define

$$\psi_2 = ((\mathcal{U}(\neg p_a, p_a \wedge \mathcal{U}(p_a, \neg p_a \wedge Fp_a))) \rightarrow Fp_b),$$

$n.a \rhd_r b$ if the following formula holds.

$$\bigwedge_{t \in T'} \neg FG(\mathit{enabled}(t)) \rightarrow \psi_2.$$

The second until in $\psi_2$ is needed when we consider the state-based logic of the 1-safe $\Sigma'$, because we need to be sure that we are reaching two different occurrences of transition $a$, and we are not just firing a concurrent transition while the place $p_a$ is still marked. For $n > 2$, $n.a \rhd_r b$ is expressed by the following formula

$$\bigwedge_{t \in T'} \neg FG(\mathit{enabled}(t)) \rightarrow (\mathcal{U}(\neg p_a, p_a \wedge \mathcal{U}(p_a, \neg p_a \wedge \psi_{n-1}))) \rightarrow Fp_b.$$

Given $n$ the size of this formula is fixed. □

## 3.3 Algorithms for finding reveals and excludes on bounded equal-conflict systems

In this section we propose algorithms for computing the relations defined in Sec. 3.1 on bounded equal-conflict systems. The results in this section are published in [7, 6, 1]. As discussed in Sec. 2.1.1 and Sec. 2.3.1, this class of systems allows for the use of maximal step semantics, and this can be exploited in the development of algorithms. In particular, in algorithms discussed here, we will use prefixes of the tree structure defined in Sec. 2.3.2.

This section is organized as follows. In Sec. 3.3.1 we present the tree structure on which the algorithms are defined, and its properties. From Sec. 3.3.2 to Sec. 3.3.5, we discuss how to compute the relations of Sec. 3.1.

Sec. 3.3.6 proposes a reduction of the tree structure, that in several cases can be used to have more efficient algorithms.

We implemented most of the algorithms presented in this section, and made them available at `https://github.com/MC3-lab/mscTree`. Sec. 3.3.7 discusses the implementation and some preliminary experiments.

### 3.3.1 Prefix of the tree-unfolding

Let TS $= (Q, U, A, q_0)$ be an LTS, the TS-tree defined in Sec. 2.3.2 can be an infinite structure, even if TS is finite. A full prefix is a finite prefix of a TS-tree, from which we can reconstruct all the information of the tree. It is constructed starting from the root of the tree, and adding all the children of its nodes, until we meet a node associated to a state that was already visited in the path.

**Definition 17.** *Let* TS $= (Q, U, A, q_0)$ *be a labelled transition system on alphabet $T$, where $U$ is the multiset of the elements in $T$. The* full prefix *of the* TS*-tree, denoted* fp(TS)*, is a labelled rooted tree defined by the following clauses.*

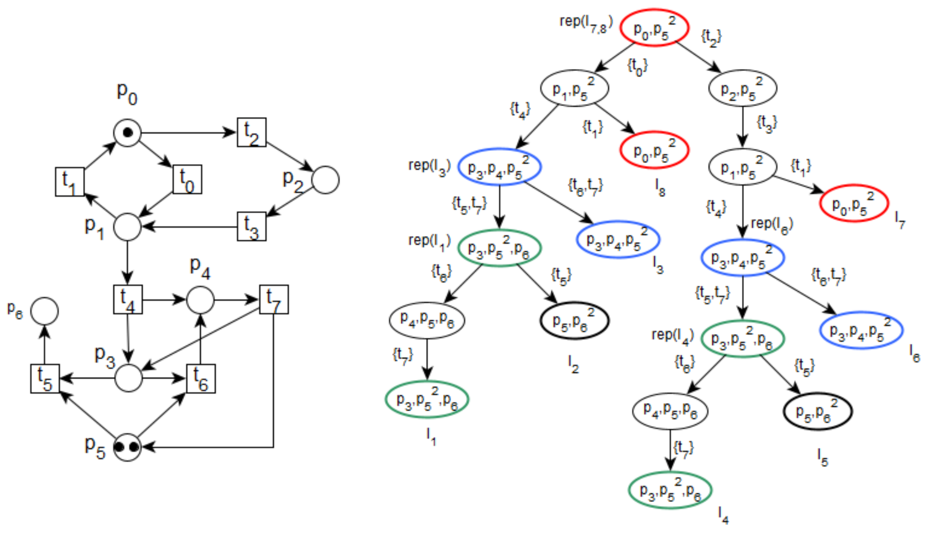


Figure 3.6: A P/T system and the full prefix of the tree unfolding its mmg.

1. *The root of* fp(TS) *is the root of the* TS*-tree.*

2. *Let $v$ be a node of* fp(TS)*. If there is no $v'$ such that $v' < v$ and $v, v'$ are associated to the same element in $Q$, then all the children of $v$ in the* TS*-tree are nodes in* fp(TS)*; otherwise $v$ is a leaf in* fp(TS)*.*

Given a leaf $l$ of the full prefix, let rep($l$) denote the ancestor of $l$ corresponding to the same state. If $l$ corresponds to a deadlock, then rep($l$) is undefined.

The subtree inherits all the notations defined for the tree in Sec. 2.3.2. We denote with |fp(TS)| the number of nodes in the prefix.

In the special case in which TS is the mmg of a bounded equal-conflict P/T system, we will show that the full prefix of the mmg-tree is sufficient to compute the information flow of the P/T system.

**Example 26.** *Fig. 3.6 shows a P/T system and the full prefix associated to its* mmg *(which is represented in Fig. 2.7). In the prefix, the leaves exhibit an additional label $l_1$, $i \in \{1, ..., 8\}$. For each non-deadlock leaf $l_i$, its ancestor associated to the same state is labelled* rep($l_i$)*. The leaves and their ancestors are also coloured in the same way. Deadlock leaves are drawn with black bold lines.*

The following lemmas state some useful properties of the full prefix.

**Lemma 5.** *Let* TS $= (Q, U, A, q_0)$ *be a finite* LTS*. The full prefix of the* TS*-tree is finite.*

*Proof.* Since the number of transitions is finite, for each node the number of children is finite. Let $n = |Q|$ be the total number of states in TS, no path of the tree can have more than $n + 1$ nodes without reaching a state that was already visited. □

This lemma holds in particular for the mmg of bounded equal-conflict P/T systems, since they have a finite number of markings.

**Lemma 6.** *For each maximal path $\pi$ of a* TS*-tree, there is a maximal path $\pi'$ of its full prefix, such that $\pi'$ is a prefix of $\pi$.*

*Proof.* Let $\pi$ be a maximal path of an TS-tree and $q$ be the last node of the intersection between $\pi$ and the full prefix. $q$ cannot have any child in the prefix, otherwise one of them should also be in the intersection, since by definition, if $q$ is not a leaf, all the children in the TS-tree are also in its full prefix. Hence $q$ must be a leaf, and the path from the root to $q$ is maximal in the prefix. □

**Lemma 7.** *Let* $\text{TS} = (Q, U, A, q_0)$ *be an* LTS*,* TS*-tree be the tree-unfolding of* TS*, and* fp(TS) *be its full prefix. For each node $v$ in* TS*-tree, there is at least a node $v'$ in* fp(TS) *such that $v$ and $v'$ are associated to the same element of $Q$.*

*Proof.* Let $v$ be a node of TS-tree. If $v$ is also in fp(TS), the proof is trivial. Otherwise, $v$ must follow a leaf in fp(TS), since by definition, for each $v_i$ internal node on fp(TS), for each child $v_{i+1}$ of $v_i$ in TS-tree, $va_{i+1}$ is also in fp(TS). Let $v_j$ be the leaf preceding $v$ and $\pi$ a maximal path on TS-tree including $v$. By construction, there must be a node $v'_j < v_j$ such that $v_j$ and $v'_j$ are associated to the same state. We can peel $\pi$ (Def. 5) by identifying $v'_j$ with $v_j$. In the peeled path $\pi'$, let $v'$ be the node associated to $v$ by the peeling operation. There are two cases: (1) there is no pair of nodes $v_1$, $v'_1$ such that associated to the same state, and such that $v_1 < v'_1 < v'$. Then, by construction $v'$ is in fp(TS). (2) $v'$ follows two nodes $v_1$ and $v'_1$ associated to the same state. In this case, we can repeat the peeling procedure until a node associated with $v$ is in fp(TS). This will happen in a finite number of steps, since each time the number of nodes between the node associated with $v$ and the root decreases. □

**Remark 10.** *If $v$ is not a deadlock, at least an occurrence of $v'$ as described in Lemma 7 is an internal node. In order to see it, we can observe that if $v'$ is a leaf and it is not a deadlock, then there is another node in* fp(TS) *preceding $v'$ and associated to the same state by construction.*

A consequence of Lemma 7 is that when we consider the mmg of a bounded equal-conflict P/T system $\Sigma$, at least an occurrence of all the transitions in $\Sigma$ appears in fp(mmg): let $t$ be a transition in $\Sigma$, an occurrence of $t$ must appear also in mmg-tree (we are considering 1-live systems). Let $v$ be a node in mmg-tree and $u$ the label of an outgoing arc from $u$ such that $t \in u$; by Lemma 7, there is a node $v'$ in fp(mmg) corresponding to the same marking, and an outgoing arc from $v$ labelled with $u$.

**Lemma 8.** *Let* $\text{TS} = (Q, U, A, q_0)$ *be a labelled transition system on alphabet* $T$*, where* $U$ *is the set of multisets of the elements in* $T$*. Let* $\pi$ *be a path on the* TS*-tree, and* fp(TS) *be its full-prefix. If* $n \leq \lambda(\pi)(t)$*, but* $t \notin \lambda(\pi \cap \text{fp(TS)})$*, then there is another path* $\pi' \in$ TS*-tree such that* $\lambda(\pi') \subseteq \lambda(\pi)$*,* $n \leq \lambda(\pi')(t)$*, and* $t \in \lambda(\pi' \cap \text{fp(TS)})$*.*

*Proof.* Let $l(\pi_1)$ be the leaf of $\pi_1 = \pi \cap \text{fp(TS)}$ and $\text{rep}(l(\pi_1))$ its repetition in the prefix. We can construct the run $\pi'_1 = \textit{peel}(\pi, \pi_{\text{rep}(l(\pi_1)), l(\pi_1)})$. If $t \in \pi_2 = \pi'_1 \cap \text{fp(TS)}$ we can stop the construction, otherwise we repeat the procedure by considering $l(\pi_2)$. Since the distance between the root and the first occurrence of $t$ is finite, and every step of the peeling reduces it, after a finite number of peeling operations the thesis must be satisfied. □

We now define an extension operation on finite paths.

**Definition 18.** *Let* $\text{TS} = (Q, U, A, q_0)$ *be an* LTS*,* $\pi = v_0 u_0 v_1 u_1 ... v_n$ *be a finite path in* TS*-tree, and* fp(TS) *be the full prefix. For each internal node* $v \in$ fp(TS) *associated to the same state as* $v_n$*, and for each* $\pi'$ *starting in* $v$*, we define the* extension *of* $\pi$ *through* $\pi'$ *as the function* $\text{ext}(\pi, \pi') = \pi \cdot \pi_{ex}$*, where* $\pi_{ex}$ *is the path with first node* $v_n$ *and such that* $\pi_{ex} \simeq \pi'$*.*

The existence of $\pi_{ex}$ in Definition 18 is a direct consequence of Lemma 1. The extension operation allows us to state another property of the full prefix, that establish a relation between its paths and the paths of the TS-tree. This is expressed by the following lemma.

**Lemma 9.** *For each* $\pi$ *finite path in the* TS*-tree, if* $\pi$ *is not contained in* fp(TS)*, then we can find a path including it, by starting from a maximal path of the full prefix and applying recursively the extension operation.*

*Proof.* Consequence of Lemma 1, Lemma 7 and Remark 10. □

The result of the previous Lemma applies also to infinite paths when considering the limit of applying recursively the extension operation. This means that when we consider mmg, we can extract all possible system behaviour under maximal-step semantics from the full prefix.

The algorithms provided in the next sections works on the full prefix of mmg of bounded equal-conflict P/T systems, and require a previous computation of the full prefix. Hence, their running time depends on the number of distinct transitions of the system, as well as on the size of the full prefix. It is therefore noteworthy that the length of the full prefix is at most the number of markings of the system that are reached under maximal-step semantics, plus one repeated marking. In case of paths composed of many cycles, the path will be as large as the least common multiple of the length of these cycles, since after this number of maximal steps, each cycle will be back at its initial conditions. In each node, the number of children depends on the number of transitions concurrent and in conflict in the corresponding marking. In particular, if the marking $m$ enables $n$ groups of concurrent transitions, each of them with $k_i$, $i \in \{1, ..., n\}$, transitions in conflict, then the node associated with the marking $m$ will have $\Pi_{i=1}^{n} k_i$ children. Once the full prefix is computed, it can serve all the subsequent information flow analysis, as presented next.

### 3.3.2 Computing reveals

In this section we describe how to compute the reveals relation. As already discussed, reveals can also be seen as a special case of more complex relations, whose algorimths are given in Sec. 3.3.4 and Sec. 3.3.5. However, I decided to discuss reveals separately for two reasons: (1) being it a basic relations on which many others build, discussing it separately may be helpful for a better understanding of the properties of the full prefix and its role in information flow analysis; (2) this section provide a procedure that allows to check all the reveals relations by looking only once at the full prefix, whereas the other algorithms either need larger computations (this is the case of Algorithm 2), or they focus on single relations (as in Algorithm 3).

Let $t_1$ and $t_2$ be two transitions of a bounded equal-conflict P/T system $\Sigma$, $t_1$ reveals $t_2$ if, and only if, each maximal configuration which includes an occurrence of $t_1$ also includes an occurrence of $t_2$ (see Definition 9). In bounded-equal conflict P/T systems, due to the relation between the footprints of maximal configurations of the unfolding, and the footprints of maximal paths on the mmg-tree, $t_1$ reveals $t_2$ if, and only if, each maximal path of the mmg-tree of $\Sigma$ which includes an occurrence of $t_1$, also includes an occurrence of $t_2$.

In the following example, we study the reveals relation among transitions of a P/T system on the full prefix of its mmg-tree.

**Example 27.** *Consider the prefix tree in Fig. 3.6, from it we can see that $t_5 \rhd t_4$, but $t_4 \not\rhd t_5$ (the path ending with the blue leaf $l_6$ has an occurrence of $t_4$, but none of $t_5$). Other relations are $t_6 \rhd t_7$, $t_2 \rhd t_3$, and $t_3 \rhd t_2$. No transition reveals $t_0$, since all of them, except for $t_0$, appear in the subtree starting with the branch labelled with $t_2$.*

Next, we prove that, for bounded equal-conflict P/T systems, the full prefix is enough to compute the reveals relation. In other words, if $t_1 \rhd t_2$ in $\Sigma$, then each maximal path of fp(mmg($\Sigma$)) which has an occurrence of $t_1$ also has an occurrence of $t_2$ and vice versa.

**Theorem 3.** *Let $\Sigma = (P, T, F, W, m_{in})$ be a bounded equal-conflict P/T system, $t_1, t_2 \in T$, $A$ be the* $\text{mmg}(\Sigma)$*-tree and* $L = \text{fp}(\text{mmg}(\Sigma))$ *be its full prefix.* $\Pi_A^{t_1} \subseteq \Pi_A^{t_2}$ *if, and only if,* $\Pi_L^{t_1} \subseteq \Pi_L^{t_2}$

*Proof.* First, we prove that if $\Pi_L^{t_1} \not\subseteq \Pi_L^{t_2}$ then $\Pi_A^{t_1} \not\subseteq \Pi_A^{t_2}$. Let $\pi$ be a maximal path in $L$ such that $\pi \in \Pi_L^{t_1}$ and $\pi \notin \Pi_L^{t_2}$. If $\text{leaf}(\pi)$ is a deadlock the conclusion is trivial because $\pi$ is a maximal path in $A$. Otherwise, $\text{rep}(\text{leaf}(\pi))$ exists in $\pi$, and we can extend infinitely many times $\pi$ with an isomorphic copy of the segment between $\text{rep}(\text{leaf}(\pi))$ and $\text{leaf}(\pi)$. This procedure produces a maximal path in $A$ with the same labels of $\pi$, therefore without occurrences of $t_2$.

We now suppose that $\Pi_A^{t_1} \not\subseteq \Pi_A^{t_2}$ and we show that $\Pi_L^{t_1} \not\subseteq \Pi_L^{t_2}$. The proof proceeds with an iterative procedure meant to find a maximal path of the prefix with an occurrence of $t_1$ and no occurrence of $t_2$, thus showing that $t_1$ does not reveal $t_2$ in $L$. Informally, this procedure constructs, at each step, a maximal path of $A$ with an occurrence of $t_1$ and no occurrence of $t_2$, in such a way that the number of elements in the past of its first occurrence of $t_2$ is reduced at each step. Since there can only be finitely many such elements, the procedure will reach a maximal path of $L$ containing an occurrence of $t_1$ but none of $t_2$, in a finite number of steps.

By hypothesis, there is a maximal path $\pi' \in \Pi_A : \pi' \in \Pi_A^{t_1}$, and $\pi' \notin \Pi_A^{t_2}$. By Lemma 6, there is $\pi \in \Pi_L$ that is a prefix of $\pi'$. Note that $\pi$ has no occurrence of $t_2$ —otherwise, so would $\pi'$—, and so if it has an occurrence of $t_1$, then $\pi \in \Pi_L^{t_1}$ and $\pi \notin \Pi_L^{t_2}$. Now, suppose that $t_1$ does not occur in $\pi$, and $v_f = \text{leaf}(\pi)$. If $v_f$ were a deadlock, we would necessarily have that $\pi = \pi'$, but this is a contradiction, since $t_1$ occurs in $\pi'$ and not in $\pi$. Then $\text{rep}(v_f)$ is defined, there is only a finite number of elements in $\pi'$ which are in the past of $t_1$, and at least one of these must be found between $\text{rep}(v_f)$ and $v_f$, not to contradict maximal-step semantics. The path $\pi'' = \text{peel}(\pi')$ is maximal in $A$, and by Lemma 6, there is a maximal path $\pi^1 \in \Pi_L$ which is a prefix of it. Clearly, $t_2$ does not occur in $\pi''$ or $\pi^1$, and so if $t_1$ occurs in $\pi^1$ we reach the desired conclusion. Otherwise, consider $\text{leaf}(\pi^1)$. Clearly, it is not a deadlock, and by construction, $\text{rep}(\text{leaf}(\pi^1))$ is defined. Since the number of elements between the root and the first occurrence of $t_1$ in $\pi''$ is strictly less than that between the root and the first occurrence of $t_1$ in $\pi'$, we may iterate the procedure by finding prefixes $\pi^2, \ldots, \pi^n$ with no occurrences of $t_2$, until eventually $\pi^n$ will have an occurrence of $t_1$, for a finite $n$. $\square$

With the above proof, we conclude that all the reveals relations among the transitions of $\Sigma$ can be found by computing the footprints of the maximal paths of $\text{fp}(\text{mmg}(\Sigma))$ and checking for the corresponding labels among them. Note that maximal paths of $\text{fp}(\text{mmg}(\Sigma))$ are finite, whereas maximal configurations of the unfolding are in general infinite.

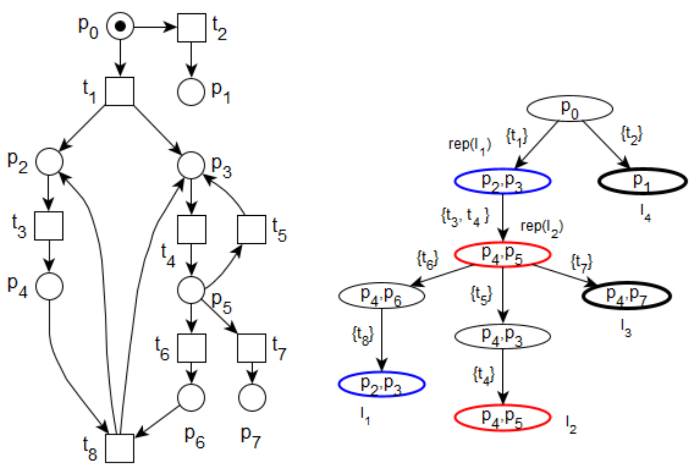


Figure 3.7: A 1-safe free-choice system and the full prefix of the tree-unfolding of its marking graph of maximal steps.

### 3.3.3 Computing excludes

Let $t_1$ and $t_2$ be two transitions of a P/T system $\Sigma$, we say that $t_1$ excludes $t_2$ (or $t_2$ excludes $t_1$ by symmetry of the relation) if, and only if, no maximal configuration of $\textsc{unf}(\Sigma)$ has both an occurrence of $t_1$ and of $t_2$ (see Definition 11). In bounded equal-conflict P/T systems, due to the relation between maximal configurations of an unfolding and maximal paths of the mmg($\Sigma$)-tree discussed in Sec. 2.4, this is directly translated to the following: $t_1$ excludes $t_2$ if, and only if, no maximal path of mmg($\Sigma$)-tree displays both $t_1$ and $t_2$. In other words, let $A$ be msct $(\Sigma)$, $t_1$ ***ex*** $t_2$ iff $\Pi_A^{t_1} \cap \Pi_A^{t_2} = \emptyset$.

In the following example, we will study the excludes relation among transitions of a bounded equal-conflict P/T system on the full prefix of its mmg-tree.

**Example 28.** *Fig. 3.7 represents a P/T system $\Sigma$ on the left, and the full prefix of its marking graph of maximal steps. As before, the leaves are represented with thick black lines if they are deadlocks, and with the same colour of their ancestor associated to the same marking if they are repetitions. The additional labels $l_i$, $i \in \{1, 2, 3, 4\}$, and* rep($l_i$) *also identify leaves and their repetitions. In $\Sigma$, $t_2$ excludes all the other transitions. By symmetry of the excludes relation, each transition (except $t_2$ itself) excludes $t_2$. These are the only excludes relations in the net.*

*Now let us examine the full prefix,* fp(mmg($\Sigma$)), *here denoted with $L$ for the sake of simplicity, and consider the transitions $t_1$ and $t_2$. The initial node corresponds to the marking $\{p_0\}$. From the initial node we can either continue with an occurrence of $t_1$ or an occurrence of $t_2$ (and these are the only occurrences of $t_1$ and $t_2$ in $L$). None of the maximal paths of $L$ has* *occurrences of $t_1$ and $t_2$ together. However, we cannot conclude $t_1$ excludes $t_2$ by looking at the maximal paths of the full prefix. We need to make sure the maximal paths of the $L$ do not have the two transitions together. To do this, we do not need to construct the whole tree. We can use the full prefix as a basis to decide excludes in a finite number of steps. This procedure is formally defined later in this section.*

*The maximal path of $L$ with the occurrence of $t_2$ ends in a deadlock so it is maximal in the* mmg*-tree as well. It is clear that after the occurrence of $t_2$, $t_1$ cannot occur. Now we need to check if $t_2$ can occur after $t_1$ does. We can extend the paths by gluing the nodes corresponding to the same marking. In this way, we can build maximal paths of* mmg*-tree. For example, in the maximal path in which $t_1, t_3, t_4, t_6$, and $t_8$ occur (let's call it $\pi_1$), the blue nodes correspond to a repeated marking, namely $\{p_2, p_3\}$. So, $\pi_1$ can be extended in many different ways by gluing the blue nodes and repeating the gluing operation on different repeated marking pairs. It can be extended to a maximal path in which only $t_1, t_3, t_4, t_6$, and $t_8$, occur. It can be extended so as to also have occurrences of either $t_5$ or $t_7$ or both. But none of the continuations can have an occurrence of $t_2$ because the marking which enables $t_2$ can never be reached during this operation. So, we can conclude $t_1$ excludes $t_2$.*

*Now let us consider transitions $t_6$ and $t_8$. We can easily see that there is a maximal path of $L$ in which both occur, namely $\pi_1$. So we can decide $t_6$ does not exclude $t_8$ on the full prefix, without any further computation.*

*If we look at the transitions $t_5$ and $t_7$, we can see that the maximal path of $L$, in which $t_5$ occurs, ends in a node corresponding to the marking $\{p_4, p_5\}$, which is repeated in the path. So the path can be extended so that after $t_5$ occurs, $t_7$ does as well. This path witnesses that $t_5$ does not exclude $t_7$.*

In the following we present a finite algorithmic procedure to check whether $t_1$ excludes $t_2$ on the full prefix of the mmg-tree; the pseudocode of this procedure is presented in Algorithm 1. Theorem 4 proves the correctness of the algorithm.

Algorithm 1 takes the full prefix $\mathrm{fp}(\mathrm{mmg}(\Sigma)) = L$ and two transitions $t_1, t_2 \in T$ as input. During its execution, it checks whether there can be an occurrence of $t_2$ following or being in the same label as an occurrence of $t_1$, in any path of the mmg-tree. If this is the case, the algorithm returns false, and the procedure terminates. Otherwise, it returns true. In this second case, in order to determine whether $t_1$ ***ex*** $t_2$, Algorithm 1 must be repeated by exchanging the roles of $t_1$ and $t_2$, so as to check the case in which an occurrence of $t_1$ follows an occurrence of $t_2$. If the algorithm also returns true in this case, we conclude that $t_1$ ***ex*** $t_2$.

Next, we discuss the pseudocode as presented in Algorithm 1. The reasoning is analogous when the roles of $t_1$ and $t_2$ are exchanged.

First, the algorithm checks whether there is any maximal path in the full prefix $L$ with both $t_1$ and $t_2$. If $t_1$ does not exclude $t_2$ on the prefix, then it will certainly not exclude it on the whole tree. Otherwise, we compute the set *sup* of nodes following an occurrence of $t_1$ and such that for each $v_i \in sup$, there is no $v_j < v_i$ following an occurrence of $t_1$. Until *sup* is empty, the function *extract* removes an arbitrary element from *sup* and assigns it to $x$. For each $x$, we consider the set *leaves*$(x)$ of the leaves in $L_x$, and we compute the minimum element of the set $(\{\text{rep}(l) : l \in \text{leaves}(x) \setminus \text{checked}\} \cup \{x\})$. This element, denoted by $m$ is unique, as proved in Lemma 10. If the returned node $m$ is different from $x$, then we check whether $t_2$ occurs in $L_m$. If it does, $t_1$ does not exclude $t_2$. Otherwise, we remove from *sup* all the nodes $i$ such that $m < i$, and we mark all the leaves over $x$ as checked. In the next steps we will not need to visit these leaves again. When *sup* is empty we can conclude that there cannot be any occurrence of $t_2$ following an occurrence of $t_1$.

**Remark 11.** *As it may be clear form the discussion above, Algorithm 1 can be easily adapted to check whether, given two transitions $t_1, t_2$, $t_1$ $\boldsymbol{ex}_f$ $t_2$, and $t_1$ $\boldsymbol{ex}_p$ $t_2$. To check if $t_1$ $\boldsymbol{ex}_f$ $t_2$, we need to modify lines 2-4 in Algorithm 1, to check whether there is any occurrence of $t_2$ following an occurrence of $t_1$ in the prefix, if this the case, the algorithm must return false, otherwise, it proceeds from line 5 on. Unlike for the general excludes, in this case we must not exchange the role of $t_1$ and $t_2$. As observed in Remark 3, to check if $t_1$ $\boldsymbol{ex}_p$ $t_2$, we can equivalently check if $t_2$ $\boldsymbol{ex}_f$ $t_1$.*

**Lemma 10.** *Let $x$ be a node in $L$, and $S$ a set of leaves in $L_x$. The minimum element of the set $\{\text{rep}(v) \mid v \in S\} \cup \{x\}$ is unique.*

*Proof.* Let $r$ be the root of $L$, and for each pair of nodes $v_1, v_2 \in L$, such that $v_1 < v_2$, let $\pi_{v_1,v_2}$ be the path starting from $v_1$ and ending with $v_2$ in $L$. Let $v' \in \{\text{rep}(v) \mid v \in S\}$, then $x \leq v'$ or $x > v'$: for each $v \in S$, $\text{rep}(v)$ must be in $\pi_{r,v}$, and $\pi_{r,v} \subseteq \pi_{r,x} \cup L_x$. So, either $v' \in \pi_{r,x}$, or $x \leq v'$. Note that $\pi_{r,x}$ is totally ordered, since it is the initial segment of a path. Then any unordered pair $v_1, v_2 \in \{\text{rep}(v) \mid v \in S\}$ must satisfy $x \leq v_1$ and $x \leq v_2$. Since for any subset $R$, the elements of $\min(R)$ are unordered, we obtain that either $v \in \min\{\text{rep}(v) \mid v \in S\}$ is unique, or $\forall v \in \min\{\text{rep}(v) \mid v \in S\} : x \leq v$, which concludes the proof. □

**Lemma 11.** *The function described in Algorithm 1 terminates.*

*Proof.* If $\Pi_L^{t_1} \cap \Pi_L^{t_2} \neq \emptyset$ then the algorithm immediately terminates. Else, the function terminates when the set *sup* is empty. Let $n$ be the number of leaves in $L$, then initially $n_0 = |sup| \leq n$. At every iteration, at least one element $x$ of *sup* is removed, and possibly an element $i < x$ is added. Hence, after a general iteration $|sup| \leq n_0$. For each $x \in sup$, the number of elements $i : i < x$ is finite, therefore after a finite number of steps there

**Algorithm 1** Computing excludes

```
function EX((L: full prefix, $t_1, t_2 \in T$)) $\in$ {true, false}
    if $\Pi_L^{t_1} \cap \Pi_L^{t_2} \neq \emptyset$ then
        return false
    end if
    sup $\leftarrow$ min($\{v_i \in L : v_i$ follows an occurrence of $t_1\}$)
    checked $\leftarrow \emptyset$
    while sup $\neq \emptyset$ do
        $x \leftarrow$ extract(sup)
        $m \leftarrow$ min($\{$rep$(l) \mid l \in$leaves$(x) \setminus$ checked$\} \cup \{x\}$)
        if $m \neq x$ then
            sup $\leftarrow$ sup $\cup\{m\}$
            if $t_2 \in L_m$ then
                return false
            end if
            for $i \in$ sup do
                if $m < i$ then
                    sup $\leftarrow$ sup $\setminus\{i\}$
                end if
            end for
        end if
        checked $\leftarrow$ checked $\cup$ leaves($x$)
    end while
    return true
end function
```

is no element left that can be added to $sup$, and $k = |sup| \leq n_0$. When this point is reached, after at most $k$ iteration the function terminates. $\square$

**Theorem 4.** *Let $\Sigma = (P, T, F, W, m_{in})$ be a bounded equal-conflict P/T system, $t_1, t_2 \in T$, $A = \mathrm{mmg}(\Sigma)$-tree, $L = \mathrm{fp}(\mathrm{mmg}(\Sigma))$, and* EX *be the function of Algorithm 1. Then $\Pi_A^{t_1} \cap \Pi_A^{t_2} = \emptyset$ iff $\textsc{ex}(L, t_1, t_2) = \mathrm{true}$ and $\textsc{ex}(L, t_2, t_1) = \mathrm{true}$.*

*Proof.* The function $\textsc{ex}(L, t_1, t_2)$ checks whether some path of $L$ containing an occurrence of $t_1$ can be extended to a path of $A$ containing an occurrence of $t_2$, and returns false in that case. It is then required to check $\textsc{ex}(L, t_2, t_1)$ symmetrically, to ensure that no path of $A$ contains both $t_1$ and $t_2$.

We first assume that either $\textsc{ex}(L, t_1, t_2)$ or $\textsc{ex}(L, t_2, t_1)$ return false. If $\Pi_L^{t_1} \cap \Pi_L^{t_2} \neq \emptyset$, then $\Pi_A^{t_1} \cap \Pi_A^{t_2} \neq \emptyset$ as an immediate consequence of Lemma 6.

We assume that $\Pi_L^{t_1} \cap \Pi_L^{t_2} = \emptyset$, and we can suppose, without loss of generality, that $\textsc{ex}(L, t_1, t_2) = \mathrm{false}$, the other case being symmetric with respect to $t_1$ and $t_2$. Since $\Pi_L^{t_1} \cap \Pi_L^{t_2} = \emptyset$, there must be an $m$ such that $t_2 \in L_m$.

At every iteration of the *while* loop, the initial segment of a path of $A$ is considered, with at least an occurrence of $t_1$ and such that its final node is isomorphic to $m$. It then considers all its possible extensions with segments isomorphic to those in $L_m$. Only if some of these extensions include $t_2$ the algorithm returns false, therefore some path of $A$ has an initial segment containing occurrences of both $t_1$ and $t_2$, so $\Pi_A^{t_1} \cap \Pi_A^{t_2} \neq \emptyset$.

We now assume that $\pi \in \Pi_A^{t_1} \cap \Pi_A^{t_2} \neq \emptyset$, and that the algorithm returns true, to derive a contradiction.

If $t_1$ and $t_2$ occur in $\pi_{|L}$, the algorithm returns false in the first conditional statement. We consider the case in which $t_1$ or $t_2$ do not occur in $\pi_{|L}$. As a consequence, and by Lemma 7, there cannot be any arc in $\pi$ with occurrences of both $t_1$ and $t_2$. Without loss of generality, we can assume that the first occurrence of $t_1$ precedes the first occurrence of $t_2$ in $\pi$. We may also suppose that $t_1$ occurs in $\pi_{|L}$. Otherwise, we can successively peel $\pi$ until $t_1$ occurs in the prefix. We also assume that there are $n$ nodes between $v_0 = \mathrm{leaf}(\pi_{|L})$ and the first occurrence of $t_2$. Let $\pi_0 = \pi_{|L}$, and suppose that for $0 \leq j$, $\pi_j$ is an initial segment of $\pi$ with an occurrence of $t_1$, but none of $t_2$. Suppose that $\mathrm{leaf}(\pi_j)$ is isomorphic to some leaf $v_j$ of $L$, and note that if it were a deadlock, we would have $\pi_j = \pi$, so $t_2$ would occur in $\pi_j$. Then $\mathrm{rep}(v_j)$ must be well-defined. Furthermore, by Lemma 1, $L_{\mathrm{rep}(v_j)}$ is isomorphic to some prefix of $A_{\mathrm{leaf}(\pi_j)}$, so we may define $\pi_{j+1}$ to be the initial segment of $\pi$, obtained by extending $\pi_j$ with a maximal path of $L_{\mathrm{rep}(v_j)}$. Then $\pi_{j+1}$ still contains an occurrence of $t_1$, and its final node is isomorphic to some leaf $v_{j+1}$ of $L$. Next, we show that $t_2$ cannot occur in $\pi_{j+1}$.

Let $m$ be as in Algorithm 1, it follows from its definition, and Lemma 10, that $L_{\mathrm{rep}(v_j)} \subseteq L_m$. Since by assumption, the algorithm does not return false, there is no occurrence of $t_2$ in $L_m$, and so no occurrence of $t_2$ can appear in $L_{\mathrm{rep}(v_j)}$. Hence $t_2$ cannot occur $\pi_{j+1}$. Let $k \geq n$, we just showed by induction that $t_2$ does not occur in $\pi_k$, but it must have at least $n$ nodes after $v_0$, and since it is a prefix of $\pi$, it must have an occurrence of $t_2$. This is a contradiction, so either $\Pi_A^{t_1} \cap \Pi_A^{t_2} = \emptyset$ or the algorithm returns false. $\square$

In the next section, we will present a more general algorithm, that also allows for checking the excludes relation. In order to justify that Algorithm 1 is in general a better option, we here provide a means of comparing the two solutions.

During the first *if*, Algorithm 1 analyses all the arcs of the tree exactly once. In the *while* loop, the algorithm crosses each arc at most once. Indeed, an appropriate implementation of the condition $t_2 \in L_m$ would not require to check the arcs of a sub-tree that has already been analysed. As discussed above, in order to determine whether $a$ ***ex*** $b$, the *while* loop must be run twice, therefore the algorithm will check $|L|$ arcs and so the complexity of determining whether a given pair of transitions excludes each other is $O(|L|)$. In order to determine excludes for all the pairs of transitions, this analysis must be repeated for all the $|T| * |T - 1|$ pairs of distinct transitions, therefore the complexity is $O(|L| * |T|^2)$. Note that in information-flow analysis, we are only interested in excludes relation between low and high transitions, not all pairs of transitions.

It will be shown, in the next section that Algorithm 2 would require, in the same conditions, to perform $O(|L|^2 * 2^{|T|})$ visits to the nodes of $L$. Therefore, it is in general more convenient to use Algorithm 1 when only the excludes relation needs to be checked.

### 3.3.4 Computing footprint-sets

In this section, we present an algorithm that computes the set (but not the multiset) of transitions occurring in each maximal run on the unfolding of a bounded equal-conflict P/T system, by using the full prefix of the mmg-tree. This algorithm allows to check all the relations between sets of transitions, in which how many times each transition has occurred is not required. In particular, it can compute all the relations defined in Sec. 3.1.1. Although it may be less efficient than the algorithms presented so far in computing reveals or an excludes relation, through its output we can check all the extended-reveals and excludes relations we are interested in, with relatively few additional operations.

Given a P/T system $\Sigma$, the footprint of a maximal configuration $C$ of $\textsc{unf}(\Sigma)$ is the multiset of transitions that can be observed through the corresponding run, i.e., $\lambda(C)$. Thanks to the bijection between maximal configurations of the unfolding and maximal paths of the mmg($\Sigma$)-tree, we know that if $\Sigma$ is a bounded equal-conflict net, for each maximal run of the unfolding we can find a maximal path on the mmg-tree with the same footprint. This equivalence holds in particular when we consider sets of transitions instead of multisets, namely, when we ignore the number of occurrences of each transitions, and we just register its occurrence in the maximal path or in the maximal run.

We define *footprint-set* of a path $\pi$, and we denote it $\lambda_s(\pi)$, the set of transitions in the footprint of $\pi$, namely, $t \in \lambda_s(\pi)$ if, and only if, $t \in \lambda(\pi)$. Analogously, let $C$ be a configuration on $\textsc{unf}(\Sigma)$, the *footprint-set* of $C$, denoted with $\lambda_s(C)$, is the set of transitions of $\Sigma$ such that $t \in \lambda_s(C)$ if, and only if, there is an event $e \in C$ with $\lambda(e) = t$. For what discussed in Sec. 2.4 and above, if $\Sigma$ is a bounded equal-conflict P/T system, for each maximal configuration $C$ on $\textsc{unf}(\Sigma)$ there must be a maximal path $\pi$ on the mmg-tree such that $\lambda_s(\pi) = \lambda_s(C)$, and vice versa.

**Example 29.** *Consider the P/T system in Fig. 3.7, and the full prefix of its* mmg*-tree on the right of the same figure. The footprint-sets of some of the maximal configurations of* $\textsc{unf}(\Sigma)$ *can be found by looking at the maximal paths on the full prefix, These are* $\{t_2\}$, $\{t_1, t_3, t_4, t_7\}$, $\{t_1, t_3, t_4, t_5\}$, $\{t_1, t_3, t_4, t_6, t_8\}$. *However, in* $\textsc{unf}(\Sigma)$ *there are also maximal configurations with a different footprint-set. For example, the net could execute the cycle with* $t_4$ *and* $t_5$ *several times and then fire* $t_7$*; in this case the footprint-set would be* $\{t_1, t_3, t_4, t_5, t_7\}$. *Footprint-sets of other maximal configurations are* $\{t_1, t_3, t_4, t_6, t_7, t_8\}$, $\{t_1, t_3, t_4, t_5, t_6, t_8\}$, $\{t_1, t_3, t_4, t_5, t_6, t_7, t_8\}$.

The footprint-sets of all the maximal paths of a mmg-tree can be computed from its full prefix. This is made possible by two key features. First, any path of the mmg-tree can be reconstructed solely with the information provided by its full prefix. This is enabled by the following properties: (a) all reachable markings of a mmg-tree are represented in the full prefix; (b) each maximal path of a full prefix which does not end in a deadlock ends in a node corresponding to a marking which is repeated in that path; and (c) the subtrees of the mmg-tree whose roots correspond to the same marking are isomorphic. Second, since the set of labels of the system is finite, there are only finitely many possible footprint-sets that can occur in runs. Then only a finite collection of maximal paths needs to be explored, making sure that each possible footprint-set is represented in the collection. Furthermore, only a finite prefix of each path needs to be explored, if we make sure that it displays all labels that will eventually occur along the path. These notions are formalised in the following lemma.

**Lemma 12.** *Let* $\Sigma = (P, T, F, W, m_{in})$ *be a bounded equal-conflict P/T system,* $A$ *be the* $\text{mmg}(\Sigma)$*-tree,* $r$ *be its root, and* $\pi$ *be a finite prefix of some maximal path of* $A$. *Consider the final node* $v$ *of* $\pi$, *and suppose there is some other node* $v'$ *in* $\pi$ *associated to the same marking and such that* $\lambda_s(\pi) = \lambda_s(\pi_{r,v'})$. *Then the following hold.*

1. *There is a maximal path $\pi'$ of $A$ such that $\lambda_s(\pi) = \lambda_s(\pi')$, and $\pi$ is a prefix of $\pi'$.*

2. *For each maximal path $\pi_v$ such that $v \in \pi_v$, and $\pi_v \neq \pi'$ of $A$, there is a maximal path $\pi_{v'}$ of $A$ such that $v' \in \pi_{v'}$,$v \notin \pi_{v'}$, and $\lambda_s(\pi_v) = \lambda_s(\pi_{v'})$.*

*Proof.* Let $s_0 = v' {\uparrow} \pi$, so that $\pi = (\pi_{r,v'}) \cdot s_0$. Then $\lambda_s(\pi) = \lambda_s(\pi_{r,v'}) \cup \lambda_s(s_0)$, and since $\lambda_s(\pi) = \lambda_s(\pi_{r,v'})$, we derive that $\lambda_s(s_0) \subseteq \lambda_s(\pi_{r,v'})$.

In order to show (1), suppose that for some $k \in \mathbb{N}$ there is a maximal path $\pi_k$ of $A$, and a sequence of segments $\{s_i\}_{i \leq k}$ such that $\forall i \leq k : s_i \simeq s_0$, and $\pi_{r,v'} \cdot s_0 \cdots s_k$ is a prefix of $\pi_k$. In particular, this is true for $k = 0$. Since all the $s_i$ are isomorphic, and can be concatenated, for each of them the initial node $v_i$ must be associated to the same marking of the final nodes $v_{i+1}$. In particular, this holds for $i = k$, and it follows from Lemma 1 that $A_{v_k} \simeq A_{v_{k+1}}$. Put $v_0 = v'$ and $v_1 = v$, clearly $\forall i \leq k+1 : \lambda(\pi_{r,v_i}) = \lambda(\pi_{r,v_0})$ and $A_{v_i} \simeq A_{v_0}$. Note that $v_k \uparrow \pi_k$ is a maximal path of $A_{v_k}$, and that $s_k$ is its prefix. Then there must be a maximal path $\pi'_k$ in $A_{v_{k+1}}$ isomorphic to $v_k \uparrow \pi_k$, with a prefix $s_{k+1}$ isomorphic to $s_k$, and therefore also to $s_0$. Then $\pi_{k+1} = (\pi_{r,v_{k+1}} \cdot \pi'_k$ is a maximal path of $A$ with prefix $(\pi_{r,v'}) \cdot s_0 \cdots s_{k+1}$. By induction, we may derive that there is a sequence of isomorphic segments $\{s_i\}_{i \in \mathbb{N}}$, and a path $\pi' = (\pi \downarrow \gamma') \cdot s_0 \cdots s_k \cdots$, maximal in $A$. Furthermore, since the segments $s_k$ are all isomorphic, we have that $\forall k \in \mathbb{N} : \lambda_s(s_k) = \lambda_s(s_0) \subseteq \lambda_s(\pi_{r,v})$. Hence, $\lambda_s(\pi') = \lambda_s(\pi_{r,v})$.

This setting also allows one to show (2). Consider a maximal path $\pi_v \neq \pi'$ of $A$, containing $v$. There must be some $k \in \mathbb{N}$ such that $v_k \in \pi_v$, but $v_{k+1} \notin \pi_v$. But $A_{v_k} \simeq A_{v'}$ implies that $v_k \uparrow \pi_v$ is isomorphic to some maximal path $\pi'_{v'}$ of $A_{v'}$. Let $\pi_{v'} = (\pi_{r,v'}) \cdot \pi'_{v'}$. The node $v$ cannot belong to $\pi_{v'}$, since otherwise we would have that $v_{k+1} \in \pi_v$. Now note that since $\forall k \in \mathbb{N} : \lambda_s(s_k) = \lambda_s(s_0) \subseteq \lambda_s(v)$ , then $\lambda_s(\pi_v) = \lambda_s(\pi_{r,v'}) \cup \lambda_s(v_k \uparrow \pi_v)$. Since $\lambda_s(v_k \uparrow \pi_v) = \lambda_s(\pi'_{v'})$, we have that $\lambda_s(\pi_v) = \lambda_s(\pi_{v'})$. □

This lemma provides a stop condition for an exploration of the mmg-tree $A$, with the aim of finding footprints. Implication (2) provides a bound to the number of maximal paths that need to be effectively visited, whereas (1) lets us know when to stop exploring a path.

Algorithm 2 simulates such an exploration of $A$, in recursive depth first, by successively exploiting Lemma 1 to explore sub-trees of the full prefix $L$.

In Algorithm 2, we suppose that the full prefix $L$ is implemented thanks to the structure NODE. Aside from the field *children*, standard in a tree structure, each node $v$ gathers the following information.

- The marking corresponding to the node $v$, denoted $v.m$.

- The set of transitions labelling the arc bringing to the parent of $v$ to $v$, denoted $v.U$.

- When $v$ is a leaf of the prefix which is not a deadlock, $\mathrm{rep}(v)$ is a well-defined node of the prefix, and $v$.ancestor stores a pointer to that node.
- The field $v$.seen is required for Algorithm 2 to terminate. It shall be initialised to $\emptyset$ and will accumulate a collection of partial footprint-sets.

Before proving the correctness of Algorithm 2, we provide an intuition with a partial simulation of how it works on an example.

**Example 30.** *Consider the P/T system $\Sigma$ and the full prefix $\mathrm{fp}(\mathrm{mmg}(\Sigma))$ in Fig. 3.7*

*The algorithm to compute the footprint-sets starts from the root and at each step, if the node is not a leaf, it makes recursive calls on the children of the considered node.*

*In the tree in Fig. 3.7, the root has two children nodes. Then the execution of function* F *on the root $\{p_0\}$ with $c$ initialized to the empty set, returns as value the union of the results of the recursive calls on the child labelled with $\{p_1\}$ and on the one labelled with $\{p_2, p_3\}$.*

*Since $\{p_1\}$ is a deadlock, the first call returns as 'value' $\{\{t_2\}\}$, indeed there is a maximal path on the* mmg*-tree whose footprint-set is $\{t_2\}$.*

*The call on the node labelled with $\{p_2, p_3\}$, with $c = \{t_1\}$, induces a further call of* F *on the only child node labelled $\{p_4, p_5\}$ with $c = \{t_1, t_3, t_4\}$, namely the footprint leading from the root to this node. This call will return as 'value' the union of the values returned by the further calls of* F *on the children of node with label $\{p_4, p_5\}$. Inside this union, we focus only on the result of the call $F(v, \{t_1, t_3, t_4, t_6\})$, with $v.m = \{p_4, p_6\}$. Node $v$ has only one child, then it returns the value of the recursive call on the node $v'$ with label $\{p_2, p_3\}$ with $c = \{t_1, t_3, t_4, t_6, t_8\}$. The node $v'$ is a leaf, and it is not a deadlock; at this step '$v'$.seen' empty until this point, gets the value $\{\{t_1, t_3, t_4, t_6, t_8\}\}$, and '$v'$.ancestor' is rep($l_1$) which has label $\{p_2, p_3\}$. Then, the for cycle inside the second if call the function* F$(v'', \{t_1, t_3, t_4, t_6, t_8\})$*, where $v''$ is labelled $\{p_4, p_5\}$. This last call returns as value the union of the recursive applications of* F *on the children of $v''$, with $c$ updated by the information on the arc leading to the considered child.*

*The call* F$(l_3, \{t_1, t_3, t_4, t_6, t_7, t_8\})$*, with $l_3$ with label $\{p_4, p_7\}$ returns as 'value' $\{\{t_1, t_3, t_4, t_6, t_7, t_8\}\}$, since $\{p_4, p_7\}$ is a deadlock and $\{t_1, t_3, t_4, t_6, t_7, t_8\}$ is a footprint-set ending in $l_3$.*

*The result of the call of* F *on the node with label $\{p_4, p_6\}$ induces the call of* F$(l_1, \{t_1, t_3, t_4, t_6, t_8\})$*, where $l_1$ is labelled $\{p_2, p_3\}$.*

*The node $l_1$ is a leaf, not a deadlock and $l_1$.seen now contains c. Then, the else branch is executed, and the value $\{\{t_1, t_3, t_4, t_6, t_8\}\}$ is returned. The set $\{t_1, t_3, t_4, t_6, t_8\}$ is indeed an other possible footprint-set.*

*The application of* F *to the next nodes will lead in an analogous way to further recursive calls that terminate when the footprint-set c is already present in the field 'seen' of the considered leaf that is not a deadlock.*

**Algorithm 2** Computing footprint-sets

```
struct NODE $v$
    $m \subseteq P$ : marking in $v$
    $U \subseteq T$: set of transitions on the arc connecting $v$ and its parent
    seen $\subseteq 2^T$ : set of footprint-sets that has been considered
    ancestor : Node rep($v$)
    children : set of children of $v$
end struct

function F($v$ :Node, $c \subseteq T$) $\subseteq 2^T$
    value $\leftarrow \emptyset$
    if $v$.children $= \emptyset$ then
        if $v.m \notin$ deadlocks($\Sigma$) & $c \notin v$.seen then
            $v$.seen $\leftarrow v$.seen $\cup \{c\}$
            for $v_i \in v$.ancestor.children do
                value $\leftarrow$ value $\cup$ F$(v_i, c \cup v_i.U)$
            end for
        else
            value $\leftarrow \{c\}$
        end if
    else
        for $v_i \in v$.children do
            value $\leftarrow$ value $\cup$ F$(v_i, c \cup v_i.U)$
        end for
    end if
    return value
end function
```

The following theorem proves the correctness of Algorithm 2.

**Theorem 5.** *Let $A$ an* mmg*-tree, $L$ be its full prefix, and $r$ be the root of $L$. Let* F *be the function defined by Algorithm 2. Then* F$(r, \emptyset)$ *returns* $\{\lambda_s(\pi) \mid \pi \in \Pi_A\}$.

*Proof.* The function F is recursive and simulates a depth first exploration of $A$ on its full prefix $L$. In order to formalise this, we will say that an input pair of the algorithm $(v', c)$ simulates a node $v$ of $A$ whenever $v$ and $v'$ are associated to the same marking, and $c = \lambda_s(\pi_{r,v})$. Since $L$ is a prefix of $A$, out of Lemma 1, $A_{v'} \simeq A_v$ whenever $(v', c)$ simulates $v$, independently of $c$.

First, we show that for each call F$(v', c)$, there is a node $v$ simulated by $(v', c)$. Indeed, the initial call F$(r, \emptyset)$ satisfies that $\lambda(r) = \emptyset$. Now suppose that we enter the body of the function from a call $\textsc{f}(v', c)$ where $(v', c)$ simulates some $v$ of $A$. Further recursive calls are performed either if $v'$ is not a leaf of $L$, or when $v'$ is a leaf which is not a deadlock.

In the first case, it follows from $A_{v'} \simeq A_v$ that a node $v'_i$ of $L$ is a child of $v'$ if, and only if, $v$ has a child $v_i$ such that $v_i$ and $v'_i$ are associated to the same marking. Furthermore, if $U$ and $U'$ are the sets of transitions labelling the arcs $(v, v_i)$, and $(v', v'_i)$ respectively, then $U = U'$. Since each of these $v_i$ satisfies $\lambda_s(\pi_{r,v_i}) = \lambda_s(\pi_{r,v}) \cup U$, then $\lambda_s(\pi_{r,v'_i}) = \lambda_s(\pi_{r,v'}) \cup U' = c \cup v'_i.U$.

In the second case, we observe that $\text{rep}(v')$ is well-defined, and $v'$, $v$, and $\text{rep}(v)$ are associated to the same marking. Then the previous argument applies to $\text{rep}(v')$ instead of $v'$. It follows that each maximal path of $A$ is explored, up to simulation, and until the stop condition $c \in v'.\text{seen}$ holds.

Next, we show that each call $\textsc{f}(v', c)$ returns a set containing only footprint-sets of maximal paths of $A$. The function returns the footprint-sets of all maximal paths of $A$ which contain a node $v$ simulated by $(v', c)$, unless these footprint-sets are already returned at a previous call $\textsc{f}(v', c)$. In the function body, the only return statement concerns the variable value, which is set to $\emptyset$ at the beginning of the call. We may suppose that all recursive calls inside the body are effectively returning a set containing only valid footprint-sets.

We distinguish two cases, either $v'$ is a leaf of $L$ or not. If it is not, then the algorithm will enter a *for* loop, at each iteration of which the content of the variable value will be extended with whatever is returned by a recursive call to function $\textsc{f}$. Thus, by hypothesis, only valid footprint-sets are added to the contents of value.

If $v'$ is a leaf of $L$, which is not a deadlock, and such that $c \notin v'.\text{seen}$, then the same argument applies. If $c \in v'.\text{seen}$, variable value gets the singleton $\{c\}$. In this case, $c$ must have been added to $v'.\text{seen}$ in a previous call $\textsc{f}(v', c)$, and so there are two nodes $v_0$, and $v_1$ of $A$, both simulated by $(v', c)$.

Suppose there is a maximal path of $A$ which contains both $v_0$ and $v_1$, then we are in the conditions of Lemma 12. For the first condition, $c$ must be the footprint-set of some maximal path of $A$. Note that in this case, the simulated visit to $v_1$ must return before that to $v_0$ does, and so the values to be returned by the latter depend on those returned by the former. However, from the second condition we may deduce that if the footprint-set of a maximal path containing $v_1$ is different from $c$, then it is returned by the first call $\textsc{f}(\gamma', c)$, independently from the second: for each such path, there is a path with the same footprint-set containing $v_0$, but not $v_1$, so its footprint-set is computed without relying on the second call. Hence, $\{c\}$ is the only footprint-set to be returned by the second call, which is not redundant with those returned by the first.

If on the contrary, there is no maximal path of $A$ containing both $v_0$ and $v_1$, then by the second call, the first call $\textsc{f}(v', c)$ must have already returned and so there must have been an intermediary call $\textsc{f}(v', c)$ simulating a visit to a node $v_2$, such that there is a maximal path of $A$ containing both $v_0$ and $v_2$. Then conditions of Lemma 12 apply to $v_0$ and $v_2$, and by the previous argument, the first call has returned the footprint-sets of all maximal paths of $A$ containing $v_0$. Since by Lemma 1, $A_{v_0} \simeq A_{v_1}$, then a maximal path of $A$ containing $v_1$ has a given footprint-set if, and only if, $A$ has a maximal path containing $v_0$ with the same footprint-set. Furthermore, by concatenating infinitely many isomorphic copies of the segment with initial node rep$(v')$ and final node $v'$ to the path leading to $v_1$, we obtain a maximal path of $A$ whose footprint-set is precisely $c$.

The case in which $v'$ is a leaf of $L$ which is a deadlock remains. Since $(v', c)$ simulates some $v$ of $A$, and since $A_v \simeq A_{v'}$, then $v$ must be a deadlock as well. Hence $\{c\} = \lambda_s(\pi_{r,v})$ is the footprint-set of the finite maximal path ending at $v$.

Finally, since $L$ and $T$ are finite, there are only finitely many pairs $(v, c)$, with $v$ node in $L$ and $c$ subset of $T$. Only the first time a call $\textsc{f}(v, c)$ is performed, the size of the sub-tree of $A$ whose exploration is simulated increases, and always by at most the size of the full prefix. Then, if $w$ is the number of leaves of $L$ which are not deadlocks, the size of the explored sub-tree is at most $|L| + w * 2^{|T|} * |L|$. This value is an upper bound to the total number of calls to $\textsc{f}$, which ensures termination and provides a worst case asymptotic complexity $O(|L|^2 * 2^{|T|})$. □

This proof concludes that the footprint-sets of maximal runs of a system can be computed on its full prefix in a finite number of steps. This enables us to identify all the excludes and extended-reveals relations among transitions by looking for labels in the computed footprints.

### 3.3.5 Computing collective and extended-repeated reveals

In this section, we propose an algorithm to compute the collective reveals relation (Def. 16), on a bounded equal-conflict P/T system $\Sigma$, through the full prefix fp(mmg($\Sigma$)). Its pseudo-code is presented in Algorithm 3.

Given two sets of transitions of $\Sigma$ $X$ and $Y$, and a natural number $n > 0$, $X$ $n$-collective reveals $Y$ if, and only if, in each maximal configuration in $\textsc{unf}(\Sigma)$ with at least $n$ occurrences of transitions in $X$, there is at least an occurrence of a transition in $Y$.

Since reveals (Def. 9) and repeated reveals (Def. 14) can be expressed as special cases of collective reveals (see Remark 8), the algorithm allows to compute also these relations. At the end of the section, we will discuss how to modify it to compute also extended-repeated reveals (Def. 15).

Algorithm 3 takes as input the full prefix $L = \text{fp}(\text{mmg}(\Sigma))$, two sets of transitions, $X$ and $Y$, and a positive number $n$. If there is no path in the mmg-tree with at least $n$ occurrences of transitions of $X$, then the algorithm returns *undefined*. Otherwise, it returns *true* if $n.X \twoheadrightarrow Y$, *false* if $n.X \not\twoheadrightarrow Y$.

The main function of the algorithm is REPEX. The variable 'Paths' includes all the prefixes of the paths of the mmg-tree that we need to check to verify the relation. Initially, it includes all the paths in fp(mmg) with at least an occurrence of a transition of $X$ (this is justified by Lemma 8). The variable 'empty' is true if a path with at least $n$ occurrences of transitions of $X$ has not been found, false otherwise. 'Paths' and 'empty' are the input arguments of the function EXTENDPATHS. This function checks which input paths already include at least $n$ occurrences of transitions of $X$, and, if it finds a path with $n$ occurrences of $X$ and none of $Y$, it sets the value of 'stop' to true, and stops the execution. In this case, also the main function will stop, and return false, since the path could be extended to a maximal path of the mmg-tree without adding any new transition label. If the path has at least $n$ occurrences of $X$ and an occurrence in $Y$, then it does not need to be elongated further, and the algorithm must continue with the analysis of the other paths.

Let $\pi$ be a path with less than $n$ occurrences of $X$; then the algorithm needs to check which extensions could be useful to add more occurrences of transitions of $X$. If $\pi$ ends with a deadlock, it is not possible to extend it further, and since there are no $n$ occurrences of $X$ we are not interested in it. Otherwise, rep(leaf($\pi$)) is well defined, and all the paths extending $\pi$ are labelled as one of the paths starting from rep(leaf($\pi$)). Let *ext* be any path starting from rep(leaf($\pi$)). If *ext* does not have any occurrence of $X$ and ends with a deadlock, then we do not need to consider it, since the concatenation $\pi \cdot ext$ of $\pi$ and *ext* does not have $n$ occurrences of $X$. Also, we do not consider the extension of $\pi$ with no occurrence of $X$, and such that rep(leaf($\pi$)) $\leq$ rep(leaf(*ext*)), since each path with such a prefix and $n$ occurrences of $X$ can be peeled removing *ext* and still have $n$ occurrences of $X$. All the other extensions are put in the variable 'newPaths', and will be considered in the next round. If there are no paths left to analyse, and in all those with at least $n$ occurrences of $X$ there is an occurrence in $Y$, then the algorithm returns true.

**Example 31.** *Consider the P/T system in Fig. 3.6 and its full prefix. Assume that we want to check the relation $2.\{t_0, t_2\} \twoheadrightarrow \{t_1\}$. We can easily see that the relation is verified. The first part of the relation is satisfied if we observe $t_0$ or $t_2$ at least twice, or both $t_0$ and $t_2$ once. In all these cases, $t_1$ must have occurred to bring the token back in $p_0$. We simulate some steps of the algorithm to see how to arrive at this conclusion. First, we need to consider all the paths with at least an occurrence of $t_0$ or of $t_2$. In this case, these are all the paths of the full prefix. Since none of them has two occurrences, we need to check the possible extensions for all of them. The two paths ending in a deadlock cannot be extended further, therefore we can stop to consider them. As in previous parts, in what follows the occurrence of transition $t_i$ $n$ times in the multiset is denoted with $t_i^n$, and with just $t_i$ if*

**Algorithm 3** Computing collective reveals

```
function REPEX(($L$: full prefix, $X, Y \subseteq T, n \in \mathbf{N}$) $\in$ {true, false, undefined})
    Paths ← List of maximal paths in $L$ with at least an element of $X$
    empty ← true
    while Paths $\neq$ [] do
        Paths, empty, stop ← EXTENDPATHS(Paths, empty)
        if stop = true then
            return false
        end if
    end while
    if empty = true then
        return undefined
    else
        return true
    end if
end function

function EXTENDPATHS((Paths, empty))
    # returns a triple $(x, y, z)$, where $x$ is a list of paths, and $y$ and $z$ are boolean values
    newPaths = []
    for $\pi \in$ Paths do
        if $|\pi \cap X| \geq n$ then
            empty ← false
            if $Y \cap \pi = \emptyset$ then
                return [], false, true
            end if
        else if $l(\pi)$ is not a deadlock then
            for $ext \in L_{rep(l(\pi))}$ do
                if $ext \cap X \neq \emptyset \vee (rep(l(ext)) < rep(l(\pi)))$ then
                    newPaths.append($\pi \cdot ext$)
                end if
            end for
        end if
    end for
    return newPaths, empty, false
end function
# | $\pi \cap X$ | is equivalent to $\sum_{t \in X} \lambda(\pi)(t)$
```

*$n = 1$.*

*Consider the path with footprint $\{t_0, t_4, t_7^2, t_6, t_8\}$. Its possible extensions are isomorphic to the segments starting with $\mathrm{rep}(l_1)$: these have labels $\{t_5\}$ and $\{t_6, t_7\}$. In none of them an occurrence of $t_0$ or $t_2$ appears; the segment labelled $\{t_5\}$ ends in a deadlock, and the leaf of the segment labelled $\{t_6, t_7\}$ ($l_1$) coincides with $\mathrm{rep}(l_1)$, therefore none of these extensions is useful to observe a second occurrence in $\{a, c\}$ and we can discard the path. Analogously for the path with footprint $\{t_2, t_3, t_4, t_5, t_6, t_7^2\}$. A similar reasoning can be done for the paths with footprints $\{t_0, t_4, t_6, t_7\}$ and $\{t_2, t_3, t_4, t_6, t_7\}$: in these cases, the possible extensions start from the nodes $\mathrm{rep}(l_3)$ and $\mathrm{rep}(l_6)$ respectively, but none of them has an other occurrence of $t_0$ or $t_2$, and the repetition of the non-deadlock leaves coincides or follows these nodes. Hence, the only paths that we can extend are those ending with the nodes $l_8$ and $l_7$ ($\{t_0, t_1\}$ and $\{t_2, t_3, t_1\}$ respectively). The extensions of these paths start from the root, therefore in all of them there is an occurrence of $t_0$ or one of $t_2$, and these extended paths have two occurrences in $\{t_0, t_2\}$. Since in all of them there is already an occurrence of $t_1$, we can stop, and conclude $2.\{t_0, t_2\} \rightarrowtriangle \{t_1\}$.*

**Lemma 13.** *The leaf of every path constructed by the algorithm is a deadlock, or is associated with a marking that is already present in the path.*

*Proof.* First we observe that every path constructed by the algorithm ends with a node equivalent to a leaf in the prefix tree. For each leaf, the path in the tree starting from the root and arriving to it is unique.

Let $r$ be the root of the tree, $\pi_0 \cdot \pi'_1 \cdot \ldots \cdot \pi'_n$ be a path constructed by the algorithm, where $\pi_0$ is a maximal path in the prefix tree and $\pi'_i$ is an added segment isomorphic to a segment $\pi_i$ starting from $\mathrm{rep}(\mathrm{leaf}(\pi_{i-1}))$ (the repetition of the leaf $\mathrm{leaf}(\pi_{i-1})$ of the segment $\pi_{i-1}$) and ending in $\mathrm{leaf}(\pi_i)$, a leaf of the prefix tree. We have to prove that, if the leaf of the constructed path is not a deadlock, then the repetition $\mathrm{rep}(\mathrm{leaf}(\pi'_n))$ of the leaf $\mathrm{leaf}(\pi'_n)$ of the path belongs to the path itself, i.e., $\mathrm{rep}(\mathrm{leaf}(\pi'_n))$ is in $\pi_0 \cdot \pi'_1 \cdot \ldots \cdot \pi'_n$.

We prove it by induction. Let $\mathrm{leaf}(\pi'_1)$ be the leaf of $\pi_0 \cdot \pi'_1$; if it is not a deadlock, then $\mathrm{rep}(\mathrm{leaf}(\pi'_1))$ is either in $\pi'_1$ or inside $\pi_{r,\mathrm{rep}(\mathrm{leaf}(\pi_0))}$, where $r$ is the root. In this last case, it is contained in $\pi_0$. Then $\mathrm{rep}(\mathrm{leaf}(\pi'_1)) \in \pi_0 \cdot \pi'_1$.

We now assume the constructed path $\pi_0 \cdot \pi'_1 \cdot \ldots \cdot \pi'_i$ ends either with a deadlock, or with a node whose repetition $\mathrm{rep}(\mathrm{leaf}(\pi'_i))$ is either in $\pi'_i$ or in the segment $\pi_{r,\mathrm{rep}(\mathrm{leaf}(\pi'_{i-1}))}$, which is contained in $\pi_0 \pi'_1 \ldots \pi'_{i-1}$, and then $\mathrm{rep}(l(\pi'_i)) \in \pi_0 \cdot \pi'_1 \cdot \ldots \cdot \pi'_i$.

We prove that the path $\pi_0 \cdot \pi'_1 \cdot \ldots \cdot \pi'_{i+1}$ ends either with a deadlock, or with a node whose repetition $\mathrm{rep}(\mathrm{leaf}(\pi'_{i+1}))$ is either in $\pi'_{i+1}$ or in the segment $\pi r, \mathrm{rep}(l(\pi'_i))$, which is contained in $\pi_0 \cdot \pi'_1 \cdot \ldots \cdot \pi'_i$, and therefore $\mathrm{rep}(l(\pi'_{i+1})) \in \pi_0 \cdot \pi'_1 \cdot \ldots \cdot \pi'_{i+1}$. In fact, $\pi'_{i+1}$ is isomorphic to a segment $\pi_{i+1}$ in the prefix tree starting from $\mathrm{rep}(\mathrm{leaf}(\pi_i))$, which is between $r$ and $\mathrm{leaf}(\pi_i)$, and ending in a leaf leaf$(\pi_{i+1})$ of the prefix tree. This last leaf is either a deadlock, or has a repetition, which is either in $\pi_{i+1}$ or in $\pi r, \text{rep}(l(\pi_i))$; since rep(leaf$(\pi'_i)$) is by inductive hypothesis in $\pi_0 \cdot \pi'_1 \cdot ... \cdot \pi'_i$, we get the thesis. □

**Lemma 14.** *Let $\pi$ be any maximal path in the* mmg*-tree. If $\pi$ has in total at least $n$ occurrences of transitions belonging to $X$, then there is at least a path $\pi'$ analysed by the algorithm with in total at least $n$ occurrences of transitions of $X$, such that $\lambda(\pi') \subseteq \lambda(\pi)$.*

*Proof.* We show that we can peel $\pi$ and obtain a maximal path of the mmg-tree such that its prefix is analysed. For Lemma 8, we can peel $\pi$ and obtain a path $\pi_1$ such that at least an occurrence of $X$ appears in its prefix in fp(mmg). Let $\pi'_1$ be such prefix. If $\pi'_1$ has $n$ occurrences of $X$, we don't need to proceed further; otherwise $\pi'_1$ must be followed in $\pi_1$ by a path isomorphic to a path starting from rep(leaf$(\pi'_1)$). Let $\pi'_2$ be this segment. If $\pi'_2$ has at least an occurrence of a transition of $X$, or rep(leaf$(\pi'_2)$) precedes rep(leaf$(\pi'_1)$), this elongation of the prefix has been considered by the algorithm. If $\pi'_2$ has no occurrences of $X$ and rep(leaf$(\pi'_2)$) $\geq$ rep(leaf$(\pi'_1)$), then we can peel $\pi_1$ of the part between rep(leaf$(\pi'_2)$) and leaf$(\pi'_2)$, obtaining $\pi_1^2$. Since in $\pi'_2$ there are no elements of $X$, this cannot influence their number in $\pi_1^2$, and $\lambda(\pi_1^2) \subseteq \lambda(\pi_1)$. The path $\pi_1^2$ has also a prefix made by $\pi'_1$ concatenated with a segment starting from rep(leaf$(\pi'_1)$). Let $\pi'_3$ be this segment. If rep(leaf$(\pi'_3)$) $\geq$ rep(leaf$(\pi'_1)$), we repeat the peeling procedure. Since $\pi_1$ has $n$ occurrence of $X$ by hypothesis, after a finite number $i$ of steps, we will obtain a peeled maximal run $\pi_1^i$ such that rep(leaf$(\pi'_i)$) $<$ rep(leaf$(\pi'_1)$), with $\pi'_i$ segment starting from rep(leaf$(\pi'_1)$) extending the segment $\pi'_1$, or $\pi'_i$ has at least an occurrence of $X$. We can repeat this reasoning until obtaining a prefix with at least $n$ occurrences of $X$. Since in our steps we never remove any of those, and $\pi$ includes them by hypothesis, this procedure ends after a finite number of steps. By construction, all the transitions in the constructed prefix are also in $\pi$, therefore we produced a prefix as required from the thesis. □

**Theorem 6.** *Algorithm 3 is correct.*

*Proof.* As first step we show that if the algorithm returns false, then $n.X \not\rhd Y$. The algorithm returns false if the variable 'stop' is true. The value of 'stop' is selected into the function EXTENDPATHS, and it is set to true if a path is found with $n$ occurrences of $X$ and none in $Y$. Each prefix is constructed so that the final leaf is a deadlock for the path, or it is repeated previously; in the first case the path is already maximal, in the second case, the path can be extended to a maximal path without adding any new transition by repeating infinitely often the segment between the leaf of the prefix and its repetition. The existence of such a repetition is guaranteed by

Lemma 13. In both cases there is a maximal run with at least $n$ occurrences of $X$ and none in $Y$, therefore $n.X \not\rightarrowtriangle Y$.

We now show that if the algorithm returns true, then $n.X \rightarrowtriangle Y$. This is a consequence of Lemma 14: if there were a path with $n$ occurrences of $X$ and none in $Y$, Lemma 14 guarantees that we would analyse a prefix with the same feature, but if this happens, the algorithm returns false.

If the algorithm returns undefined, then there cannot be any run in the mmg-tree with $n$ occurrences of transitions of $X$ as a consequence of Lemma 14. □

**Theorem 7.** *Algorithm 3 terminates.*

*Proof.* The algorithm ends when the variable ‘Paths’ becomes empty, or when a path with $n$ occurrences of $X$ and none in $Y$ was found. We show that ‘Paths’ becomes empty after a finite number of steps. The variable ‘Paths’ is a list of prefixes of paths in the mmg-tree. Its content in each iteration of the while loop is entirely determined by the function EXTENDPATHS, that extends some of its elements. In particular, the function extends the paths with less of $n$ occurrences of $X$. Each path can be extended in two ways: adding at least an additional occurrence of $X$, and this happens only finitely many times, since when the path has at least $n$ occurrences of $X$ it is not extended anymore, or with a segment whose repetition of the leaf is strictly closer to the root than the repetition of the previous leaf. Also in this second case the number of extension is finite, since the distance between each node and the root is finite. □

**Remark 12.** *When we consider a reveals relation, e.g. $t_1 \triangleright t_2$, this algorithm needs to analyse only the maximal runs of the prefix tree, without further extensions. This is coherent with the result in Sec. 3.3.2, where it is shown that all the reveals relations can be computed by looking one time at the prefix.*

Algorithm 3 can be adapted to compute the extended-repeated reveals relation (Def. 15). Here we give a sketch of how this can be done. Let $X = \{t_1, ..., t_k\}$ and $Y$ be the input sets. Instead of having just a single threshold $n$ in input as for repeated reveals, the input must include all the thresholds $\{n_1, ..., n_k\}$ related to transitions of $X$, and the information about how they are associated to these transitions. Since the number of observations in which we are interested changes for every transition, when the algorithm needs to decide whether a path can stop or needs to be extended, it must consider all transitions of $X$ separately, each with its threshold. In addition, if a path has already reached the number of required occurrences of a certain transition, we should stop to consider this transition as useful when we evaluate the possible extensions.

Since extended reveals (Def. 10) can be expressed as a special case of extended-repeated reveals, modifying the algorithm as described would allow for its computation.

Unlike the algorithm proposed in Sec. 3.3.4, the algorithm proposed in this section is designed to check a specific instance of the collective reveals or extended-repeated reveals relations.

### 3.3.6 Toward more efficient algorithms: a reduction of the prefix

In this section we describe an algorithm that, given a relation based on reveals and a bounded equal-conflict P/T system, computes a reduced version RMG of the mmg, and uses an adapted version of the algorithm presented in Sec. 3.3.5 on the full prefix of the RMG-tree to compute the relation.

We describe in details the case in which the given relation is an instance of collective reveals, but the result can be adapted to the case of extended-repeated reveals with an analogous reasoning to the one presented at the end of Sec. 3.3.5, and therefore also to the other relations based on reveals.

Let $\Sigma = (P, T, F, W, m_{in})$ be a bounded equal-conflict P/T system, $X, Y \subseteq T$, and $n \geq 1$, and assume that we need to verify $n.X \rightarrowtriangle Y$. The relation $n.X \not\rightarrowtriangle Y$ holds if, and only if, there is a maximal configuration in $\textsc{unf}(\Sigma)$ with $n$ occurrences of $X$ and none of $Y$. Whereas finding a configuration with $n$ occurrences of $X$ and at least an occurrence of $Y$ does not give us any information, finding a configuration with $n$ occurrences of $X$ and none of $Y$ is sufficient to conclude that the relation does not hold. For this reason, while checking collective reveals, we can consider only the paths with no occurrence of $Y$, and check if there is one with at least $n$ occurrences of $X$ among them.

Note that in this way we do not detect the case in which the relation is not defined. However, it is reasonable to assume that in practical cases, users may be interested in verifying a relation which they already know to be defined on the system. If this is not the case, we need to use the algorithm presented in Sec. 3.3.5.

The procedure presented in this section reduce mmg($\Sigma$) by keeping only the sequences of maximal steps without occurrences of $Y$, and then check on them whether $n.X \not\rightarrowtriangle Y$. We denote such a reduction of mmg with respect to $Y$ as $\text{RMG}_Y(\Sigma)$, or just $\text{RMG}_Y$ if $\Sigma$ is clear from the context.

Given $\Sigma$ and $Y$, the construction of $\text{RMG}_Y$ is presented in Algorithm 4 and Algorithm 5. The construction start with the function $\textsc{rmg}$ in Algorithm 4. During its execution, the algorithm constructs a reduction of mmg, where the arcs labelled with an occurrence of $Y$ have been removed. At the same time, it stores in *'remove'* the list of markings that are not deadlock in $\Sigma$, but have no outgoing arc in the reduction of mmg. The paths crossing these markings need to be removed from the final result, since a sequence ending on them would not be associated to any maximal sequence in mmg. This further reduction is carried out by function UPDATE_RMG in Algorithm 5, that takes the list *'remove'* and the reduced marking graph constructed so far as input. This is a recursive function that removes from the marking graph all the arcs having a marking in *'remove'* as final point, and constructs another list of markings (*'new_bad'*) that have no outgoing arcs at the end of the function. The function calls recursively itself until the list of nodes to remove is empty. This must happen at some point, since the number of markings is finite, and once that a marking has been removed, it can never be added again.

**Algorithm 4** Computing the reduction $\text{RMG}_Y$ of mmg($\Sigma$)

```
function RMG($\Sigma$: net, $Y \subseteq T$)
    pending ← [$m_0$]
    RMG.markings = {$\Sigma.m_0$}
    RMG.trans ← $\emptyset$
    remove ← []
    while pending ≠ [] do
        $m$ ←pending.pop()
        steps ← compute_max_steps($m$)
        if steps = $\emptyset$ then
            new_dead ← False
        else
            new_dead ← True
        end if
        for step ∈ steps do
            if step ∩ $Y = \emptyset$ then
                new_dead ← False
                m_next ← compute_next_mrk($m$, step)
                if m_next ∉ RMG.markings then:
                    RMG.markings.add(m_next)
                    pending.append(next_mrk)
                end if
                RMG.trans.add(($m$, step, m_next))
            end if
        end for
        if new_dead = True then
            remove.append($m$)
        end if
    end while
    return update_RMG(RMG, remove)
end function
```

**Algorithm 5** Function removing deadlocks generated from the reduction

```
function UPDATE_RMG(RMG, remove)
    if remove = [] then:
        return RMG
    end if
    new_bad ← []
    for $m \in$ remove do
        RMG.markings.remove($m$)
        for $t \in$ RMG.trans do
            if $t[2] = m$ then
                RMG.trans.remove($t$)
            end if
            if $\nexists t' \in$ RMG.trans : $t'[0] \leftarrow t[0]$ then
                new_bad ← $t[0]$
            end if
        end for
    end for
    return update_RMG(RMG, new_bad)
end function
```

**Example 32.** *Fig. 3.8 recalls the P/T system introduced in Fig. 2.5 and its* mmg*, and shows two reduction of the* mmg*, in particular the first reduction on the left is made on the set* $\{t_1\}$ *and the second one is made on the set* $\{t_7\}$*. Considering set* $\{t_1\}$*, only one arc is removed from the original* mmg*, since there is only one arc labelled with* $\mathrm{t}_1$*. Since this arc arrives in the node labelled* $\mathrm{p}_0, p_5^2$*, that is still reachable in the* $\mathrm{RMG}_{\{t_1\}}$ *(being the initial node) and has two outgoing arcs, no other element is removed from* $\mathrm{RMG}_{\{t_1\}}$*. Instead, when we consider the set* $\{t_7\}$*, the size of* $\mathrm{RMG}_{\{t_7\}}$ *is much smaller than the one of* mmg*. This happens also because Algorithm 4 does not only remove all the arcs labelled with* $\mathrm{t}_7$ *from* mmg*, but also the nodes that are not deadlocks in* mmg*, but have no outgoing arcs after removing the occurrences of* $\mathrm{t}_7$ *from* mmg*, and the arcs pointing to these nodes. An example of this, is the node labelled* $\{p_3, p_4, p_5^2\}$*, that is removed although it is reachable without occurrences of* $\mathrm{t}_7$*, since its only outgoing arc is labelled with* $\mathrm{t}_7$*. The graph resulting from this procedure, exactly produces all the maximal runs of the P/T system which do not have any occurrence of* $\mathrm{t}_7$*.*

Given a reduction $\mathrm{RMG}_Y$ of an mmg, the tree-unfolding of $\mathrm{RMG}_Y$ (Def. 4) and its full prefix $\mathrm{fp}(\mathrm{RMG}_Y)$ (Def. 17) are well-defined. Since each maximal path in the $\mathrm{RMG}_Y$-tree are also paths in the mmg-tree, each maximal path in the $\mathrm{RMG}_Y$-tree can be associated to at least one maximal configuration of the unfolding, with the same footprint. The vice versa does not hold for the $\mathrm{RMG}_Y$-tree, since all the paths with an occurrence of $Y$ have been removed; still, the vice versa holds when we consider only the

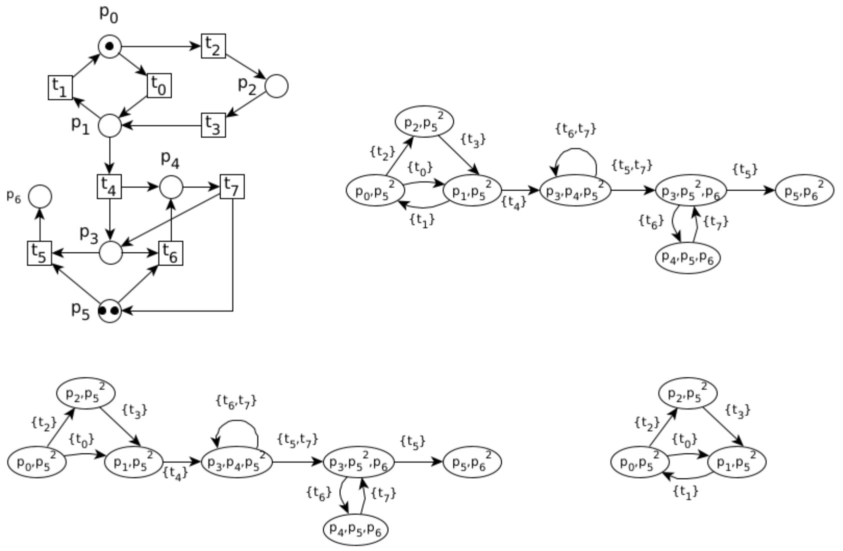


Figure 3.8: Above, the P/T system introduced in Fig. 2.5 and its mmg. Below, two reduction of the mmg, on the left with respect to the set $Y = \{t_1\}$, and on the right with respect to the set $Y = \{t_7\}$.

maximal configurations on the unfolding without any occurrence of $Y$.

**Remark 13.** *If* $Y = \emptyset$*,* mmg *coincides with* $\mathrm{RMG}_Y$*, therefore, if a property holds on* $\mathrm{RMG}_Y$*-tree for any* $Y$*, it holds also on* mmg*-tree.*

**Example 33.** *Fig. 3.9 illustrates the full prefix of the* $\mathrm{mmg}(\Sigma)$*-tree, the full prefix of the* $\mathrm{RMG}_{\{t_1\}}(\Sigma)$*-tree, and the full prefix of the* $\mathrm{RMG}_{\{t_7\}}(\Sigma)$*-tree, where* $\Sigma$ *is the system net illustrated in Fig. 3.8.*

Let $\Sigma = (P, T, F, W, m_{in})$ be a bounded equal-conflict P/T system, $X, Y \subseteq T$, $n \in \mathbb{N}$. To check whether the relation $n.X \twoheadrightarrow Y$ on $\Sigma$, we compute $\mathrm{fp}(\mathrm{RMG}_Y)$, and we use the adapted version of Algorithm 3 presented in Algorithm 6, in which the elements of $Y$ are not considered (since they are not present in $\mathrm{fp}(\mathrm{RMG}_Y)$).

**Example 34.** *Consider the P/T system* $\Sigma$ *in Fig. 3.8. As in Ex. 31, assume that we want to check the relation* $2.\{t_0, t_2\} \twoheadrightarrow \{t_1\}$*, and that we knows that the relation is well defined. Given the P/T system* $\Sigma$ *and the set* $\{t_1\}$*, Algorithm 4 computes the reduced marking graph* $\mathrm{RMG}_{\{t_1\}}(\Sigma)$ *which is also illustrated in Fig. 3.8 (below on the left). This procedure results in a marking graph which only produces the maximal runs without occurrences of* $\{t_1\}$*. Then the full prefix of the reduced graph* $\mathrm{fp}(\mathrm{RMG}_{\{t_1\}}(\Sigma))$ *is constructed as defined in Def. 17 and is illustrated in Fig. 3.9 (below on the left). Algorithm 6 takes* $\mathrm{fp}(\mathrm{RMG}_{\{t_1\}}(\Sigma))$*, the set* $\{t_0, t_2\}$ *and number* $2$ *as input. When*

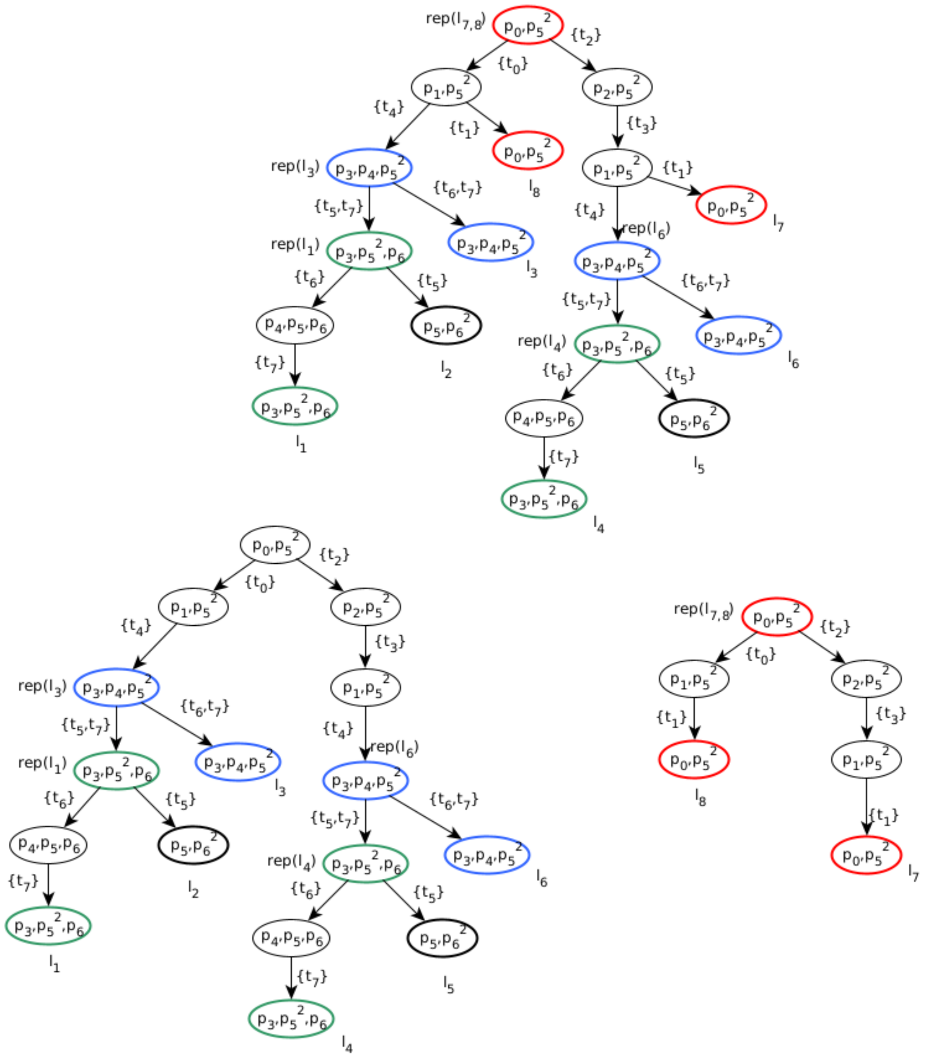


Figure 3.9: Above, the full prefix of the mmg-tree of the P/T system in Fig. 3.8. Below, the full prefixes fp($\mathrm{RMG}_{\{t_1\}}$) on the left, and the full prefix fp($\mathrm{RMG}_{\{t_7\}}$) on the right.

**Algorithm 6** Computing collective reveals through $\mathrm{RMG}_Y$

```
function REPEX($L : \mathrm{fp}(\mathrm{RMG}_Y), X \subseteq T, n \in \mathbf{N}$)
    Paths = The list of maximal paths in $L$ with at least an element of $X$
    while Paths $\neq$ [] do
        Paths, stop $\leftarrow$ EXTENDPATHS(Paths)
        if stop = True then
            return False
        end if
    end while
    return True
end function

function EXTENDPATHS(Paths)
    # returns a pair $(x, y)$, where $x$ is a list of paths, and $y$ is a boolean value
    newPaths $\leftarrow$ []
    for $\pi \in$ Paths do
        if $|\pi \cap X| \geq n$ then
            return [], True
        else if $l(\pi)$ is not a deadlock then
            for $ext \in L_{rep(l(\pi))}$ do
                if $ext \cap X \neq \emptyset \vee (rep(l(ext)) < rep(l(\pi)))$ then
                    newPaths.append($\pi$ + $ext$)
                end if
            end for
        end if
    end for
    return newPaths, False
end function
```

*the algorithm runs, it detects that there is no maximal path with at least two occurrences of* $\{t_0, t_2\}$*. Then* $2.\{t_0, t_2\} \twoheadrightarrow \{t_1\}$*.*

*Now let us assume that we want to check the relation* $t_5 \rhd t_7$ *which is equivalent to* $1.\{t_5\} \twoheadrightarrow \{t_7\}$*. We can follow the same procedure as above. Given the net system* $\Sigma$ *and the set* $\{t_7\}$*, Algorithm 4 produces the reduced marking graph* $\mathrm{RMG}_{\{t_7\}}(\Sigma)$ *which is also illustrated in Fig. 3.8 (below on the right). The full prefix* $\mathrm{fp}(\mathrm{RMG}_{\{t_7\}}(\Sigma))$ *is illustrated in Fig. 3.9 (below on the right). Algorithm 6 takes* $\mathrm{fp}(\mathrm{RMG}_{\{t_7\}}(\Sigma))$*, the set* $\{t_5\}$ *and number* $1$ *as input. When the algorithm runs, it detects that there is no maximal path with at least an occurrence of* $t_5$*. Then,* $t_5 \rhd t_7$*.*

*Lastly, let us consider the relation* $2.\{t_0, t_2\} \twoheadrightarrow \{t_7\}$*. This time we call Algorithm 3 with* $\mathrm{fp}(\mathrm{RMG}_{\{t_7\}}(\Sigma))$*, the set* $\{t_0, t_2\}$ *and number* $2$ *as input. The algorithm finds a maximal run such that* $t_7$ *does not occur but the number of occurrences of* $\{t_0, t_2\}$ *is* $2$*. In fact, the prefix of* $\mathrm{RMG}_{\{t_7\}}(\Sigma)$*-tree illustrated in Fig. 3.9 can be extended to produce many of such maximal runs, e.g., the maximal run which repeats the sequence* $t_0t_1t_2t_3t_0t_1$*. With these given inputs, Algorithm 6 returns* false *meaning that* $2.\{t_0, t_2\} \not\twoheadrightarrow \{t_7\}$*.*

### 3.3.7 Experiments

We implemented the algorithms in Sec. 3.3.2, Sec. 3.3.5, and Sec. 3.3.6, and tested them on some examples. This section describes the results obtained in these experiments. Since the contribution of this chapter and of the papers from which it is derived ([6, 1, 7]) are mainly theoretical, we did not focus on the optimization of the code and we did not carry out a systematic set of experiments; this is left as future work. However, I think that the experiments and considerations presented in this section can be useful to get some insights on the algorithm efficiency and to suggest directions for future improvements.

Our tool takes as input a P/T system specified in the PNML format [130], and has the following usage options.

1. The user can specify on the command line a collective reveals relation $n.X \twoheadrightarrow Y$. In this case, the tool proceeds as described in Sec. 3.3.6: first it computes $\mathrm{RMG}_Y$, then the $\mathrm{RMG}_Y$-tree, and finally decides whether $n.X \twoheadrightarrow Y$, using the tree.

2. The user may also not specify any relation on the command line, and require the computation of mmg and the mmg-tree. In this case, after the computation of the mmg-tree, the user can specify any collective reveals relation in an interactive way, and the verification proceeds as in Algorithm 3.3.5.

3. Finally, the user may ask for the computation of all the (simple) reveals relations (Def. 9) on the tree. As in the previous case, this is done by

computing mmg and the mmg-tree, and by looking at the leaves of the tree. To increase the efficiency, instead of checking the relations one by one, for each leaf we check all the reveals relations that cannot hold, since they are denied by that path, and if a relation is not excluded by any leaf, then it must hold in the system. In this case, the code refers to the result in Sec. 3.3.2.

For our experiments we considered the following sets of P/T systems. (1) We combined in different ways the nets shown in Fig. 3.3 and Fig. 3.4 (rows pn1 through pn6 in the tables). (2) We developed a set of examples inspired by the Kanban P/T systems in [97], used as benchmark for the tool [117] (rows k33, k34, and k4 in the tables). (3) Again from [97], we tested two P/T systems describing industrial business process models [54] (rows IBM319 and IBM703); these are interesting models, since we expect business processes to be an application of the methods described in this chapter.

For each P/T system, we computed the number of reachable markings (column RMa); the number of markings in mmg (MM) and RMG$_Y$ (MMY); the number of leaves in the mmg-tree (L) and RMG$_Y$-tree for some set of transitions $Y$ (LY); the time needed for computing mmg (TMM); the time needed for computing the mmg-tree (Tt); the time for computing the RMG$_Y$-tree (TYt); the time for computing a collective reveals relation (Trev); the time to compute all the reveals relations (Tall). The results of our experiments are in Table 3.1 and Table 3.2. All times are given in seconds. The second column in Table 3.1 (#$\Sigma$) denotes the sum of places and transitions in the P/T system. Missing values denote cases in which we stopped the computation before termination, after at least an hour.

The PNML files for all these nets are available in the git repository at this link `https://github.com/MC3-lab/mscTree`, together with the tool.

From the results we can observe that in some cases, such as the sets of examples (1) and (3), using maximal-step semantics significantly reduces the number of markings to analyse. In the sets of examples (2) and (3), the number of leaves in the tree is larger than the number of nodes in mmg and RMG; this suggests that some of the constructed subtrees may be isomorphic, and therefore it would be possible to prune them. However, deciding how to prune the tree is not trivial, since two nodes in the prefix associated to the same marking but with a different past, may be the starting point of non-isomorphic subtrees in the prefix. Some criteria that we plan to test in future works consist in pruning the tree if it reaches a node whose marking has already been analysed and the set of markings preceding the repeated node includes all the marking in the past of the node already analysed, or in pruning the tree if we reach a node that has already been analysed, and that is not part of any cycle.

Set (2) is particularly interesting. These nets are critical for our algorithm and tool; the interplay between concurrency among components and

Table 3.1: Results of the experiments using mmg.

| Name | #Σ | RMa | MM | L | TMM | Tt | Trev | Tall |
|---|---|---|---|---|---|---|---|---|
| pn1 | 85 | 1289 | 172 | 44 | 0.244 | 0.018 | 0.001 | 0.384 |
| pn2 | 88 | 408630 | 1173 | 315 | 0.370 | 0.216 | 0.005 | 0.418 |
| pn3 | 91 | 2093213 | 3291 | 1494 | 0.953 | 1.150 | 0.012 | 0.511 |
| pn4 | 80 | 2252 | 408 | 182 | 0.267 | 0.101 | 0.002 | 0.397 |
| pn5 | 83 | 958197 | 3420 | 2656 | 0.970 | 4.529 | 0.021 | 0.613 |
| pn6 | 129 | $\sim 15 \cdot 10^6$ | 143341 | 132800 | 2622.647 | 374.301 | 5.997 | 424.937 |
| k33-f1 | 25 | 64 | 29 | 155 | 0.002 | 0.029 | 0.052 | 0.021 |
| k33-f2 | 25 | 64 | 29 | 155 | 0.002 | 0.029 | 0.001 | 0.021 |
| k34 | 25 | 160 | 129 | - | 0.012 | - | - | - |
| k4 | 32 | 160 | 49 | 3339 | 0.234 | 0.945 | 661.353 | 0.539 |
| IBM319 | 431 | 2482 | 325 | 1282 | 0.387 | 28.717 | 0.007 | 30.147 |
| IBM703 | 546 | 8370 | 732 | 1948 | 1.007 | 66.251 | 0.015 | 67.109 |

Table 3.2: Results of the experiments using $\mathrm{RMG}_Y$.

| Name | MMY | LY | TMMY | TYt | Trev |
|---|---|---|---|---|---|
| pn1 | 146 | 32 | 0.243 | 0.013 | $1.27 \cdot 10^{-4}$ |
| pn2 | 895 | 207 | 0.350 | 0.150 | 0.001 |
| pn3 | 2086 | 684 | 0.675 | 0.491 | 0.003 |
| pn4 | 357 | 169 | 0.259 | 0.0900 | $4.15 \cdot 10^{-4}$ |
| pn5 | 1089 | 498 | 0.357 | 0.418 | 0.002 |
| pn6 | 29946 | 18592 | 148.412 | 52.011 | 0.109 |
| k33-f1 | 3 | 1 | $2.98 \cdot 10^{-4}$ | $3.89 \cdot 10^{-5}$ | $8.58 \cdot 10^{-6}$ |
| k34-f2 | 23 | 40 | 0.002 | 0.012 | $9.44 \cdot 10^{-5}$ |
| k34-f1 | 3 | 1 | 0.001 | $3.98 \cdot 10^{-5}$ | $8.34 \cdot 10^{-6}$ |
| k34-f2 | 41 | 690 | 0.006 | 0.158 | 0.001 |
| k4 | 16 | 38 | 0.232 | 0.005 | $7.25 \cdot 10^{-5}$ |
| IBM319 | 281 | 610 | 0.230 | 13.801 | 0.002 |
| IBM703 | 714 | 1860 | 0.971 | 61.861 | 0.006 |

local conflicts has two effects: first, there is no relevant difference between the number of reachable markings in the full marking graph and in the reduced marking graph; second, the tree is very large, because many different combinations of maximal steps are possible, thus generating many branches, and long paths before returning to the same marking. Hence, for these nets, a small number of components can become unfeasible; the use of the reduction technique improves anyway the performance for these nets.

The reduction proposed in Sec. 3.3.6 seems to improve the performance in all the steps of the algorithm with respect to the technique proposed in Sec. 3.3.5, and in case of a large numbers of markings in mmg or in the mmg-tree, it can become very convenient, even if we need to compute a few reveals relations. The efficiency of the reduction of the mmg-tree strongly depends on the set $Y$; in some cases, removing the arcs with $Y$ allows to cut a huge number of nodes in the tree, whereas in other cases, the number of nodes is similar in the $\mathrm{RMG}_Y$-tree and in the mmg-tree.

## 3.4 Related works and application of the relations based on reveals and excludes

In this section, we discuss other notions developed in the literature for the analysis of information flow. In particular, we present a non-exhaustive overview of the works dealing with diagnosis, noninterference, and opacity in Petri nets and related formal models, and discuss some simple applications of the relations based on reveals and excludes in these contexts.

The problem of diagnosability consists in determining whether observing some behaviour on a system allows to determine that a fault has happened or will inevitably happen. The faults are usually unobservable (otherwise their identification would be trivial).

In the '90s, diagnosis was formalized on discrete event systems (DES) [113]. Several subsequent works extended the results in [113] on DES by proposing different notions of diagnosability, developing algorithms, and proposing applications. In [133] the authors propose an overview describing the main lines developed in this context. The techniques developed for diagnosis on Petri nets follow different approaches. In [34] the authors study diagnosis on labelled Petri nets, and propose a verification algorithm based on a reduction of the reachability graph. Their algorithms are then applied to a manufacturing system. The work in [47] proposes an online method for fault detection based on Integer Linear Programming. The algorithm waits for the observation of a transition and establish whether the behaviour of the system is normal, or a fault has occurred. The approach more related to ours is based on the unfolding of the Petri net [16, 70]. This is also the context in which the reveals relation was defined for the first time [70]. The use of reveals to model diagnosis was then developed in [12, 72, 71]. In [17] the authors compare several notions of diagnosis (and opacity) and determine their complexity. The works in [14, 90] give an overview of the methods used in diagnosis with Petri nets.

The concept of noninterference was introduced by Goguen and Meseguer for deterministic state machines in [66]. In [32] Busi and Gorrieri discusses several notions of noninterference defined on 1-safe system. In the study of noninterference, the transitions of the system are divided into high transitions and low transitions on the base of their confidentiality. A system satisfies a noninterference property if no information about high transitions can be deduced through the interaction with low transitions. Noninterference has applications in checking security and privacy requirements. In studying noninterference, in [33], the authors define structural noninterference properties for elementary nets based on absence of particular places in the net. Baldan and Carraro give a characterisation of noninterference based on unfoldings of 1-safe system in terms of causalities and conflicts in [13]. In [15], the authors provide an algorithm to compute all the minimal solutions for enforcing noninterference on bounded Petri nets by using linear integer programming techniques. The definitions of reveals and excludes used in this thesis were introduced in [21, 86]. Unlike the definition in [70], where reveals was defined on the events of an occurrence net, [21, 86] define reveals and excludes between transitions of a P/T system, and propose several noninterference properties based on these relations. The parametric relations defined in this chapter can contribute to define more general and tailored noninterference properties on the line of the definitions in [21].

The last notion that we discuss in this section is opacity. As for noninterferece, also in the study of opacity we assume that some parts of the system are observable, while other are not. Some part of the system should remain secret for a standard observer. The secret may be for example the value of a state or the occurrence of a transition. A system is opaque if for each run with the secret, there is another run without the secret such that the two runs are indistinguishable for an observer. There is a strong relation between noninterference and opacity. In [31] the authors study the relation between opacity and noninterference, and show that, for some formulations of the two notions, noninterferece implies opacity, while the opposite is not true. A first notion of opacity was introduced in [96] to define properties of security protocols. In [31] opacity is defined on transition systems. In [30] the authors provides a framework to study three definitions of opacity on weighted Petri nets. They assume that an observer knows the structure of the system, but may not observe the value of some places, therefore it cannot know the global state of the system, included the initial marking. In [122] the authors tackle the problem of enforcing opacity on a DES. They base their approach on supervisory control, developed for the first time in [109]: this consists in synthesising a supervisor limiting the behaviour of the system, so that all the remaining behaviours satisfy a certain property, for example opacity. An updated overview of methods used to study opacity in DES can be found in [68]. So far there is no work modelling opacity with reveals and excludes; however, in the following section we discuss an opacity problem with the methods presented in this chapter. In future works we plan to formalize the intuition provided by the example by using the relations defined in Sec. 3.1 to express opacity properties.

### 3.4.1 Examples of applications

We now discuss two simple examples to show possible applications of the relations and algorithms presented in this chapter. The example proposed in this section are also published in [7].

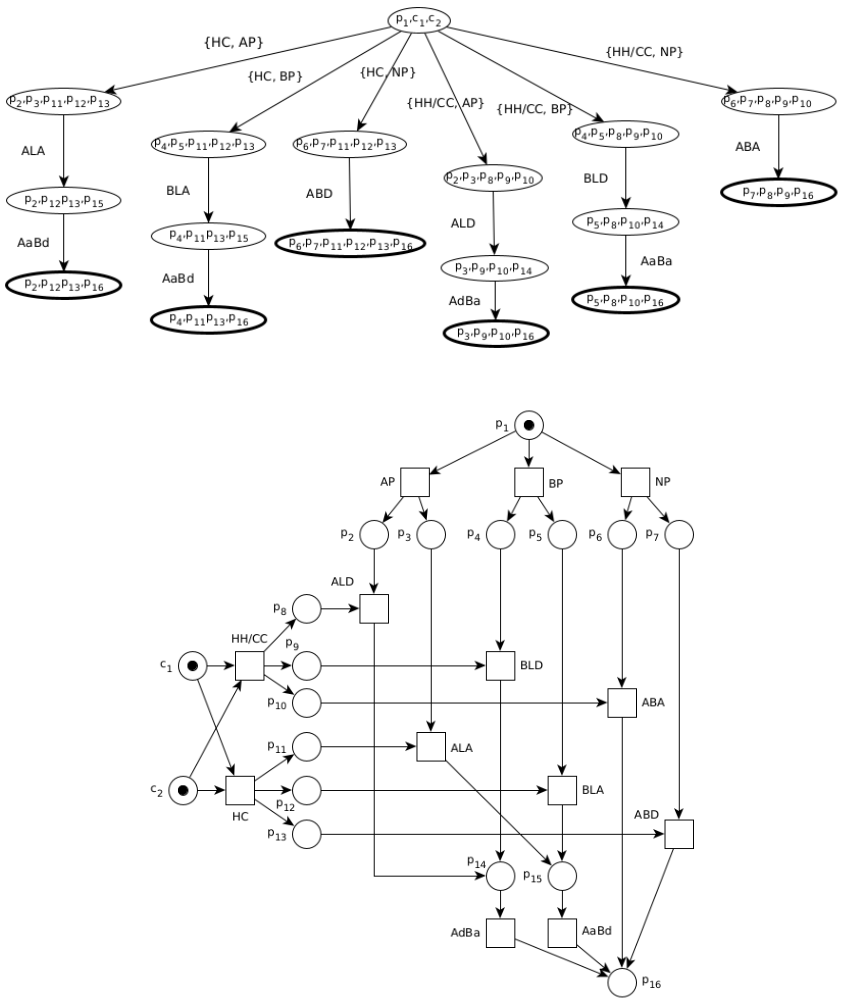
A first example related to business process model has been already introduced at the beginning of this chapter (see Fig. 3.1 and Ex. 18). The next example that we propose models a variant of the dining cryptographers protocol introduced in [30] and [96] in the context of opacity. We show how the scenario described in the example can be modelled with a free-choice net, and how the security requirements can be checked through the information flow relations previously introduced in the chapter.

**The dining cryptographers** The two cryptographers Anne and Bob enjoy a meal at the restaurant. When they ask for the bill, they are informed that it has already been paid. They want to know whether one of them or their employer National Security Agency (NSA) paid, but in case one of them paid, they do not want their neighbour Eve to discover who. To this aim, they perform a protocol to exchange information in a secure way. They toss two coins, visible to both of them, and they state their parity ('agree' if the coins show the same side, 'disagree' otherwise). Eve can hear whatever they say, but she cannot see the coins. If Anne paid, she will lie about the parity of the two coins, otherwise she will tell the truth. Bob will do the same. After this procedure Anne and Bob will know who paid the bill, whereas Eve will only be able to say whether one of them or the NSA paid. But if NSA did not pay, Eve will not be able to know who paid among Anne and Bob.

This protocol is modelled with the 1-safe free choice net in Fig. 3.10, where the mmg-tree of the net and the legend of the labels of each transition are also presented. Below we examine the net with our techniques.

**Example 35.** *The protocol must make sure (1) Eve is not able to determine who paid among Anne and Bob, (2) Anne and Bob are able to determine who paid.*

*In the protocol, Eve can observe $AdBa, AaBd, ABA$ and $ABD$. Everything else is hidden. We can verify the first requirement by checking whether the transitions that are observable to Eve reveal if Anne or Bob paid (AP*



| Legend of transitions | |
|---|---|
| AP/ BP/ NP | Anne/ Bob/ NSA paid |
| HH/CC | the coins show two heads or two crosses |
| HC | one coin shows head, the other cross |
| ALA/ BLA | Anne/ Bob lies stating that the two coins agree |
| ALD/ BLD | Anne/ Bob lies stating that the two coins disagree |
| ABA | Anne and Bob state that the coins agree |
| ABD | Anne and Bob state that the coins disagree |
| AdBa | Anne states that the two coins disagree, whereas Bob states they agree |
| AaBd | Anne states that the two coins agree, whereas Bob states they disagree |

Figure 3.10: The dining cryptographers net and the full prefix of its maximal step computation tree.

*and $BP$). As discussed in Sec. 3.3.2, by looking at the footprints of the maximal paths of the full prefix (which is actually the whole* mmg*-tree in this case), we see that none of the mentioned transitions reveals $AP$ or $BP$, e.g., $AdBa \not\triangleright AP$, $AdBa \not\triangleright BP$, etc. We see that no combination of the observable transitions appears in the same path, so it is not relevant to check whether any combination extended-reveals $AP$ or $BP$. We can conclude that the first requirement is satisfied.*

*The second requirement is a property that the protocol must satisfy to ensure the required information is exchanged between Anne and Bob. This property can be expressed in terms of extended-reveals relation and can be verified by computing the footprints of the* mmg*-tree as discussed in Section 3.3.4. However, in this example there are no cycles and so the full prefix is actually the whole* mmg*-tree. This means that further computation is not required. The set of footprints of maximal paths of the full prefix and the* mmg*-tree are the same. The protocol is finalised by one of the following transitions: $AdBa, AaBd, ABA, ABD$. All are observable. Nothing else is observable to Eve, but $AP$ is observable to Anne while $BP$ is observable to Bob. Looking at the footprints, we deduce that $ABA \triangleright NP$ and $ABD \triangleright NP$. We also see that $\{AdBa\} \rightarrowtriangle \{AP, BP\}$ and $\{AaBd\} \rightarrowtriangle \{AP, BP\}$. This means that by observing $AdBa$ or $AaBd$, the observer can deduce either Anne or Bob paid. So, while Eve is not able to understand who paid, when Bob pays, Anne understands and vice versa.*

As second example, we consider a simple modular representation of a client interacting with a group of providers, which could grow in size to express arbitrarily complex behaviours. We discuss system reliability and information-flow security in terms of reveals and excludes relations on a simple instance of this model.

**Client and providers** We model a client as a controller interacting with providers through some actions. Each action may send one or several service requests in parallel, each to a different service provider, and remains idle until all the services are performed. When all services are performed the action is concluded and control returns to the client. The client controller may implement an arbitrary program by sequentially determining which action to take.

Suppose that the client needs to perform an operation for which she requires data, and some remote computational power. She then needs to check the outcome against a data set from a different database. Fig. 3.11 displays a controller for such task. It accesses the database of Provider 1 by performing request $cr1$. If the query produces an error, the controller launches an emergency interrupt procedure. If the data is obtained, the firing of $cr23$ launches two requests in parallel to concurrently process the data on the remote cluster of Provider 2, and query the database of Provider

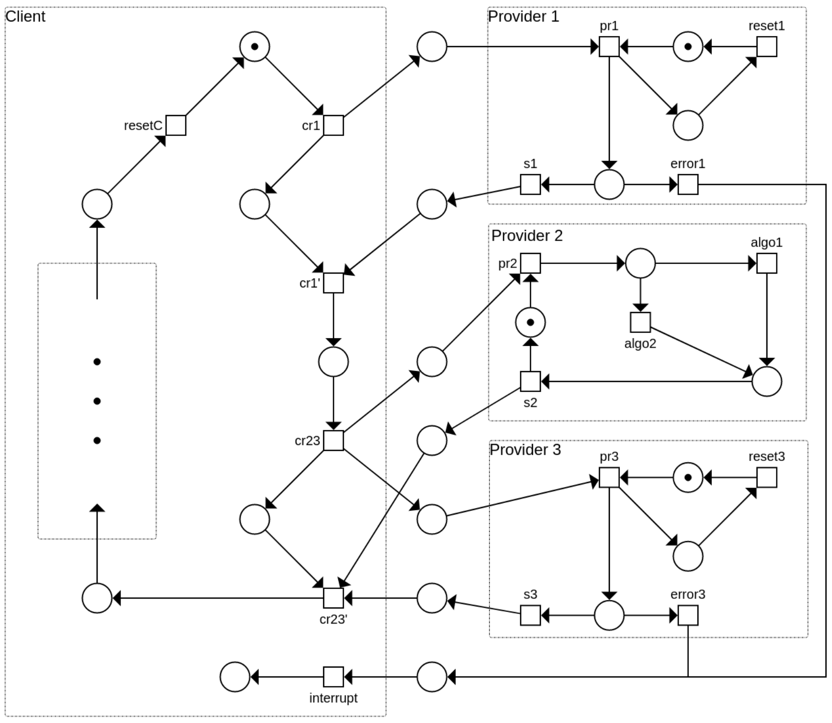


Figure 3.11: A client exchanging information with three providers.

3 for the second data set. If this second query does not produce an error, then $cr23'$ may fire, allowing the system to check the processed data against this data set, possibly by invoking yet other services, and continue with the execution of its arbitrary program, and service requests.

When a data provider receives the request, it simply processes it through $pr1$ or $pr3$, which sends a positive or negative reply and enables it to reset. Provider 2 is optimised so that depending on the request, it may choose to perform the operation with algorithms $algo1$, or $algo2$.

Below, we examine the model illustrated in Fig. 3.11 for system reliability and information-flow security by means of reveals and excludes relations. Note that, in this setting, client can observe the transitions that are within the client component (illustrated via the rectangle labelled "Client"). The transitions belonging to the providers are hidden from the client. And conversely, the providers can only observe their own transitions. In the below example, the observability of the transitions is relevant for the analysis of security (noninterference) properties.

**Example 36.** *In this model, transition $cr23'$ reveals transitions $s1$, $s2$, and $s3$, but does not reveal transitions algo1, algo2. In fact, no observable transition reveals algo1 or algo2. This can be interpreted as the fact that once* *the client has received all the data, she is sure that the three services have been provided, but she cannot decide which algorithm was chosen by Provider 2 to perform its task. Conversely, transition* $s1$ *reveals* $cr1$*, and* $s2$ *and* $s3$ *both reveal transition* $cr23$*; i.e., the providers send a reply only when the client sends them a request. In general, reveals transition doesn't imply any order, i.e., the revealed transition may occur before or after the revealing transition or even simultaneously with it. However, knowing the structure of the net, we can safely say that the mentioned reveals relations guarantee that the services are provided only when a request is made.*

*Note that* $cr1$ *does not reveal* $s1$*, since the request might lead to* $error1$*. Transition* $cr23$ *does not reveal* $s3$*, but it reveals* $s2$*, since Provider 2 always processes the data it receives. Also,* $error1$ *and* $error3$ *exclude each other, because the system stalls after interrupt.*

*These properties can easily be verified on the full prefix of the* mmg-*tree. Note that for all practical purpose, the undefined process in Fig. 3.11 may be abstracted as a single transition* $t$*. This transition represents all the transitions* $T$ *in the abstracted module. In this context, for a transition* $t' \notin T$*,* $t \rhd t'$ *can be interpreted as* $\exists t_0 \in T : t_0 \rhd t'$*, and conversely,* $t' \rhd t$ *as* $\exists t_0 \in T : t' \rhd t_0$*.*

*With this abstraction, the full prefix of the* mmg-*tree has five maximal paths. Three of these correspond to executions that end in a deadlock after the firing of interrupt, and have for respective footprints the sets* $\{cr1, pr1,$ *error1, reset1, interrupt*$\}$*,* $\{cr1, pr1,$ *reset1,* $s1, cr1', cr23, pr2,$ *algo1,*$s2, cr23,$ $pr3,$*error3, reset3, interrupt*$\}$*, and* $\{cr1, pr1,$ *reset1* $, s1, cr1', cr23, pr2,$ *algo2,* $s2, cr23, pr3,$ *error3, reset3, interrupt*$\}$*. Note that the last two footprints only differ in the choice of Provider 2 for the algorithm. This is also the case for the two remaining paths: each depicts all transitions in the system but for error1, error3, and interrupt in its footprint, and present exactly one of the labels among algo1 and algo2.*

*We may conclude that it is impossible, by observing the client, or the database providers, to determine whether Provider 2 used algorithm 1 or 2. On the other hand, the client may determine that Provider 1 did provide the requested service* $s1$*, by observing* $cr1', cr23, cr23'$*, or resetC. With analogous computations all the reveals relation can be derived, as for instance* $cr23 \rhd s1$*, but* $cr23 \not\rhd algo1$*.*

*Another property that we may want to check is the possibility of error. It is easy to see that error1* ***ex*** *error3, i.e., only one error can occur in the system that performs an emergency stop. However, by applying Algorithm 1 we can see that* $cr23'$ *does not exclude error1 (or error3), i.e., even when the three services were successfully provided once to the client, an error could still occur after the first resetC.*

# Chapter 4

# Control of a system

In this chapter, we analyse P/T systems in which a user controls a subset of transitions. Controlling a transition means that whenever it is enabled, the user can decide whether to fire it or not. The user can take decisions about the occurrence of its transitions based on its information about the current state of the P/T system, which is not necessarily complete, namely the user may not know the current global state. The goal of the user is to force the behaviour of the system so that every execution satisfies a certain property (e.g. reaching and/or avoiding some subsets of places).

We model the interaction of the user on the system as a two-player game, where the players are the *user* and the *environment*, which controls all the transitions uncontrollable for the user. The goal of the user is a property of the behaviour of the P/T system, and the user wins a *play* if such property is satisfied during the execution. The game is *asynchronous*: the two players can move whenever they have enabled transitions to fire, without any turn division. The user has a *winning strategy* if it is able to win every game, independently from the environment moves.

The ability of the user to win does not only depend on the structure of the system and on the transitions under its control, but also on the information that it can use to make its decisions, namely its observation of the current state and the memory of what already happened on the system. Sec. 4.1 gives the formal definitions of the games, discusses notions of observability and memory, and their relation with the concurrent semantics of Petri nets.

The rest of the chapter is organized as follows. Sec. 4.2 and Sec. 4.3 propose algorithms to check whether the user has a winning strategies in some special cases. In particular, in Sec. 4.2 we study how to use prefixes of the unfolding to find algorithms; in Sec. 4.3, we look for strategies on the marking graph, and compare our game with the one defined on *concurrent game structures* [9]. Sec. 4.4 tackles the problem of "correcting" the behaviour of a P/T system, constraining its behaviour so that only the executions that follows a given strategy are allowed in the corrected P/T system. Finally,

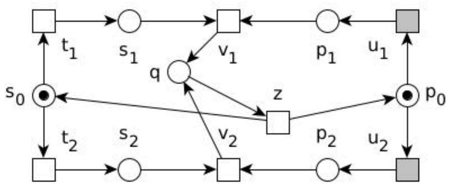


Figure 4.1: A game net

Sec. 4.5 discusses some related works.

The results discussed in this chapter are partially collected in [3, 2, 4, 5].

## 4.1 An asynchronous game on P/T systems

We introduce the game through an example. Consider the P/T system in Fig. 4.1. The user and the environment interact on it, the *user* by controlling the grey transitions, and the *environment* by controlling the white ones. Both players have enabled transitions, namely $t_1, t_2$ for the environment, and $u_1, u_2$ for the user; since the game is asynchronous, both of them could decide to fire them. In the game, we assume that there is a set $T_f$ of transitions bound to a progress constraint, namely they cannot be blocked forever once they are enabled. In this example, we assume that the progress constraint is on the transitions controlled by the environment, whereas the user has the right to keep its transitions blocked when they are enabled. We suppose that the user has full observation of the current marking, and that it has the goal of marking place $q$ infinitely often. In order to win, the user must wait for the environment to choose between firing $t_1$ or $t_2$. It can do it, since the environment cannot delay this choice forever. In the former case, the user chooses $u_1$, otherwise $u_2$. The environment is then forced to fire either $v_1$ or $v_2$, with the effect of marking $q$, and then to fire $z$, reproducing the initial marking.

Formally, we define a game on a bounded P/T system $\Sigma = (P, T, F, W, m_{in})$, where the transitions are partitioned into two sets: $T_u$, the *controllable* transitions, i.e. the ones controlled by the user, and $T_{env}$, the *uncontrollable* transitions, i.e. the ones controlled by the environment; $T = T_u \cup T_{env}$, $T_u \cap T_{env} = \emptyset$. Graphically, controllable transitions are coloured in grey, and uncontrollable transitions are white.

We consider the unfolding of the P/T system $\Sigma$, where events are partitioned into *controllable* ($E_u$) and *uncontrollable* ($E_{env}$) events, depending on their correspondence to occurrences of controllable or uncontrollable transitions, respectively.

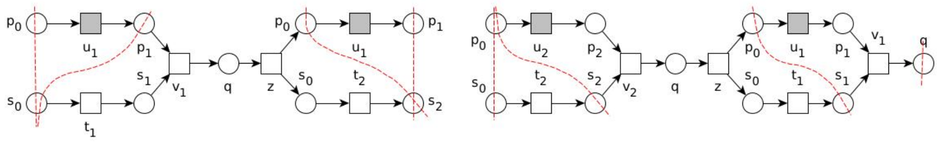


Figure 4.2: A terminating play (left) and a partial play (right)

**Definition 19.** *Let* $\Sigma = (P, T, F, W, m_{in})$ *a P/T system,* $\text{UNF}(\Sigma) = (B, E, F, \lambda)$ *its unfolding,* $T_f \subseteq T$ *be the subset of transitions that must satisfy a progress constraint, and* $E_f = \{e \in E : \lambda(e) \in T_f\}$. *A* play *is a pair* $(\rho, \delta)$, *where* $\rho = (B_\rho, E_\rho, F_\rho)$ *is a run in* $\text{UNF}(\Sigma)$ *and* $\delta$ *is an increasing sequence of B-cuts satisfying the following conditions.*

1. $\rho$ *is maximal with respect to the events in* $E_f$, *namely there cannot be any event* $e \in E_f$ *such that* ${}^\bullet e \in B_\rho$, *and* $({}^\bullet e)^\bullet \cap E_\rho = \emptyset$;

2. *for each event* $e \in E_\rho$, *there is a cut* $\gamma \in \delta$ *such that* $e < \gamma$.

Intuitively, the sequence of B-cuts can be seen as a record of some moments of the play taken by an external referee. This will be used in Sec. 4.3 to define the validity of an LTL formula on a given play.

The *winning condition* for the user is a set of plays, satisfying a certain property. The user wins a play if the play belongs to the winning condition. The goal of the user is to win all the plays.

**Example 37.** *Consider the net in Fig. 4.1, in which* $T_f = T_{env}$. *Fig. 4.2 shows a play on the left and a prefix of a play on the right. If the user needs to guarantee infinite occurrences of* $q$, *then it looses the play on the left, whereas it can win the play on the right.*

In order to win, the user can apply a *strategy*. We assume that the user has in general only a partial knowledge of the current state of the net system. This is formalized by a notion of *observation*. An observation is given by an equivalence relation on the set of B-cuts of the unfolding, denoted by $\equiv$, where two B-cuts are equivalent if they are indistinguishable for the user. Examples of observations are discussed in Sec. 4.1.1.

**Definition 20.** *A* strategy $\alpha$ *is a function from the set of observations, denoted by* $\mathsf{Obs}$, *to sets of controllable enabled transitions:* $\alpha : \mathsf{Obs} \to 2^{T_u}$.

When the user follows a strategy, it reduces the possible runs on the system by blocking certain behaviours. The following definition provides the formal conditions that a play needs to satisfy when the user follows a strategy.

**Definition 21.** *A play $(\rho, \delta)$ is* consistent *with a strategy $\alpha$ if the following conditions hold.*

1. *For each controllable event $e \in E_\rho \cap E_u$ there is a B-cut $\gamma \in \delta$ such that $\lambda(e) \in \alpha([\gamma])$ and $\lambda(\gamma)[\lambda(e)\rangle$, where $[\gamma]$ denotes the equivalence class w.r.t. $\equiv$ containing $\gamma$.*
2. *There is no event $e \in E_u$, $e \notin E_\rho$ such that there is a B-cut $\gamma_j \in \delta$ : $\lambda(e) \in \alpha([\gamma])$, for each B-cut $\gamma \geq \gamma_j$ compatible with $\delta$.*

Loosely speaking, a play is consistent with a strategy if each choice of the user in the play is justified by the strategy, and there is no event constantly enabled and constantly selected by the strategy which never fires.

A strategy $\alpha$ is *winning* if all the plays consistent with $\alpha$ are in the winning condition of the user.

**Example 38.** *In the P/T system $\Sigma$ in Fig. 4.1, assume that the user can observe every marking, namely given two B-cuts $\gamma_1, \gamma_2$ in $\text{UNF}(\Sigma)$, $\gamma_1 \equiv \gamma_2$ iff $\lambda(\gamma_1) = \lambda(\gamma_2)$. If the goal of the user is to reach $q$ infinitely often, and $T_f = T_{env}$, a winning strategy for the user is the following:*

- $\alpha(\{s_1, p_0\}) = \{u_1\}$*;*
- $\alpha(\{s_2, p_0\}) = \{u_2\}$*;*
- $\alpha(m) = \emptyset$ *for each* $m \notin \{\{s_1, p_0\}, \{s_2, p_0\}\}$*.*

### 4.1.1 Observability and memory

In Sec. 4.1, an observation is defined as an equivalence class of B-cuts on the unfolding of a P/T system. Determining the equivalence classes allows us to specify both the local states of which the user can observe the value, and the memory of the user during the game. This is because each condition on the unfolding carries the information about its past; hence, when we allow the user to discriminate such conditions, the strategy can use an information that we do not have when the user can only distinguish between places of the P/T system. In addition, the value of some local states may never be observable for the user, for example for security requirements or physical distance, or it can be visible only under specific circumstances.

In this section, we discuss different equivalent classes that could be considered as observations, showing what kind of observability and memory they imply for the user, and their relations with concurrency.

#### Observation of the global states

As first example, we consider the case in which all the local states are observable for the user, and we show that there is a difference if the user can

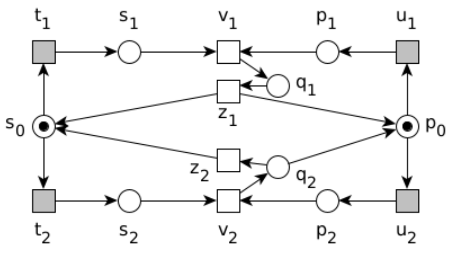


Figure 4.3: A game net in which memory is necessary to win.

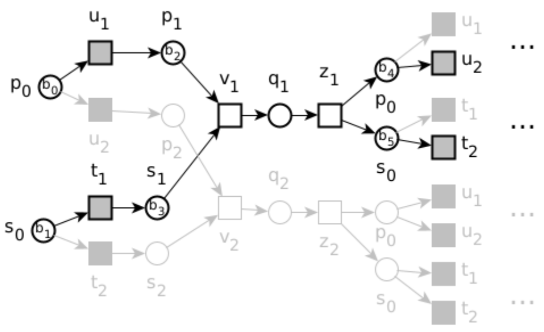


Figure 4.4: Prefix of the unfolding of the P/T system in Fig. 4.3, with a highlighted run.

discriminate between conditions on the unfolding, or it can only distinguish places on the P/T system (Ex. 39).

In the first case, the user has the greatest possible information. In this case, there is a bijection between B-cuts and equivalence classes, namely, for each pair of B-cuts $\gamma_1$, $\gamma_2$, $\gamma_1 \equiv \gamma_2$ iff $\gamma_1 = \gamma_2$, and the user has a perfect memory on the P/T system, that can use in its strategy.

The second case is the one that we considered in Ex. 38, where for each pairs of B-cuts $\gamma_1$ and $\gamma_2$, $\gamma_1 \equiv \gamma_2$ iff $\lambda(\gamma_1) = \lambda(\gamma_2)$.

**Example 39.** *To satisfy some properties, the information about the current marking is not sufficient. Fig. 4.3 represents a modified version of the P/T system in Fig. 4.1, where also $t_1$ and $t_2$ are controllable by the user. Assume that the user needs to guarantee that both $q_1$ and $q_2$ are reached infinitely often. The user has a winning strategy only if it has memory on the system.*

*Fig. 4.4 represents a prefix of the unfolding of the P/T system in Fig. 4.3. The run with black borders is the beginning of an execution where the user is winning. A winning strategy for the user realizing such a run could be $\alpha(\{b_0, b_1\}) = \{t_1, u_1\}$, $\alpha(\{b_0, b_3\}) = \{u_1\}$, $\alpha(\{b_1, b_2\}) = \{t_1\}$, $\alpha(\{b_4, b_5\}) = \{u_2, t_2\}$, and so on by alternating the occurrences of $u_1$ and $t_1$ with the occurrences of $u_2$ and $t_2$. This is not the only strategy for the user to win. However, no winning strategy exists when the user can discriminate only markings: if the user cannot remember how the initial marking was recreated, it could always choose the same pair of transitions, and one between $q_1$ and $q_2$ may occur only a finite number of times.*

Often, assuming the observability of all the local states is not realistic, therefore in the next equivalence classes we will present other scenarios. In addition to that, in some systems, it is not reasonable to consider strategies based on the global markings, as showed in the following example.

**Example 40.** *In the P/T system in Fig. 4.5, assume that everything is observable, and that the goal of the user is to reach one of the two markings $\{p_0, p_4\}$ or $\{p_1, p_3\}$. If the user could base its decision on the global states, then it could reach its goal by firing $t_2$ when the marking is $\{p_0, p_2\}$ or $t_3$ when the marking is $\{p_1, p_2\}$. However, even if the user observes an occurrence of $p_0$, it is in general unrealistic to assume that the communication arrives without time delay, and that it succeeds in firing $t_2$ before the environment can fire $t_0$.*

*According to the definition of play and of play consistent with a strategy (Def. 19 and Def. 21), the user does not have a winning strategy to reach this goal, since neither $t_2$ nor $t_3$ satisfy requirement (2) in Def. 21, and therefore a play without any occurrence of them is consistent with the strategy. Also a strategy selecting both $t_2$ and $t_3$ on each marking would not be winning, since the transition may fire in the wrong marking, and $t_4$ or $t_5$ may fire immediately after. Although this is a reasonable conclusion, we*

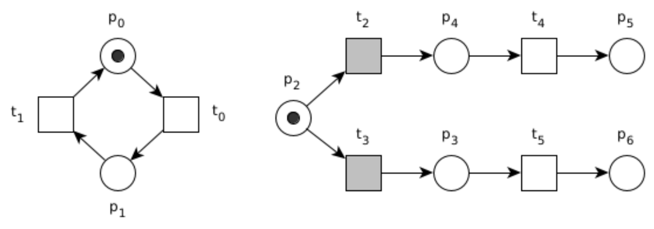


Figure 4.5: A net system with controllable and uncontrollable transitions (coloured resp. in grey and white)

*may wonder how reasonable defining the strategy on global states is, if the global information cannot be properly used in the strategy.*

## Observation of subsets of places

We consider the case in which the values of some local states are not observable by the user. We denote the observable places as $P_O \subseteq P$, and assume that ${}^\bullet T_u \subseteq P_O$, namely all the preconditions of controllable transitions are observable by the user. The case of full observability can be seen as a special case of this, in which $P_O = P$. However, since in literature many results hold only with full observability, I believe it makes sense to consider them separately.

Let $\Sigma = (P, T, F, W, m_{in})$ be a P/T system, $P_O$ the subset of observable places, $\textsc{unf}(\Sigma) = (B, E, F, \lambda)$ its unfolding, and $\gamma_1, \gamma_2$ two B-cuts. If the user has no memory, $\gamma_1 \equiv \gamma_2$ iff $\lambda(\gamma_1) \cap P_O = \lambda(\gamma_2) \cap P_O$. If the user has memory, let $B_O = \{b \in B : \lambda(b) \in P_O\}$; $\gamma_1 \equiv \gamma_2$ iff $\gamma_1 \cap B_O = \gamma_2 \cap B_O$.

When the user has partial information, it may not have a winning strategy on P/T systems where the strategy would exist with full information. In Fig. 4.1, if the user cannot observe the value of $s_1$ and $s_2$, it cannot decide which transition between $u_1$ and $u_2$ it should fire to avoid the deadlock.

The aim of the following example is to present a toy scenario in which the partial observability context is relevant, and show that also in this case the memory of the user may play a crucial role in the existence of a strategy. Here and in the following examples, observable places are drawn with a bold line.

**Example 41.** *Consider the net in Fig. 4.6, where the user represents a robber and the environment a banker. The goal of the user is to reach place w, representing a successful robbery, without ending in place p, representing the prison. In the initial marking, the environment can set the code of the vault to one or to two (by executing $sc_1$ or $sc_2$ respectively). The user can either go directly to the vault through transition gv, or can threaten the environment and receive a communication of the code through transition t.*

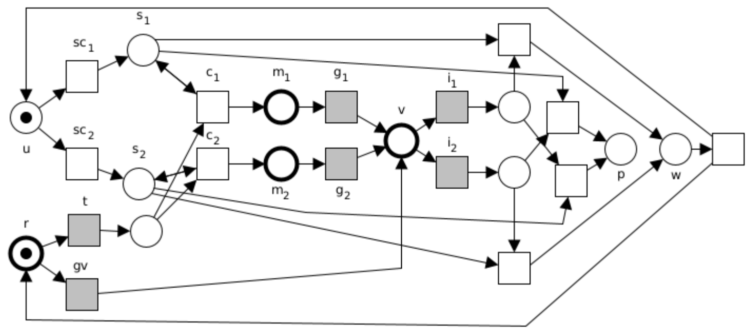


Figure 4.6: 1-safe system inspired by the example presented in [115], where the user represents a robber, and the environment a banker

*In order to successfully end the robbery, the user needs to choose between $i_1$ and $i_2$, agreeing with the decision of the environment. If the robber observes everything, there is no need to communicate with the environment in order to know the code; the robber could directly execute transition gv and observe whether $s_1$ or $s_2$ is marked. However, this scenario is not realistic, and we should assume $s_1$ and $s_2$ are not visible by the robber. In this case, the robber has a winning strategy only if some memory is allowed: in the marking $\{s_1, v\}$, in order to win the robber should choose $i_1$, whereas in the marking $\{s_2, v\}$, the robber should choose $i_2$. However, since $s_1$ and $s_2$ are not observable, the two markings are indistinguishable. In the unfolding this is not the case: if the robber fires transition $t$ in the initial marking, the occurrence of $v$ reached in the unfolding provides information about its story, therefore the robber can deduce which between $s_1$ and $s_2$ is marked without observing them directly.*

**A different notion of memory** In some cases, we may want the user to have memory of what happened in the system, but without discriminating between runs in which the user observed the same elements of the P/T system in the same order. This condition is in general not satisfied when we assume that the user can distinguish between conditions of the unfolding, as illustrated by the following example.

**Example 42.** *Consider the net in Fig. 4.7. We assume that the user cannot observe any occurrence of the places $p_3, p_4$ and $p_7$, and that its goal is to never reach a deadlock. In the figure, observable places are represented with bold lines. Let $B_O = \{b \in B : \lambda(b) \in P \setminus \{p_3, p_4, p_7\}\}$ be the set of conditions observable on the unfolding. Given two B-cuts $\gamma_1$ and $\gamma_2$, according to the notion of memory that we defined so far, $\gamma_1 \equiv \gamma_2$ iff $\lambda(\gamma_1) \cap B_O = \lambda(\gamma_2) \cap B_O$. With this definition of observation, the user has a winning strategy. Consider the prefix of the unfolding of Fig. 4.7 represented in Fig. 4.8. In the*

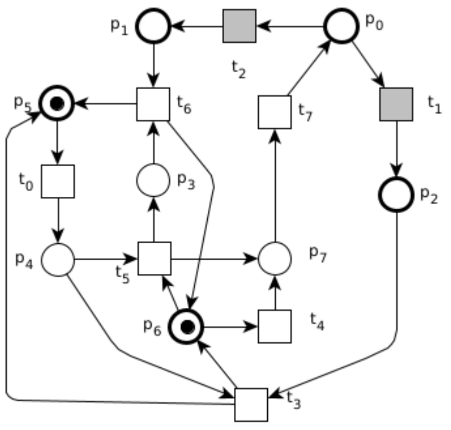


Figure 4.7: A P/T system where a winning strategy can be defined on B-cuts.

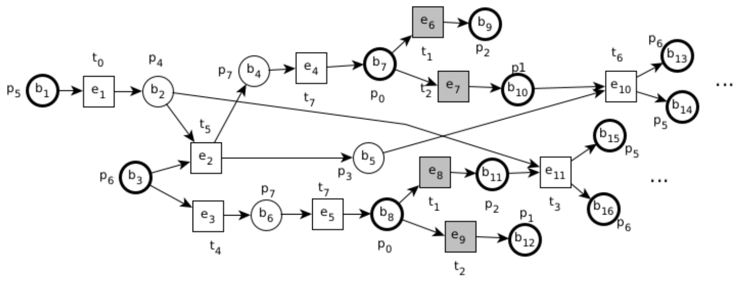


Figure 4.8: Prefix of the unfolding of the P/T system in Fig. 4.7

*B-cut $\gamma_1 = \{b_5, b_7\}$, the user can observe only $b_7$; in the B-cut $\gamma_2 = \{b_2, b_8\}$ the user can observe only $b_8$. Although $\lambda(b_7) = \lambda(b_8)$, $b_7 \neq b_8$, and therefore, based on the notion of observation given above $\gamma_1 \not\equiv \gamma_2$. This notion of observation implies that the user has full knowledge of what happened on the system, even when it was not able to observe it: by observing a condition on the unfolding, the user knows its entire past.*

*A different definition of observation may require that the user remembers every element of the P/T system that it observed, but cannot discriminate different elements of the unfolding. In this case, $\gamma_1 \equiv \gamma_2$ iff $\{\lambda(b) : b \in B_O,\ b \leq \gamma_1\} = \{\lambda(b) : b \in B_O,\ b \leq \gamma_2\}$, and the partial order of their occurrence is the same. When we consider this notion, $\{b_5, b_7\} \equiv \{b_2, b_8\}$, since in both cases the user observes the places $p_5$ and $p_6$ followed by $p_0$. The user can still remember the observations that it made in the past, but cannot distinguish between two runs if they differs only for unobservable elements in the P/T system.*

Finally, with the next notion of observation, we tackle the problem raised in Ex. 40.

**Observation of stable parts of markings**

As in the previous case, we assume that the user observes a subset of places $P_O \subseteq P$, and that ${}^\bullet T_u \subseteq P_O$. We restrict ourselves to the case of 1-safe systems.

In addition to the observability, in order to define what information the user can actually exploit to reach its goal, we introduce the concept of *stable part* of a B-cut. Let $\gamma$ be a B-cut on $\textsc{unf}(\Sigma) = (B, E, F, \lambda)$. Its stable part is defined as the set $\gamma^s = \{b \in \gamma \mid \nexists e_1 e_2 ... e_n \in E^*_{env} : b \in {}^\bullet e_n \wedge (\gamma[e_1 ... e_{n-1}\rangle \gamma_n \wedge {}^\bullet e_n \subseteq \gamma_n)\}$; in other words $\gamma^s$ is the set of conditions in $\gamma$ that cannot be consumed by any sequence of uncontrollable events enabled in $\gamma$. We can assume that the user can base its decisions only upon the observable and stable parts of B-cuts, namely, for each pair of B-cuts $\gamma_1$ and $\gamma_2$, $\gamma_1 \equiv \gamma_2$ iff

$$\gamma_1^s \cap \{b \in B : \lambda(b) \in P_O\} = \gamma_2^s \cap \{b \in B : \lambda(b) \in P_O\}.$$

The stable part of a marking is defined analogously. Given a marking $m$ and a cut $\gamma$ in $\textsc{unf}(\Sigma)$ with $\lambda(\gamma) = m$, compute $\gamma^s$; then, the stable part of $m$ is $m^s = \lambda(\gamma^s)$. This definition does not depend on the choice of $\gamma$ since, for each $\gamma'$ with $\lambda(\gamma') = m$, the future of $\gamma'$ in $\textsc{unf}(\Sigma)$ is isomorphic to the future of $\gamma$.

This notion is motivated by cases analogous to the one discussed in Ex. 40, where even if the value of a certain local state is not hidden to the user, its information may arrive to the user with a certain delay. By definition, if a place is not in the stable part of a marking, its value may change due to a transition out of the control of the user, therefore the information about it may arrive outdated to the user, that cannot count on it, as it was the case in Ex. 40.

**Example 43.** *In Fig. 4.5, in the initial marking only $p_2$ is in the stable part, since the value of $p_0$ may change due to the uncontrollable transition $t_0$. Analogously for $\{p_1, p_2\}$.*

*In some cases, the definition of stable part of a marking has consequences also on the controllable transitions that the user can fire. Consider the* 1*-safe system in Fig. 4.9, where $p_1$ and $p_2$ are the observable places, and $t_1$ and $t_2$ are the controllable transitions. Assume that the goal of the user is to reach $p_8$. From the initial marking the user cannot prevent the sequence $t_3 t_6$ to fire, and therefore it cannot count on the occurrence of $t_2$ to win, even if $t_2$ is controllable and enabled in the initial marking. This can be modelled thanks to the definition of stable part of a marking: there is no marking where $p_2$ is stable, because from every marking containing $p_2$, there is a sequence of*

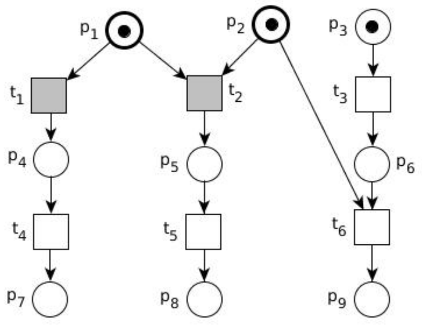


Figure 4.9: A P/T system with no stable marking in which $t_2$ is enabled

*uncontrollable transitions that could fire and make $p_2$ false (for example, as we already observed, the sequence $t_3t_6$ in the initial marking).*

In some cases, the information coming from the structure of the net and from some observations, even if delayed, is sufficient for the user to reach his goal, but is not included in the stable parts of the reachable markings. For example, the user may know that an observable place cannot be marked anymore in the future or that there must be a token in a certain set of observable places, even without knowing precisely in which one (see Ex. 44). For this reason, in order to use this information in the strategy, we consider some implicit places, corresponding to unions and complements of observable and compatible places (defined in Sec. 2.5.1). These new places, together with the set $P_O$, form a new set $P'_O$, defined recursively as follows. $P_O \subseteq P'_O$; if $p \in P'_O$, then its complement $p^c$ is in $P'_O$; if $p_1, p_2$ are in $P'_O$ and $p_1 \$ p_2$ then $p_1 \vee p_2 \in P'_O$. By construction all the places in $P'_O \setminus P$ are observable. Then, $P'$ is obtained as $P' = (P \setminus P_O) \cup P'_O$. We denote the net with these additional places as $\Sigma_S = (P', T, F', m'_{in})$; in what follow $\Sigma_S$ will be also referred as *extension* of $\Sigma$ or *extended net*. In the figures, the implicit places will be coloured in green.

Recall, from Sec. 2.5.1, that adding implicit places to a net does not change its behaviour. The extended net just makes explicit an information which was actually available in the original net. On the other hand, in defining a strategy we cannot use information corresponding to the union, or logical disjunction, of non compatible places, because this information cannot be coded in the form of a place, explicit or implicit.

**Example 44.** *Consider the net in Fig. 4.10, without the green places. Assume that the user can observe everything and its goal is to reach any marking among $\{p_3, p_5\}$, $\{p_5, p_7\}$ and $\{p_4, p_6\}$. A way for the user to reach this goal would be to wait for the environment to move, then fire $t_3$ if transition $t_1$ occurred, and $t_4$ otherwise. This would not be possible with a strategy defined on the stable parts of the markings of the initial net: the stable part*

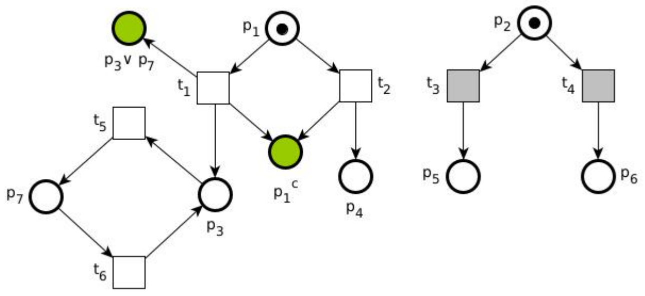


Figure 4.10: A 1-safe system and two of its implicit places (in grey)

*of $\{p_1, p_2\}$, $\{p_2, p_3\}$ and $\{p_2, p_7\}$ is $\{p_2\}$, therefore the strategy could not differentiate between the initial marking ($\{p_1, p_2\}$) and the markings reached after the occurrence of $t_1$ ($\{p_2, p_3\}$ and $\{p_2, p_7\}$). This problem is overcome when we consider the extended net, with the place $p_3 \vee p_7$ (added in green in the figure). Once that $t_1$ occurs, this place will belong to the stable parts of all the markings subsequently reached, therefore the strategy can use the information that one of those two places must be marked to select transition $t_3$. Note that the net in Fig. 4.10 is not the complete extended net, but the represented places are sufficient for the sake of presentation.*

## 4.2 Finding a strategy on a prefix of the unfolding

In this section we apply the general idea of asynchronous game to a specific reachability problem, and propose an algorithm to determine if the user has a winning strategy by constructing a prefix of the unfolding.

In particular we assume that the system is modelled with a 1-safe system in which choices among transitions are local either to the environment or to the user, and that transitions controlled by the user are never concurrent with each other, while they can be concurrent with transitions in the environment. The goal of the user is to force the occurrence of a target transition. The user can observe all the places in the 1-safe system and has no memory. The transitions of the environment must satisfy a progress constraint. The results presented here are published in [3].

As a formal setting, we refer to the so-called *distributed net systems*, as introduced and studied in [24] and in [128].

**Definition 22.** *A* distributed net system *over a set $K$ of locations is a 1-safe net system $\Sigma = (P, T, F, m_{in})$ together with a map*

$$\eta : (P \cup T) \to K$$

*such that for every $p \in P$, $t \in T$, if $p \in {}^\bullet t$, then $\eta(p) = \eta(t)$.*

We consider the special case of distributed net systems $\langle \Sigma, \eta \rangle$ such that $K = \{\text{environment}, \text{user}\}$, namely of distributed net systems with only two components, representing the environment and the user, respectively; we assume that the user controls all the transitions in its location, and these transitions are never concurrent with each other. From now on, by distributed net system we will mean a net system satisfying these constraints.

The choice of considering distributed net systems is due to the assumption that, in case of a conflict between a transition of the user and a transition of the environment, the user cannot be sure to win the conflict. Hence, the existence of a winning strategy cannot depend on the occurrence of such transitions. In distributed net systems, when a transition is enabled, it can never be disabled by the occurrence of transitions belonging to different components. In the case of a cycle this observation justifies the following lemma.

**Lemma 15.** *Let $\langle \Sigma, \eta \rangle$ be a distributed net system with two locations, $A$ and $G$. Let $m$ be a marking, and*

$$m_1[t_1\rangle m_2[t_2\rangle m_3[...\rangle m_1$$

*be a firing sequence with $\eta(t_i) = A$ for each $i$. Then, if $\eta(t) = G$, and $t$ is enabled at $m_i$ for some $i$ between the two repetitions of $m_1$, then $t$ is enabled at $m_j$ for each $m_j$ in the cycle.*

The notions of unfolding and run apply in the obvious ways to distributed net systems.

**Example 45.** *Fig. 4.1 and Fig. 4.6 are examples of distributed net systems with two locations. Places are not explicitly divided into the two components, because their partition can be inferred by their post-transitions.*

Let $\langle \Sigma, \eta \rangle$, be a distributed net system, where $\Sigma = (P, T, F, m_{in})$, $\text{UNF}(\Sigma) = (B, E, F, \lambda)$ be its unfolding, and $t$ the target transition that the user needs to fir to win. We define as winning condition for the user the set of plays $(\rho, \delta)$ such that there is an event $e \in E_\rho$ with $\lambda(e) = t$. In this case, the sequence $\delta$ is not relevant to decide whether a play is won by the user. We say that the target transition $t$ is *controllably reachable* in $\Sigma$ if, and only if, there is a strategy $\alpha$ on $\text{UNF}(\Sigma)$ such that the user wins every play consistent with $\alpha$. Example 38 can be seen as an example of the game presented in this section. The strategy discussed in the example is a winning strategy for this game.

**Example 46.** *The net shown in Fig. 4.11 is distributed, with two locations, its unfolding is represented in Fig. 4.12. Assume that $t_4$ is the target transition. If the environment cooperates with the user by eventually choosing $t_1$, then the target is reached. However, the environment can choose $t_2$ at*

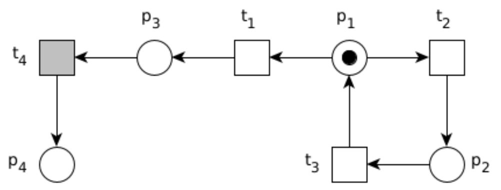


Figure 4.11: A distributed net system

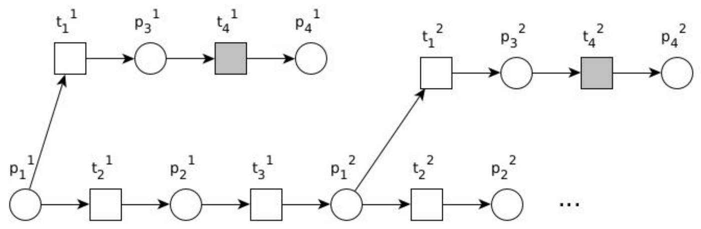


Figure 4.12: The (prefix of the) unfolding of the net system shown in Fig. 4.11

*every B-cut consisting in an occurrence of $p_1$ without violating any fairness constraint. Hence, irrespective of the strategy chosen by the user, an infinite play made of repeated occurrences of the cycle $p_1$, $t_2$, $p_2$, $t_3$, $p_1$ is admissible, therefore the user does not have any winning strategy.*

In a general case, given a strategy $\alpha$, there are infinitely many plays consistent with $\alpha$ in $\text{UNF}(\Sigma)$, and some plays could be infinite, hence the exhaustive exploration of them would take infinite time.

We propose an algorithm that, given a distributed net system and a target transition, establishes if there is a winning strategy for the controlled reachability of the target and, if so, computes a winning strategy.

### 4.2.1 Algorithm for a winning strategy

In this section we present an algorithm that looks for a winning strategy for the reachability of a target transition, and we illustrate how it works on the 1-safe system in Fig. 4.13. This 1-safe system modifies the one represented in Fig. 4.1, and the discussion in Ex. 38 can be adapted to it. Unlike the 1-safe system in Fig. 4.1, this 1-safe system does not restore the initial marking after the occurrence of $z$, and there may be an uncontrollable cycle made by transitions $t_3$ and $t_4$ firing concurrently to $z$. These details help to highlight some features of the proposed algorithm.

The algorithm generates a prefix of the unfolding of a given distributed

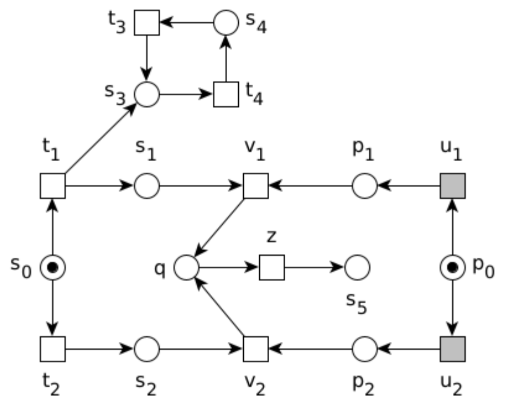


Figure 4.13: A variant of the distributed net system presented in Fig. 4.1.

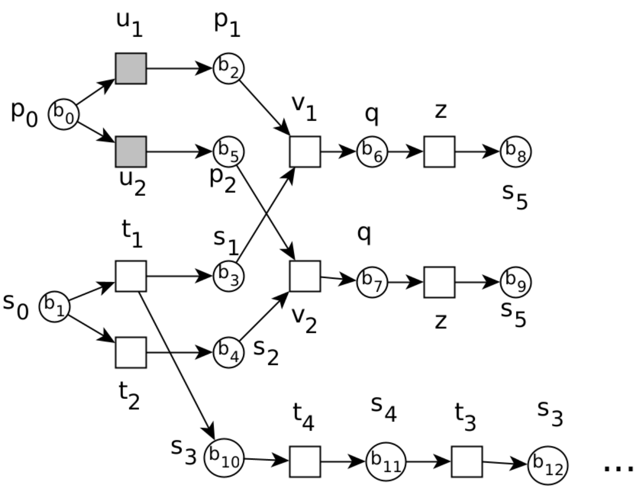


Figure 4.14: A (prefix) of the unfolding in Fig. 4.13.

net system, deciding whether there exists a winning strategy for the user. In the positive case, it gives as output a strategy as a function on reachable markings; the strategy is initially associated to B-cuts of the unfolding, but the algorithm works so that, for distinct B-cuts corresponding to the same marking, the strategy gives the same answer.

Before starting a detailed description of the algorithms, we give an intuition of the ideas underlying it. The strategy of the user cannot be based on firing a controllable transition faster than an uncontrollable enabled transition; furthermore, once that a controllable transition is enabled, only the user can disable it. Therefore, waiting for the decisions of the environment as much as possible can help the user to gather information useful for the strategy. However, waiting until only controllable transitions are enabled is not possible, since the environment could enter an infinite cycle (for example by firing infinitely often transitions $t_3$ and $t_4$ in Fig. 4.13), and this must be detected by the algorithm. Hence, in order to find the strategy, the algorithm starts unfolding the uncontrollable part of the 1-safe system, and once that this part is maximally unfolded or a cyclic uncontrollable behaviour was found, it checks which controllable choices result in a victory for the user. Once that a candidate solution has been found, the algorithm needs to backtrack to see whether alternatives uncontrollable behaviours may necessitate to revise some choices.

The input data are the following:

- A net in which the transitions are enumerated so that all the uncontrollable transitions precede all the controllable ones. If the target is a controllable transition, it must be the first of the controllable transitions.
- The position of the first controllable transition.
- The initial marking $m_{in}$ of the system.
- The target transition.

The value of these variables is available for all the functions of the algorithm and does not change during its execution.

The core of the algorithm is the recursive function UNF_EXPLORATION (see Algorithm 7), which unfolds the 1-safe system by exploring reachable B-cuts, and constructs at the same time a prefix of the unfolding and a strategy.

The function takes five input arguments:

1. $\gamma$: the B-cut that must be analysed;
2. $M$: the list of markings associated to the B-cuts already analysed in the current run;

**Algorithm 7** Unfolding exploration

```
function UNF_EXPLORATION($\gamma$, $M$, $E_l$, $e$, $sz$)
    if $e = target$ then
        return true
    else if $\gamma$ is a deadlock or enables only transitions in $sz$ then
        return false
    else if $\gamma \in \Gamma_{bad}$ then
        return false
    else if $\gamma \in \Gamma_{good}$ then
        return true
    else if $\lambda(\gamma) \in M$ then return EXPLORE_CUT_C$(\gamma, M, E_l)$
    else if ENAB_N$(\gamma) \neq \emptyset$ then
        $E \leftarrow$ ENAB_N$(\gamma)$
        repeat
            $e_0, E \leftarrow$ EXTRACT$(E)$
            $v \leftarrow$ UNF_EXPLORATION$(\gamma + e_0, M.append(\lambda(\gamma)), E.append(e), e_0)$
            if $v =$ true then
                unf $\leftarrow$ unf $\cup[\gamma, e_0, \gamma + e_0]$
            end if
        until $E = \emptyset \vee v =$ false
        if $v =$ true then
            if $\gamma \in ver$ then
                $sz \leftarrow$ STABLE_ZONE$(E)$
                $v \leftarrow$ UNF_EXPLORATION$(\gamma, M, E_l, e)$
            else
                $\Gamma_{good}.append(\gamma)$
            end if
        else
            $\Gamma_{bad}.append(\gamma)$
        end if
        return $v$
    else
        $E \leftarrow$ENAB_C$(\gamma)$
        repeat
            $e_0, E \leftarrow$ EXTRACT$(E)$
            $v \leftarrow$ UNF_EXPLORATION$(\gamma + e_0, M.append(\mu(\gamma)), E_l.append(e), e_0)$
            if $v =$ true then
                unf $\leftarrow$ unf $\cup[\gamma, e_0, \gamma + e_0]$
                str $\leftarrow$ str $\cup$ $[\gamma,\ e_0]$
            end if
        until $E = \emptyset \vee v =$ true
        if $v = true$ then
            if $\gamma \in ver$ then
                $sz \leftarrow$ STABLE_ZONE$(E)$
                $v \leftarrow$ UNF_EXPLORATION$(\gamma, M, E_l, e)$
            else
                $\Gamma_{good}.append(\gamma)$
            end if
        else
            $\Gamma_{bad}.append(\gamma)$
        end if
        return $v$
    end if
end function
```

3. $E_l$: the list of events that fired in the current run;

4. $e$: the last event added to the current run, leading to $\gamma$;

5. $sz$: the set of events enabled in $\gamma$ that are part of a cycle or that are in conflict with events that are in a cycle.

It returns a boolean variable, that is equal to true if there is a winning strategy, for all the plays passing from the input B-cut $\gamma$ consistent with the strategy, false otherwise. In addition, it possibly modifies the prefix and the strategy, initially empty, filling them with events, B-cuts and choices already explored.

The first time that the function is called, the input consists always in the initial cut $\gamma_{in}$ in the unfolding, empty lists for the list of visited markings, the list of analysed events and the list of events that are part of cycles or in conflict with them (those events will be discovered during the execution of the algorithm), and a fictitious event $i$. The function UNF_EXPLORATION uses some auxiliary functions:

- ENAB_N is a function that has an input B-cut and returns the list of uncontrollable events which are enabled in that specific B-cut;

- similarly, ENAB_C returns the controllable enabled events.

- EXTRACT returns the first element of an input list, and the list deprived of this element.

- STABLE_ZONE returns the set of events that can be part of a cycle, and those in conflict with them.

Let us recall that we denote with $\gamma + e$ the B-cut obtained by firing the event $e$ in the B-cut $\gamma$.

The function constructs every run by adding uncontrollable events until one of the following cases occurs: (1) the target occurs; (2) a deadlock cut is reached or a cut is reached in which only transitions that are part of a cycle or that are in conflict with events in a cycle are enabled; (3) a cut that has been previously analysed is reached (two subcases are considered); (4) a cut is reached in which no uncontrollable event is enabled, and some controllable events are enabled; (5) a cut is reached corresponding to a marking which has already been visited in the current run, and there are not uncontrollable enabled events that are concurrent with all the ones that occurred between the two equivalent markings.

In case (1), the current run corresponds to a play won by the user; hence the function tries to backtrack along choices among uncontrollable events, if possible. Symmetrically, in case (2), the current run corresponds to a play won by the environment; hence, the function tries to backtrack along choices among controllable events, if possible. In cases (3), the current run is the prefix of a set of runs that have been already analysed. The user wins or loses according to the analysis previously done. In case (4), a controllable event is added, and the exploration restarts from the new B-cut. Finally, in case (5), if possible, a controllable event is added, and the exploration restarts from the new B-cut; if this is not possible, the run corresponds to a play won by the environment and the function tries to backtrack and change the previous controllable choices.

**Example 47.** *Consider the net system shown in Fig. 4.13 and its unfolding (Fig. 4.14). Starting from the initial cut, the algorithm adds the event labelled with $t_2$, reaching a B-cut in which only controllable transitions are enabled. It then adds the event labelled with $u_2$, reaching the B-cut $\{b_4, b_5\}$, and starts again adding uncontrollable transitions. This run will lead to the target event $z$, hence it is not necessary to backtrack on controllable events.*

*The next backtracking step goes back to the initial cut, and starts exploring a new run by adding the event labelled with $t_1$; from $\{b_0, b_3, b_{10}\}$, the events labelled with $t_4$ and $t_4$ fire. This produces the cut $\{b_0, b_3, b_{12}\}$, that corresponds to a marking that has already been visited. Hence, the controllable event labelled with $u_2$ is added, leading to a cut in which only the cycle formed by occurrences of $t_4$ and $t_3$ can occur, thus repeating the same marking. The algorithm backtracks and tries the occurrence of $u_1$. The events $v_1$ and $t_4$ are enabled in $\{b_2, b_3, b_{12}\}$. After the occurrence of $v_1$, the target is reached.*

In the following, we explain in detail how UNF_EXPLORATION works in a general step of execution of the algorithm. If $\gamma$ is a cut of a play on the unfolding, one of these situations is verified:

1. $\gamma$ is not a deadlock, it enables events that are not part of cycles or in conflict with them, has not been previously analysed, it is the first time that the associated marking is visited in the play, the target has not occurred yet and there are $k$ uncontrollable enabled transitions to analyse in $\lambda(\gamma)$. In this case, the prefix of the play currently ending with $\gamma$ is extended in $k$ different plays, each of them obtained by adding a different uncontrollable event after $\gamma$. The output for this step is 'true' only if the values returned by all recursive calls on the cuts that immediately follow $\gamma$ is 'true'.

   Considering the system in Fig. 4.13 and its unfolding (Fig. 4.14), we find the described situation in the initial cut of the unfolding: in $\{b_0, b_1\}$, both the event labelled with $t_1$ and the one labelled with $t_2$ are enabled. Therefore, the algorithm extends the current prefix considering the two plays obtained by adding the two events and the cuts that follow their occurrence.

2. $\gamma$ is not a deadlock, $\lambda(\gamma)$ has never been analysed in the play, the target did not fire in the previous part of the play and the only enabled events that are not part of cycles or in conflict with them are controllable. In this case, the algorithm analyses the controllable events in the order induced by enumeration of the transitions in the net, until it finds an extension that returns 'true' as output or it ends the analysis of all the controllable events enabled in $\gamma$.

   Referring to Fig. 4.14, the cut $\{b_0, b_4\}$ enables an occurrence of transitions $u_1$ and $u_2$. Assume that the algorithm starts constructing the play with $u_2$. After verifying that the user has a strategy to win all the plays passing from the cut $\{b_4, b_5\}$, the function does not continue with the analysis of $\{b_4, b_2\}$, and returns the Boolean value 'true'.

3. Either $\gamma$ is a deadlock, or all the enabled events are part of a cycle or in conflict with events in a cycle, or $\gamma$ follows the target transition. These are base cases for the recursive algorithm. Their occurrence stops the exploration for that play. If the target fired, the algorithm returns 'true', in all the other cases of this situation, it returns 'false'.

   In the considered example, all the plays ending with a cut in which there is an occurrence of $s_5$ are winning for the user (because an occurrence of $z$ has necessarily fired).

4. $\gamma$ has already been considered in a previous step. In this case, the analysis stops and the function returns 'true', if the first analysis of the cut returned 'true', and 'false' otherwise. This case is verified in case of concurrency in the environment component.

5. $\lambda(\gamma)$ was already visited in the play. In this case, the algorithm checks if any controllable event fired between the two repetitions. If this happens it returns 'false'. (This is justified by the fact that the victory of the user cannot depend on the choice of a controllable transition that contributes to a cycle without the target.) Otherwise, it analyses only the events that are enabled and concurrent with all the ones fired in the cycle. If there are uncontrollable events among them, then it behaves as in 1; if there are only controllable events, it behaves as in 2; if there is no event satisfying the requirements, it behaves like in a deadlock situation.

   During the execution of the algorithm on the system in Fig. 4.13, the cut $\{b_3, b_5, b_{10}\}$ is analysed. The only enabled event is the occurrence of $t_4$, but it is not added to the play, because it depends on the repeated occurrences of transitions $t_3$ and $t_4$, that create a cycle in the system. Hence, the algorithm returns 'false' for this particular play. Later, changing the controllable choice, it analyses the cut $\{b_2, b_3, b_{10}\}$. In this cut, $v_1$ is enabled and concurrent with all the occurrences of $t_3$ and $t_4$, hence, the algorithm extends the play with it.

The functions EXPLORE_CUT_C (Algorithm 8) and F (Algorithm 9) deal with concurrency. Specifically, EXPLORE_CUT_C is called by UNF_EXPLORATION when a cut associated with a marking repeated in the run is detected. The function F is called by EXPLORE_CUT_C; it takes the current cut, the list of the previously visited markings, and the list of events that have been fired. It checks whether a controllable event fired in the cycle; if not, it returns the list $E$ of events concurrent with all the events occurred after the first cut in the run associated to the same marking as the current one. The events in $E$ are the only ones considered by EXPLORE_CUT_C to extend the prefix of the run.

In Algorithm 8, there are two more auxiliary functions:

- $E_{NC}$ takes a list of events as input, and returns only the uncontrollable ones.
- Symmetrically, $E_C$ takes a list as input, and returns the controllable events in it.

Both UNF_EXPLORATION and EXPLORE_CUT_C are responsible for the construction of the prefix and the strategy. The prefix is updated every time that UNF_EXPLORATION returns the value 'true' (with the exception of the very first call). When this happens, the receiving function appends to the prefix a triple consisting of its input cut $\gamma$, the following cut $\gamma + e$ that was in input to the call to the function that just returned 'true', and the event $e$. If the added event $e$ is controllable, then the strategy is also updated. In particular, the algorithm appends the input cut $\gamma$ coupled with the controllable transition $\lambda(e)$ to the current strategy.

At the end of the execution of UNF_EXPLORATION, if there is a winning strategy, it is defined on the cuts of the prefix. To complete it, we have to define it on the markings, detect the parts of the plays corresponding to a cyclic behaviour on the system and, if the strategy chooses a transition immediately after them, the algorithm has to fill the strategy, attributing the same choice to all the markings in the cycle.

### 4.2.2 Discussion

In this section, we discuss the correctness of the proposed algorithm.

**Lemma 16.** *Every play exploration ends due to one of the following ending criteria:*

1. *The target fires. In this case the user wins all the plays with the constructed prefix.*

**Algorithm 8** Cuts associated with markings already visited in the prefix

**Input:** the cut $\gamma$ that must be analysed, the ordered list $M$ and $E$ of the markings and events that occurred in the run before $\gamma$.

```
function EXPLORE_CUT_C($\gamma, M, E_l$)
    $E, E_{l_{reap}} \leftarrow$ F($\gamma, M, E_l$)
    if $E = \emptyset \vee E_C(E_{l_{reap}}) \neq \emptyset$ then return false
    else
        $E \leftarrow$ F($\gamma, M$)
        $E_{nc} \leftarrow E_{NC}(E)$
        $E_c \leftarrow E_C(E)$
        if $E_{nc} \neq \emptyset$ then
            $v$ = true
            repeat
                $e_0, E_{nc} \leftarrow$ EXTRACT($E_{nc}$)
                $v \leftarrow$ UNF_EXPLORATION($\gamma + e_0, M, e_0$)
                if $v$ = true then
                    unf $\leftarrow$ unf $\cup[\gamma, e_0, \gamma + e_0]$
                end if
            until $E_{nc} = \emptyset \vee v$ = false
            if $v = true$ then
                $\Gamma_{good}.append(\gamma)$
            else
                $\Gamma_{bad}.append(\gamma)$
            end if
            $ver.append(\lambda(\gamma), E_{l_{reap}})$
            return $v$
        else
            $v \leftarrow$ false
            repeat
                $e_0, E_c \leftarrow$ EXTRACT($E_c$)
                $v \leftarrow$ UNF_EXPLORATION($\gamma + e_0, M, e_0$)
                if $v$ = true then
                    unf $\leftarrow$ unf $\cup[\gamma, e_0, \gamma + e_0]$
                    str $\leftarrow$ str $\cup$ $[\gamma, e_0]$
                end if
            until $E_c = \emptyset \vee v$ = true
            if $v = true$ then
                $\Gamma_{good}.append(\gamma)$
            else
                $\Gamma_{bad}.append(\gamma)$
            end if
            $ver.append(\lambda(\gamma), E_{l_{reap}})$
            return $v$
        end if
    end if
end function
```

**Algorithm 9** Events that are concurrent with the ones that already fired in the run

**Input:** the cut $\gamma$ that must be analysed and the ordered lists $M, E_l$ of the markings and the events that occurred in the run before $\gamma$.

**Output:** list of events that have been enabled from the cut associated with the first occurrence of the marking $\lambda(\gamma)$ to the current cut $\gamma$, list of events that occurred in the run between the two repetitions on $\lambda(\gamma)$.

```
    function F($\gamma, M, E_l$)
2:      i ← 0
        while $M[i] \neq \lambda(\gamma)$ do
4:          i ← i+1
        end while
6:      $E_{l_{reap}} \leftarrow E_l[i : len(E_l)]$
        $E \leftarrow []$
8:      for all $e \in$ENAB_C($\gamma$) do
            if $\lambda(e)$ enabled in $m$ $\forall m \in M[i : len(M)]$ then
10:             $E.append(e)$
            end if
12:     end for
        return $E, E_{l_{reap}}$
14: end function
```

**Algorithm 10** Full strategy

```
    $v$ = UNF_EXPLORATION($\gamma_{in}$, [], i)
2:  if v == True then
        str = CUTS_TO_MARKINGS()
4:      str = COMPLETE_STRATEGY()
    end if
```

*2. The play reaches a deadlock cut $\gamma$ before reaching the target. In this case the user loses the play.*

*3. The play reaches a cut in which the target has not fired, and the only enabled transitions can be part of cycles or in conflict with transitions that can be part of a cycle. In this case the user loses the play.*

*4. The play reaches a cut $\gamma$ that was previously analysed.*

*5. The play reaches a cut $\gamma'$ such that there is another cut $\gamma : \gamma < \gamma'$ for which $\lambda(\gamma) = \lambda(\gamma')$, $\gamma$ corresponds to the first occurrence of $\lambda(\gamma)$ in the play, and*

   - *either $\gamma'$ does not enable any event that is concurrent with all the events occurred between $\gamma$ and $\gamma'$,*
   - *or there is a controllable event $e$ such that $\gamma < e < \gamma'$.*

   *If the prefix is consistent with the strategy, the user loses at least a play by following the induced strategy.*

*Moreover, if $\alpha$ is a strategy defined on the markings, then, for every prefix of a play consistent with $\alpha$ determined with one of these criteria, we can decide if the user wins all the plays consistent with $\alpha$ starting with such a prefix.*

*Proof.* 1. If the target fired in the prefix, then every play with such a prefix is winning for the user, because it includes the target.

2. If the target does not fire and the play is in a deadlock, the prefix coincides with the whole play. Since it does not have the target, it is losing for the user.

3. If the target does not fire and the only enabled transitions can be part of a cycle or be in conflict with transitions in cycles, then the user cannot prevent the environment to remain in the cycles forever (the transitions in a cycle are uncontrollable by construction). Since the target is not part of this cycle, the user cannot be sure to reach it.

4. If two prefixes end with the same cut $\gamma$, it means that they differ only for the order in which the concurrent events occurred, and their possible elongations are the same. The winning condition for the user does not depend on the order in which events occurred, but only from the presence of the target in the run. Hence, if the algorithm is requested to analyse a cut for which it has already determined if $\alpha$ is winning, it can immediately stop and return the same answer.

5. First, we have to show that if the play does not reach the target, does not end with a deadlock, and does not reach a cut previously analysed, then this last criterion is verified. The number of reachable markings in the system is finite, hence after a number of steps equal at most to the number of reachable markings, the algorithm analyses a cut $\gamma'$, such that $\lambda(\gamma) = \lambda(\gamma')$, where $\gamma$ is a cut preceding $\gamma'$ and belonging to the same play. Let us suppose that $k$ events are enabled in $\gamma'$ and concurrent with all the ones fired between $\gamma$ and $\gamma'$. The algorithm adds one of these to the play and continues as before. If the play reaches a cut $\gamma''$ such that $\lambda(\gamma) = \lambda(\gamma'')$, then the events that the algorithm analyses are necessarily strictly less then $k$, because they should be concurrent both with the events occurred between $\gamma$ and $\gamma'$ and with those fired between $\gamma'$ and $\gamma''$. Since for every repetition, the number of events satisfying the requirements to be added decreases, after at most $k$ cuts corresponding to the same marking $\lambda(\gamma)$, the third criterion is satisfied. Notice that this does not depend on the specific cut: the same reasoning applies to all markings.

   The next step is showing that if there is a play with such a prefix, then there is at least a play that follows the same strategy in which the user loses. We first consider the case in which there are not enabled events concurrent with all the ones in the cycle. If the prefix follows the strategy $\alpha$, then the play repeating infinitely many times the behaviour of the prefix is a play consistent with $\alpha$ and the target never occurs. We cannot guarantee that the user will lose all the plays consistent with $\alpha$ with such a prefix, but the fact that there is at least one is enough to state that $\alpha$ is not a winning strategy for the user. Secondly, we consider the case where a controllable transition fired between two occurrences of the same marking. By construction, the algorithm analyses controllable events only when all the significant uncontrollable events have been fired; hence, there cannot be any uncontrollable event that is concurrent with the cycle and that leads to the target, otherwise it would have been analysed before in the prefix. Again, if the prefix is consistent with the strategy $\alpha$, the play that repeats infinitely often the cycle is a play consistent with $\alpha$ and does not contain the target.

□

A consequence of Lemma 16 is the termination of the algorithm. We proved that every prefix constructed by the algorithm is finite. The number of considered plays is also finite, because at every step there is only a finite number of enabled events to extend the prefix.

By construction, if the algorithm finds a winning strategy, all the runs in the prefix: (1) are consistent with the strategy, and (2) contain the target.

(1) All the plays in the list are consistent with the strategy. Every time that the algorithm analyses a B-cut $\gamma$ and chooses to extend the prefix with a controllable event, it explores all the plays including $\gamma$ and, one by one, each of the controllable enabled events. It stops when it finds a controllable enabled event such that, from the cut of the unfolding following this event, the user has a winning strategy. When this happens, the prefix is updated, adding the event and the B-cuts preceding and following it. Also the strategy is updated, choosing the associated controllable transition in $\gamma$. In this way, at every step, all the parts of runs in the prefix constructed until that moment are consistent with the strategy updated until that moment. If in $\gamma$ there is no controllable enabled event such that, after it, the user has a winning strategy, then the part of the prefix already generated is not connected to the initial cut in the unfolding, since the event connecting this part to $\gamma$ is not added to the prefix. At the same way, if there is a winning strategy, it cannot depend on the strategy calculated on the disconnected parts of the unfolding. If the algorithm finds a winning strategy and a disconnected part was found, since the algorithm chooses a controllable event in $\gamma$ only when it is necessary to win, then there must be another cut in the prefix, that precedes $\gamma$ in the partial order, in which the algorithm adds a controllable transition that allows the user to avoid $\gamma$.

(2) All maximal runs in the prefix contain the target. If a run ends without the target, then the strategy allowing that run is not winning and must be changed. If it cannot be changed, then the algorithm will not state that there is a winning strategy, hence there must be a controllable node in which the decision previously taken can be changed. When another possible choice is analysed, all parts of runs depending on the previous one are deleted. Hence all the remained runs contain the target.

If the algorithm finds a winning strategy, every play in the unfolding consistent with this strategy is equivalent to an extension of a play in the prefix. This is shown in two steps.

1. Let us first consider the case without uncontrollable cyclic behaviours of the system.

   The strategy $\alpha$ constructed by the algorithm chooses a controllable transition only if there are not uncontrollable enabled ones. Let $\{t_1, ..., t_n\}$ be a set of uncontrollable transitions in a play, so that after their occurrence, there are not other uncontrollable enabled transitions. In whatever order the transitions are considered, the cut in the unfolding after their occurrence is the same, and the strategy will choose the same transition, because the following part of the unfolding is every time visited in the same way. Considering a play consistent with $\alpha$, there must be a prefix of its run in the unfolding, because all the uncontrollable transitions are analysed in all the uncontrollable cuts of the prefix and the strategy chooses only a transition for every cut, the controllable choices must be the same of the ones considered in the prefix. This is enough to state that the play is won by the user, because in the common prefix of the run there is the target transition.

2. If there are uncontrollable cyclic behaviours, such that there is a concurrent enabled transition leading to the target, then there is more variety in the possible plays consistent with $\alpha$, because the strategy is defined on markings in which uncontrollable events are enabled. Anyway, if a consistent play has a prefix with the same events of one of the prefixes produced by the algorithm, then it is won by the user, regardless of the order in which the events occurred. Some of the plays consistent with $\alpha$ have a longer uncontrollable part, because if an uncontrollable transition would be finally enabled, or a controllable transition would be finally enabled and eligible, there must be a certain point in which it will fire, but the precise point is unknown. However, since we complete every cycle at least once and from every cut that is not a repetition all the possible uncontrollable extensions are explored, and since the part of the unfolding starting from a given cut is isomorphic to the part of the unfolding starting from every cut corresponding to the same marking, the uncontrollable sequence of the play can be divided in parts such that an isomorphic one has been considered by the algorithm.

Based on the previous observations, if the algorithm finds a winning strategy, the proposed strategy is winning in the unfolding.

Finally, we wish to show that if the algorithm states the existence of a winning strategy and proposes one on the cuts of the prefix, to complete it adding the same choice to all the markings that are part of a cycle is necessary and does not change the correctness.

- Let us suppose that a cycle with only uncontrollable transitions is in the net, and there is a controllable enabled transition that is concurrent with all the transitions in the cycle and which is necessary for the victory of the user. The strategy constructed together with the prefix adds the choice of the controllable transition only in the cut associated to the last repeated marking. This strategy is incomplete, because the chosen transition is not finally eligible, since every time that the system is in a marking of the cycle that is not the one that has been repeated in the prefix, the strategy does not choose it. To overcome this problem we fill the strategy by adding the choice of the controllable transition in every marking along the cycle.

- This preserves the correctness of the strategy. Actually, if $\gamma$ is a cut that in a certain play is between $\gamma_1$ and $\gamma_2$ with $\gamma_1 < \gamma_2$ and $\lambda(\gamma_1) = \lambda(\gamma_2)$ and $\alpha'$ is the strategy computed by Algorithm 7 and translated on markings, then necessarily $\alpha'(\lambda(\gamma)) = \alpha'(\lambda(\gamma_1))$ or $\alpha'(\lambda(\gamma)) = \emptyset$.

  Specifically, if $\alpha'(\lambda(\gamma)) \neq \emptyset$, then it has to be $\alpha'(\lambda(\gamma)) = \alpha'(\lambda(\gamma_1))$. Let us suppose that $\{t_i\} = \alpha'(\lambda(\gamma))$, $\{t_j\} = \alpha'(\lambda(\gamma_2))$ and $t_i$ precedes $t_j$ in the enumeration defined by the input net. Then, $t_i$ is not a winning choice in $\lambda(\gamma_2)$, but there is a play that leads from $\gamma$ to $\gamma_2$ and between these two cuts only uncontrollable events fire (because the controllable component is sequential). The algorithm updates the strategy in a cut only if, starting from that cut, the user is able to win every play. This cannot be the case, because the play can arrive in $\gamma_2$ and the user loses the play. Reasoning in the same way, it cannot be that $t_i$ follows $t_j$ in the enumeration. Hence it must be $\{t_i\} = \alpha'(\lambda(\gamma)) = \alpha'(\lambda(\gamma_1))$.

  If $\lambda(\gamma)$ is never visited more than once in any run, then $\alpha'(\lambda(\gamma)) = \emptyset$. We construct a final strategy $\alpha$ such that $\alpha = \alpha'$ for every marking $m$ in which $\alpha'(m) \neq \emptyset$ and $\alpha(m') = \alpha'(m_1)$ for all $m'$ such that there is a play in the unfolding with two cuts $\gamma_1, \gamma_2 : \lambda(\gamma_1) = \lambda(\gamma_2) = m_1$ and a cut $\gamma : \lambda(\gamma) = m'$ and $\gamma_1 < \gamma < \gamma_2$.

  The marking $m'$ could be reached in more than one run, and if it is part of two different uncontrollable cycles, with different repeated markings, there could be the doubt that $\alpha(m')$ is not well defined, but this is not possible. Let us suppose that there is another play in the unfolding with two cuts $\gamma_1', \gamma_2' : \lambda(\gamma_1') = \lambda(\gamma_2') = m_2 \neq m_1$ and a cut $\gamma' : \lambda(\gamma') = \lambda(\gamma) = m'$ and $\gamma_1' < \gamma' < \gamma_2'$. We have to show that if there is a winning strategy, then $\alpha'(\lambda(\gamma_1)) = \alpha'(\lambda(\gamma_1'))$. By contradiction, let us assume $\{t_i\} = \alpha'(\lambda(\gamma_1))$, $\{t_j\} = \alpha'(\lambda(\gamma_1'))$ and $t_i$ precedes $t_j$ in the enumeration (the opposite case is equivalent due to the symmetry of definitions). Then, $t_i$ is not a winning choice for $\gamma_1'$, otherwise it would have been chosen before analysing $t_j$. If $t_i$ is not winning for $\gamma_1'$, then it cannot be winning from $\gamma_1$, because, starting from $\gamma_1$ the play can arrive in $\gamma$ firing only uncontrollable transitions, and from $\gamma$ there is a path made only by uncontrollable transitions to a cut $\gamma_2''$ such that $\lambda(\gamma_2'') = \lambda(\gamma_1')$. Since the unfolding starting from $\gamma_2''$ is isomorphic to the one starting from $\gamma_1'$, if the strategy is not winning from $\gamma_1'$ it cannot be winning from $\gamma_2''$ and therefore from $\gamma_1$.

### 4.2.3 Experiments

This work is mainly theoretical, and a full experimental evaluation of the algorithm is left for future works. However, we tested the algorithm on some

Table 4.1: Results of the experiments

| Net | $\|P \cup T\|$ | $\|K\|$ | conc | cycles | #calls | g_dim |
|---|---|---|---|---|---|---|
| heart | 15 | 2 | 2 | no | 10 | 8 |
| double_heart | 26 | 2 | 3 | no | 31 | 24 |
| big_heart | 141 | 30 | 2 | no | 355 | 126 |
| heartC | 19 | 2 | 2 | yes | 20 | 14 |
| bc | 23 | 2 | 3 | no | 19 | 16 |
| bc2 | 27 | 2 | 4 | yes | 1882 | 1162 |
| 10conc0 | 32 | 0 | 10 | no | 5122 | 1024 |
| 10conc1 | 32 | 1 | 10 | no | 2307 | 1025 |
| 10conc2 | 32 | 2 | 10 | no | 1028 | 258 |
| conc | 12 | 2 | 3 | yes | 255 | 143 |

preliminary examples. The set of the examples and a Python implementation of the algorithm are available at `https://github.com/MC3-lab/PNstrunf`.

The parameters of the net that we think are important to consider are: (1) the number of elements in the net; (2) the number of controllable transitions; (3) the level of concurrency, i.e. the maximum number of concurrent transitions that are enabled in a reachable marking; (4) the presence of cycles. We evaluate the performance of the algorithm by showing the total number of calls to the functions UNF_EXPLORATION and EXPLORE_CUT_C, and the number of cuts in the prefix at the end of the execution. The results of the experiments are shown in Table 4.1. In all these cases, the user has a winning strategy.

From the results, we see that the level of concurrency and the cycles increase the computational cost of the algorithm. In some cases, cycles raise a lot the number of necessary steps to arrive at the solutions, without contributing in the research of the strategy (this is the case in the comparison between the nets *bc* and *bc2*). We are currently working to develop a preprocessing of the net, in order to identify these inactive parts that may not be considered in the search for a strategy.

## 4.3 Finding a strategy on the marking graph

In this section we use the marking graph to find a strategy for the user, when its goal is an LTL-X formula, the user has no memory, and the system is 1-safe. LTL-X is the fragment of LTL [106] in which the X operator has been removed; other works, such as [51, 87, 116], studied this logic on Petri nets. We identify the set of atomic propositions as the set of places of the net; a proposition $p$ is true in a marking $m$ iff $p \in m$. Given an LTL-X formula $\psi$, and a play $(\rho, \delta)$, we can evaluate $\psi$ on a maximal refinement $\delta'$ of $\delta$ by considering the sequence of markings corresponding to the B-cuts in $\delta'$. A play $(\rho, \delta)$ is winning for the user when the goal is $\psi$ iff $\psi$ holds in all the maximal refinements of $\delta$.

Through LTL-X, we can express both safety and reachability goals. Referring to the net in Fig. 4.6, described in Ex. 41, a goal for the user could be to always avoid prison (place $p$), and to finish successfully at least a robbery (place $w$). This can be expressed by the formula $G\neg p \wedge Fw$. Also the goal considered in the previous section could be easily translated in an LTL-X formula: if $t$ is the target transition, we can split $t$ in a sequence of two transitions, with a place $p_t$ in the middle. The problem studied in the previous section can then be expressed with the formula $Fp_t$.

Using the marking graph allows us to consider techniques developed for infinite games on finite graphs, that have been widely studied in the literature for decades (see for example [98]). However, many of them consider synchronous models, where each player has its turns to move. In [9], the authors defined a graph based structure, called *concurrent game structure*, allowing for some kinds of asynchronous behaviour. This structure is an extension of a Kripke model, and was introduced for model-checking the game-based temporal logic ATL (Alternating-time Temporal Logic). In Sec. 4.3.1 we recall some formal definitions for concurrent game structures, and we show that when we consider strategies based on stable parts of markings, the game on Petri nets can be translated into a game on concurrent game structures. Finally, in Sec. 4.3.2 we provide a method for finding a winning strategy when the user has no memory, and its strategy can use the information given by stable parts of markings on a 1-safe system in which implicit information has been added. Most of the results in this section are also in [4].

### 4.3.1 Relation between Petri nets and concurrent game structures

We recall the formal definition in [9] of *turn-based asynchronous games* played on a concurrent game structure (CGS). This is a special case in the general context of CGS, that allows to model situations in which the players move asynchronously (although it still imposes a total order between transitions that, as we will see in Ex. 48, may lead to unwanted situations when modelling concurrent asynchronous systems). In [9] the authors considered only the case of full observability and full memory. Subsequent works such as [115, 29] extended the definitions to include other cases. Since in this section we are also interested in the case of no memory and partial observability, we consider some of the adapted definitions in [115].

In this section, the goal of the user is an LTL-X formula. The problem of finding a winning strategy for the user can be interpreted as a special case of model-checking an ATL* formula. ATL* is an extension of ATL including LTL. Checking whether the user is able to enforce a behavior specified by an LTL-X formula $\psi$ can be translated into the problem of verifying whether the system satisfies the corresponding ATL* formula $\langle\langle user\rangle\rangle\psi$ (the meaning of the formula will be explained later in this section). This fragment of ATL* was previously considered in [79] and denoted with 1ATL-X.

## Formal definitions

**Definition 23.** *A* turn-based asynchronous game structure *is a tuple* $G = \langle n, Q, \Pi, w, d, \tau, \sim\rangle$*, where*

- $n \geq 2$ *is the number of* players*. Every player is identified with a number 1,..., n. Player n represents the* scheduler*. At every turn, the scheduler selects one of the other players.*
- $Q$ *is a finite set of* states*.*
- $\Pi$ *is a set of* propositions*.*
- $\forall q \in Q$*,* $w(q) \subseteq \Pi$ *is the set of propositions that are true in* $q$*.*
- $\forall a \in \{1, ..., n\}, q \in Q$*,* $d_a(q) \in \mathbf{N}$ *is the number of moves available for the player* $a$ *in the state* $q$*. Every move is identified with a number* $1, ..., d_a(q)$*. For every state* $q \in Q$*,* $d_n(q) = n - 1$*.*
- *For every state* $q \in Q$*,* $D(q)$ *is the set* $\{1, ..., d_1(q)\} \times ... \times \{1, ..., d_n(q)\}$ *of* move vectors*.*
- $\tau$ *is the* transition function*. For every* $q \in Q$ *and vector move* $\langle j_1, ..., j_n\rangle$*,* $\tau(q, j_1, ..., j_n) \in Q$ *is the state where the system goes if from state* $q$*, every player* $a \in \{1, ..., n\}$ *chooses move* $j_a$*. For all move vectors* $\langle j_1, ..., j_n\rangle$*,* $\langle j'_1, ..., j'_n\rangle$*, if* $j_n = j'_n = a$ *and* $j_a = j'_a$*, then* $\tau(q, j_1, ..., j_n) = \tau(q, j'_1, ..., j'_n)$*, for every* $q \in Q$*.*
- *For each* $a \in \{1, ..., n\}$*,* $\sim_a$ *is an equivalence relation specifying which states are indistinguishable for player* $a$*.*

From now on with CGS we will refer to the structure in Def. 23. Given an initial state $q_0$, an infinite computation $\sigma = q_0 q_1 ...$ on a turn-based asynchronous game structure is an infinite sequence of states such that for each $i \geq 0$, there is a move $d_i$ with $\tau(q_i, d_i) = q_{i+1}$.

We denote with $Q^+$ the set of all the finite prefixes of the computations $\sigma$ in the concurrent game structure and with $\sigma_i$ the prefix of $\sigma$ ending at the $i$-th state. A *strategy* for a player $a$ is a function $f_a : Q^+ \to \mathbb{N}$ such that $f_a(\sigma_i) \leq d_a(q)$, where $q$ is the last element of $\sigma_i$. In addition, for each pair of sequences $\sigma_i = q_0 q_1 ... q_i$, $\sigma' = q'_0 q'_1 ... q'_i$, if $q_j \sim_a q'_j$ for $j \in \{0, ..., i\}$, then $f_a(q_1) = f_a(q_2)$.

The strategy is *memoryless* if, and only if, for every pair of prefixes $\sigma_i, \sigma_j$ ending with the same state, $f_a(\sigma_i) = f_a(\sigma_j)$. We say that a computation $\sigma$ follows a strategy $f_a$ iff, for every prefix $\sigma_i$ of a computation, player $a$ chooses the move $f_a(\sigma_i)$. Let $A$ be a set of players and $F_A = \{f_a : a \in A\}$ a set of strategies $f_a$, one for each player in $A$. We denote with $out(q, F_A)$ the set of computations starting from state $q$ in which each player in $A$ followed its strategy in $F_A$. We can include some *fairness constraints* to the CGS, in order to ignore some computations.

**Definition 24.** *A* fairness constraint *is a pair $\langle a, c \rangle$, where $a$ is a player, and $c$ is a function that for every state $q \in Q$ selects a subset of moves available for $a$ in $q$.*

Let $\sigma$ be an infinite computation and $q_i$ its $i - th$ state. A fairness constraint $\langle a, c \rangle$ is *enabled* in $q_i$ if $c(q_i) \neq \emptyset$; $\langle a, c \rangle$ is *taken* in $q_i$ if there is a vector move $\langle j_1, ..., j_n \rangle$ with $j_a \in c(q_i)$ and $q_{i+1} = \tau(q_i, \langle j_1, ..., j_n \rangle)$. If $a < n$ we also require that $j_n = a$.

A computation $\sigma$ is *weakly fair* with respect to a fairness constraint $\langle a, c \rangle$ if $\langle a, c \rangle$ is not enabled in infinitely many positions of $\sigma$ or if it is taken infinitely many times in $\sigma$.

**The logic ATL*** The logic ATL* extends CTL* by considering a larger number of paths quantifiers. Both logics are defined and model-checked on Kripke structures, in case of ATL* extended by modelling the interaction of a group of agents. The only path quantifiers allowed in CTL* formulas are the universal and the existential quantifier (resp. a formula is satisfied in all the paths or a formula is satisfied in at least one path); in ATL*, quantifiers may be defined by groups of agents. For example, let $A$ be a group of agents interacting on the structure, and $\psi$ be an LTL formula. With an ATL* formula we can require that the agents in $A$ can coordinate their action on the structure to satisfy $\psi$ despite of the behaviours of the other agents. This is expressed by the formula $\langle\langle A \rangle\rangle \psi$. The universal quantifier in CTL* can be expressed as $\langle\langle\rangle\rangle \psi$, namely it is associated to the case in which a formula is required to be verified whatever the agents decide to do; the existential quantifier, at least for agents with full information and full memory, can be expressed with quantifiers including all the agents.

In this paragraph we consider the formalization in [115] for ATL* formulas. Let $G$ be a CGS, $p$ be an atomic proposition, and $A$ be a set of agents. The syntax of an ATL* formula $\phi$ is expressed as follows.

$$\phi ::= p \mid \top \mid \neg\phi_1 \mid \phi_1 \vee \phi_2 \mid \langle\langle A \rangle\rangle \phi_1 \mid \phi_1 \mathcal{U} \phi_2 \mid X\phi_1$$

with $\phi_1, \phi_2$ ATL* formulas. In the syntax above, the symbol $\mathcal{U}$ stands for 'until', and $X$ stands for 'next state'. As usual in temporal logics, formulas use the two additional temporal operators $F$ ('eventually') and $G$ ('always'), where $F\phi = \top \mathcal{U}\ \phi$, and $G\phi = \neg F \neg \phi$.

The semantics of ATL* extends the semantics of CTL* (a formalization of which can be found in [42]). In addition, let $q_0$ be the initial state in $\sigma$, $\sigma \models \langle\langle A \rangle\rangle \phi_1$ if there is a set of strategies $Y$, one for each agent in $A$, such that, for each computation $\sigma'$ in $out(q_0, Y)$, $\sigma' \models \phi_1$.

## Construction of a CGS from a game on Petri nets

Let $\Sigma = (P, T, F, m_{in})$ be a net system, $T_u$ the set of controllable transitions, $T_{env} = T \setminus T_u$ the set of uncontrollable transitions, and $P_O$ the set of observable places. Assume that the transitions in $T_{env}$ need to satisfy a progress constraint. In order to put our game in relation with the one on concurrent game structures, we need to assume that the user cannot discriminate conditions of the unfolding, but only places on the 1-safe system. Among the definitions of observations defined in Sec. 4.1, this happens when the user has no memory, and with the notion of memory defined in the paragraph starting on page 85. Here, we focus on the case of no memory. Since in this case, for each pair of B-cuts $\gamma_1, \gamma_2$, if $\lambda(\gamma_1) = \lambda(\gamma_2)$ then $\gamma_1 \equiv \gamma_2$, we will consider the relation $\equiv$ on the elements of $[m_{in}\rangle$.

We construct a CGS (denoted $G_\Sigma$) where the elements are defined as follows.

- $\{u, env, s\}$ is the set of *players*; specifically $u$ denotes the user, $env$ the environment, and $s$ the scheduler.
- $[m_{in}\rangle$ is the set of *states* of the system. Each state is a reachable marking of $\Sigma$; $m_{in}$ is the initial state.
- The set of *propositions* on the concurrent game structure is given by the set of places $P$ of the net; for each state $m \in M$, for each proposition $p \in P$, $p$ is true in $m$ iff $p \in m$.
- For $i \in \{u, env\}$, for each $m \in [m_{in}\rangle$, $d_i(m)$ is given by the transitions enabled in $m$ belonging to player $i$. Specifically, for each $m \in [m\rangle$, $d_u(m)$ is the set of controllable transitions enabled in $m$, plus a move remaining in the same state, from now on denoted with $\epsilon$; $d_{env}(m)$ is the set of uncontrollable transitions enabled in $m$; if no such transition is enabled, $d_{env}(m) = \{\epsilon\}$; $d_s(m) = \{u, env\}$. For each state $m \in [m_{in}\rangle$, a *move vector* enabled in $m$ is a tuple $\langle j_u, j_{env}, j_s \rangle$, where $j_i \in d_i(m)$, $i \in \{u, env, s\}$.
- $\tau$ is the *transition function*; for each $m \in [m_{in}\rangle$ and vector move $\langle j_u, j_{env}, j_s \rangle$ enabled in $m$, $\tau(m, j_u, j_{env}, j_s)$ is the state reached on $\Sigma$ when transition $j_{j_s}$ occurs.

- $\equiv \subseteq [m_{in}\rangle \times [m_{in}\rangle$ is an *equivalence relation* that specifies which states are indistinguishable by the user (player $u$). In our setting, we are not interested in the observations of the other players

An infinite computation $\sigma = m_0 t_0 m_1 t_1 ....$ on $G_\Sigma$ is an infinite sequence of states and transitions starting from $m_{in}$ such that for each $m_i$, $m_{i+1}$ is the marking reached from $m_i$ after the occurrence of $t_i$. In order to exclude the computations associated with unfair runs on the net (for the progress constraint or for condition (2) of Def. 21), we need to add to $G_\Sigma$ some *weak fairness constraints.*

1. There must be no point in the computation from which the scheduler always selects the same player: this is expressed by the constraints $\langle s, c_u\rangle$ and $\langle s, c_{env}\rangle$, where $\forall m \in [m_{in}\rangle\ c_i(m) = \{i\}$, $i \in \{u, env\}$. This requirement guarantees that if the user wants to move, at some point it has the possibility of doing it, and that the environment can progress.

2. For each uncontrollable transition $t$, let $\#t : [m_{in}\rangle \to 2^{T_{env}}$ be a function such that $\#t(m) = \emptyset$ if $t$ is not enabled in $m$, $\#t(m) = \{t\} \cup \{t_i : {}^\bullet t_i \subseteq m \wedge t_i \# t\}$ otherwise; intuitively $\#t$ is the set of transitions in conflict with $t$ at marking $m$ together with $t$ itself. For each $t$ uncontrollable, we add the constraint $\langle env, \#t\rangle$. This point together with point (1) guarantees that uncontrollable transitions satisfy their progress constraint in all the fair computations.

For each state $m \in [m_{in}\rangle$, let $[m]$ be the equivalence class of $m$ with respect to $\equiv$, and $[m_{in}\rangle^+$ the set of finite computations. A memoryless strategy is a function $f : [m_{in}\rangle \to T_u \cup \{\epsilon\}$ such that if $m \equiv m'$, then $f(m) = f(m')$.

Given $\psi \in$ LTL-X, a strategy is *winning* with respect to the goal expressed by $\psi$ iff all the fair computations in which the user follows the strategy satisfy $\psi$.

In general, looking for a winning strategy on the unfolding and on the CGS constructed as described above is not equivalent, as illustrated by the following example.

**Example 48.** *Consider the* 1*-safe system in Fig. 4.5 on page 84, and the goal discussed in Ex. 40, that can be equivalently expressed with the* LTL-X *formula $F(p_0 \wedge p_4) \vee F(p_1 \wedge p_3)$. In Ex. 40 we motivated the non-existence of a winning strategy, and the problems in defining the strategy on the markings.*

*The ambiguity of defining a strategy on markings, when part of the information cannot actually be used causes problems when we look for a strategy on the marking graph or on the associated $G_\Sigma$. Due to the asynchronicity defined in the* CGS *we do not know in which state the user will be selected to move; however, when the user is selected, it can choose between transition $t_2$ and transition $t_3$ based on the current marking. In order to prevent this* *situation, we need to define the markings $\{p_0, p_2\}$ and $\{p_1, p_2\}$ as indistinguishable for the user, for example by defining the equivalence relation for observations on stable parts of markings.*

Motivated by the example above, we now prove the equivalence of games on the unfolding and derived CGS, when $\equiv$ is defined on stable parts of markings, namely for each pairs of markings $m_1, m_2$, $m_1 \equiv m_2$ iff $m_1^s \cap P_O = m_2^s \cap P_O$.

Given a computation $\sigma = m_{in} m_1 ... m_n ...$ (finite or infinite) we can construct a new sequence $\mu(\sigma)$ in which all the consecutive states such that $m_i = m_{i+1}$ have been identified. Given an infinite sequence $\sigma$, $\mu(\sigma)$ can be also infinite, if there is no state $m_i \in \sigma$ such that for each $m_j : j > i$ $m_j = m_{j+1}$, or finite otherwise.

**Lemma 17.** *Let $\psi$ be an* LTL-X *formula, and $\sigma_1$ and $\sigma_2$ be two infinite computations such that $\mu(\sigma_1) = \mu(\sigma_2)$. $\sigma_1$ satisfies $\psi$ iff $\sigma_2$ does.*

*Proof.* If the operator $X$ is not allowed, then the validity of $\psi$ depends only on the sequence of distinct states in computations; since, by hypothesis, $\mu(\sigma_1) = \mu(\sigma_2)$, the thesis follows. $\square$

**Relation between plays in the two models**

Let $\Sigma = (P, T, F, m_{in})$ be a 1-safe system, and $G_\Sigma$ the derived game structure.

We now prove some propositions that show the relations between the plays on the unfolding and the infinite fair computations on $G_\Sigma$.

Let $\sigma = m_0 m_1 \cdots$ be a fair computation on $G_\Sigma$. For every pair of consecutive states $m_i, m_{i+1}$, such that $m_i \neq m_{i+1}$, the move on the concurrent game structure is associated, by construction, with a transition $t$ in $\Sigma$, such that $m_i[t\rangle m_{i+1}$.

Given $\sigma$, we construct a run $\rho$ on the unfolding of $\Sigma$ and a sequence $\delta$ of B-cuts, starting from the initial cut and adding the events associated with the moves that bring from a state to the next one in the same order that the states have in $\sigma$, and the cuts following those events. The run derived in this way will be denoted by $(\rho, \delta)[\sigma]$.

**Lemma 18.** *The run $(\rho, \delta)[\sigma]$ is a play on* UNF$(\Sigma)$.

*Proof.* We need to prove that there is no uncontrollable event $e \notin \rho$ such that $\rho \cup \{e\}$ is a run on the unfolding. By contradiction, assume that we can find such an event $e$. Then, there is $i \in \mathbb{N}$ such that, on $G_\Sigma$, the move $\lambda(e) = t$ is enabled in every state $m_j \in \sigma : i < j$, but is never chosen. The reason cannot be that the environment player is never selected by the scheduler, otherwise, the constraint $\langle s, c_{env} \rangle$ would not be satisfied, and $\sigma$ would not be a fair computation. Hence the move $t$ is available in every state $m_j$ for player *env*, but it never chooses it. Then, the computation does not satisfy the fairness constraint $\langle env, \#t \rangle$.

Hence, $(\rho, \delta)[\sigma]$ constructed in this way is a play on the unfolding. $\square$

With a similar idea, for each strategy $\alpha$ defined on stable and observable parts of markings, we can associate every play on $\textsc{unf}(\Sigma)$ consistent with $\alpha$ with a set of infinite fair computations on $G_\Sigma$: let $(\rho, \delta)$ be a play consistent with $\alpha$ on the unfolding. As first step, we consider all the possible sequentializations of events included between the initial cut $\gamma_0$ and the next cut $\gamma_1 \in \delta$. For each of these linearizations we can find a prefix of a computation on the concurrent game structure by executing on it, from the state corresponding to the initial marking, all the events in the order given by the linearization; then, we extend all these prefixes by considering the successive pairs of consecutive cuts, and the sequentializations of the events between them. If the play on the 1-safe system reaches a deadlock, then the only possibility in the associated concurrent game structure is to execute the transition that remains in the same state infinitely often.

In this way, we obtain a set of infinite computations associated with a play $(\rho, \delta)$, denoted by $\Lambda(\rho, \delta)$.

**Lemma 19.** *Let $\alpha$ be a strategy on $\Sigma$, and $(\rho, \delta)$ be a play consistent with $\alpha$ on $\textsc{unf}(\Sigma)$. For every computation $\sigma \in \Lambda(\rho, \delta)$ on $G_\Sigma$, there is at least one fair computation $\sigma' : \mu(\sigma') = \mu(\sigma)$.*

*Proof.* The set of constraints of the scheduler guarantees that no player is neglected forever. If $\rho$ is finite then every $\sigma \in \Lambda(\rho, \delta)$ is fair: the only move available for the environment is the empty move, that keeps the system in the same state; when the scheduler selects the user, it can decide to execute the move that keeps the structure in the same state.

In addition, when $\rho$ is finite, also the constraints on the environment are satisfied: after a finite number of states there is no uncontrollable enabled transition, therefore those constraints are infinitely not enabled.

We now consider the case in which $\rho$ is infinite. Since a play on the unfolding must be fair with respect to uncontrollable transitions, in the computation $\sigma$ the fairness constraint $\langle s, c_{env} \rangle$ must be satisfied. If $\rho$ has infinitely many controllable events, then the fairness constraint $\langle s, c_u \rangle$ is also satisfied by construction; otherwise, since the user has always the possibility not to move in the system, we can construct a sequence $\sigma' : \mu(\sigma) = \mu(\sigma')$ with some repeated states, that represent points of the sequence in which the user was selected by the scheduler in $G_\Sigma$, but the state of the system did not change.

The set of fairness constraints for the environment must be satisfied: by contradiction, we assume that there is a function $\#t$, such that $\langle env, \#t \rangle$ is not satisfied in $\sigma'$; this means that $\langle env, \#t \rangle$ is not enabled in a finite number of states in $\sigma$, but it is taken only a finite number of times. This means that there is an uncontrollable event $e \notin \rho$ in the net that is enabled in all the cuts compatible with $\delta$, except for a finite set; since $\rho$ is infinite, there must be a cut $\gamma \in \delta$ such that $e$ is enabled in $\gamma$ and in all the cuts $\gamma_j \in \rho : \gamma_j > \gamma$; hence $\rho \cup \{e\}$ is a run on UNF$(\Sigma)$. This is in contradiction with the hypothesis that $(\rho, \delta)$ is a play. □

**Lemma 20.** *Let $\psi$ be an* LTL-X *formula. A computation $\sigma$ satisfies $\psi$ iff $(\rho, \delta)[\sigma]$ does. Analogously, a play $\rho$ consistent with a strategy $\alpha$ according to an increasing sequence of B-cuts $\delta$ satisfies $\psi$ iff all the computations in $\Lambda(\rho, \delta)$ do.*

*Proof.* Constructing the play $(\rho, \delta)$ from $\sigma$, $\delta$ is by construction a maximal refinement, in which events are ordered in the same way as in $\sigma$.

Each element $\sigma \in \Lambda(\rho, \delta)$ is associated with a maximal refinement of $\delta$ by construction. If all the elements in $\Lambda(\rho, \delta)$ satisfy $\psi$, then also all the refinements of $\delta$ do. Vice versa, if there is a $\sigma \in \Lambda(\rho, \delta)$ that does not satisfy $\psi$, also the $\delta'$ maximal refinement of $\delta$ associated with $\sigma$ does not. □

**Relations between winning strategies in the two models**

Let $\alpha : \mathsf{Obs} \to 2^{T_u}$ be a strategy for the user on $\Sigma$. We call $f_\alpha$ a strategy on $G_\Sigma$ constructed from $\alpha$ as follows.

1. If $\alpha([m])) \neq \emptyset$, and $t$ is a transition in $\alpha([m])$ arbitrarily chosen, then $f_\alpha(m) = t$.
2. $f_\alpha(m) = \epsilon$ otherwise.

Analogously, let $f : [m_{in}\rangle \to T_u \cup \{\epsilon\}$ be a strategy on $G_\Sigma$; we define the derived strategy $\alpha_f$ on $\Sigma$ as follows:

1. $\alpha_f([m]) = \{f(m)\}$, if $f(m) \neq \epsilon$.
2. $\alpha_f(m) = \emptyset$, otherwise.

**Theorem 8.** *Let $\psi$ be an* LTL-X *formula, $\Sigma$ a 1-safe system, and $G_\Sigma$ the derived* CGS. *A winning strategy exists on $\Sigma$ for $\psi$ iff a winning strategy exists on $G_\Sigma$.*

*Proof.* As first step, we show that if $\alpha$ is a winning strategy on $\Sigma$ for $\psi$, then $f_\alpha$ is a winning strategy for $\psi$ on $G_\Sigma$.

Let $out(m_{in}, f_\alpha)$ be the set of fair computations starting from $m_{in}$ that the user enforces by following the strategy $f_\alpha$. For any $\sigma \in out(m_{in}, f_\alpha)$, we show that $(\rho, \delta)[\sigma]$ satisfies the conditions of Def. 21, and therefore is consistent with $\alpha$: (1) let $e \in E_u \cap \rho$ be an event controllable by the user. By construction, $\delta$ is a maximal refinement, hence there must be two cuts $\gamma_j$ and $\gamma_{j+1}$ such that $e$ is the only event between them. By construction, if $e$ was added to the run after $\gamma_j$, there must be a state $m \in \sigma : \lambda(\gamma_j) = m$ such that $f_\alpha(m)) = \lambda(e)$. By construction of the strategy, $\lambda(e) \in \alpha([m])$. (2) By contradiction, assume that the user is finally postponed; then there is a cut $\gamma \in \delta$ such that for each cut $\gamma_j$ compatible with $\delta$ such that $\gamma_j > \gamma$, $\alpha([\lambda(\gamma_j)]) \neq \emptyset$. By construction, there must be a state $m_i \in \sigma : \lambda(\gamma) = m_i$ such that $f(m_j) \neq \emptyset$ for each $m_j, j > i$; then $\sigma$ cannot be fair with respect to $\langle s, c_u \rangle$. $(\rho, \delta)[\sigma]$ is a play, and it is consistent with the strategy by construction, hence it is a winning play for the user and it satisfies the property expressed by $\psi$. Hence also $\sigma$ satisfies it on the concurrent game structure.

As second step, let $f$ be a winning strategy on $G_\Sigma$. We show that $\alpha_f$ is a winning strategy on $\Sigma$.

Let $(\rho, \delta)$ be a play consistent with $\alpha_f$. By contradiction, we assume that the formula $\psi$ is not satisfied on UNF($\Sigma$), and that the play $(\rho, \delta)$ testifies it. Then there must be a maximal refinement $\delta'$ of $\delta$ that does not satisfy the formula. $\delta'$ is associated with an infinite sequence of states $\sigma$ on $G_\Sigma$. If $\sigma$ is a fair computation on $G_\Sigma$, then it is also a computation consistent with the strategy: let $t$ be a user's move in $m_i \in \sigma$. By construction, there must be $e : \lambda(e) = t$ in $(\rho, \delta)$ such that $t \in f(\lambda(\gamma_i))$, where $\gamma_i \in \delta'$ and $\lambda(\gamma_i) = m_i$.

If $\lambda$ is not fair, we know by Lemma 19 that we can construct a fair computation $\sigma'$ such that $\mu(\sigma) = \mu(\sigma')$. In addition, from the proof of Lemma 19 we know that the only reason why $\sigma$ can be unfair is that the user is finally neglected during the play. This cannot happen if the user has the strategy finally non-empty, otherwise the play on the net would not be consistent with this strategy (by definition the strategy selects only enabled events). Then, there is an infinite number of states in the sequence in which $f$ selects only the transition that keeps the system in the same state. We add a copy of these states in the position coming immediately after them. This computation $\sigma'$ is fair with respect to the user, therefore it is a play consistent with the strategy on the concurrent game structure and by hypothesis is winning. By construction $\mu(\sigma) = \mu(\sigma')$, hence Lemma 17 guarantees that also $\sigma$ respect $\psi$. □

### 4.3.2 An algorithm for finding a winning strategy

In this section we describe a method to find a winning strategy for the user on the 1-safe system, if one exists, when the user has no memory, and the strategy is defined on observable and stable parts of markings including the implicit places. In Sec. 4.3.1 we showed that this is equivalent to find a strategy on the derived game structure, and we mentioned that this problem can be translated into the problem of model-checking a 1 ATL* formula. However, to our knowledge, none of the model-checkers developed for ATL can be directly applied to solve the problem as we stated. This is because we want to model check a fragment of ATL*, whereas most of the tools only support ATL, and because even those tools supporting ATL* do not allow to specify the observability and the fairness constraints that we require. Among the most prominent tools, MOCHA [10] works only for synchronous systems and allows neither partial observability nor any kind of fairness constraint; MCMAS [92] does not support partial observability and consider only the so called *unconditional fairness*, where each fair computation must cross infinitely often some subsets of states; finally the recent STV+ [89] supports a special case of partial observability, in which the agents can observe only their local states, and does not support any fairness constraint.

In this section we focus on the research of a winning strategy on the derived game structure.

Let $\Sigma$ be a 1-safe system. We want to find a strategy on $G_{\Sigma_S}$, where $\Sigma_S$ is the extension of $\Sigma$ with implicit observable places. In order to reach this goal, we proceed as follows.

1. We construct the marking graph of $\Sigma$, $\textsc{mg}(\Sigma)$, and we interpret it as a Kripke model.

2. We compute all the regions of $\textsc{mg}(\Sigma)$ associated with the unions and the complements of observable places. We update the set of propositions true in each state as follows. For each region $r$, for each state $m \in r$, we add the boolean combination of places associated to $r$ as local proposition that is true in $m$. With this operation we obtain a Kripke model isomorphic to $G_{\Sigma_S}$.

3. For each marking $m$, we compute its stable part $m^s$, and define the relation $\equiv$ as follows. Let $P'_O$ be the set of observable places in $P'$; $m_1 \equiv m_2$ iff $m_1^s \cap P'_O = m_2^s \cap P'_O$.

4. From each state $m$, we compute which controllable transitions are enabled in $[m]$ and we remove from $G_{\Sigma_S}$ the controllable transitions which are enabled in $m$ and not in $m^s \cap P'_O$. We denote as $M^r$ the subset of states that are reachable in $G_{\Sigma_S}$ once that this step has been executed. Since some places may not belong anymore to any reachable marking, the associated propositions are false in all the states of $G_{\Sigma_S}$.

An example of this construction is in Ex. 49.

**Example 49.** *Let $\Sigma$ be the 1-safe system in Fig. 4.15, except for the green places ($p_1^c$ and $p_2^c$). Let $p_1$ and $p_2$ be the observable places. In the right part of the figure, we show $\textsc{mg}(\Sigma)$ (in black and grey), and the structure of $G_{\Sigma_S}$ (in black). In this example, the extended net $\Sigma_S$ has only two places more than $\Sigma$, represented in green in the figure, associated with the complements of the two observable places. In Table 4.2, each state of $\textsc{mg}(\Sigma_S)$ is associated with its stable and observable part. As observed in Ex. 43, there is no marking where $p_2$ is stable. For this reason, a winning strategy for the user can*

Table 4.2: Table representing the stable and observable part of each state of the marking graph in Fig. 4.15

| States | Propositions |
|---|---|
| $m_1$, $m_2$ | $p_1$ |
| $m_5$ | $p_1$, $p_2^c$ |
| $m_3, m_6, m_8, m_{11}$ | $p_1^c$ |
| $m_4, m_7, m_9, m_{10}, m_{12}, m_{13}$ | $p_1^c, p_2^c$ |

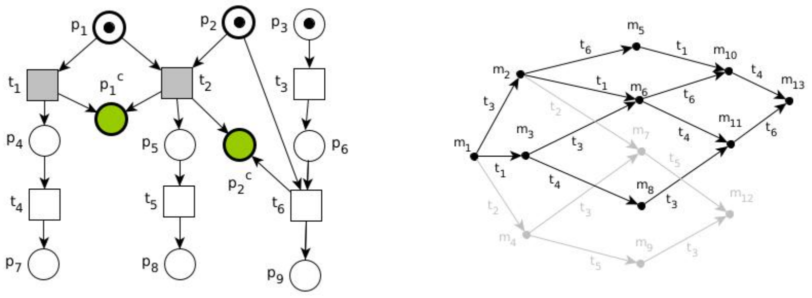


Figure 4.15: A 1-safe system, its marking graph (black and grey) and its associated game structure (black)

*never depend from the occurrence of $t_2$, and when we construct $G_{\Sigma_S}$ we can remove all the occurrences of this transition as well as of the transitions causally depending from it (as explained in point 4 of the procedure above).*

The method that we propose can be summarized in the following steps, that will be explained in detail in the next subsection.

1. Among the set of potential winning strategies, we select a strategy $f$ for $G_{\Sigma_S}$, and we prune $G_{\Sigma_S}$ accordingly.

2. From the pruned $G_{\Sigma_S}$ and the strategy $f$, we construct a Kripke model $K(G, f)$ that allows us to verify fairness constraints without having them externally specified. To the same aim, we construct a formula $\psi'$ from $\psi$ such that the user has a winning strategy for $\psi$ on $G_{\Sigma_S}$ if, and only if, $\psi'$ is verified in the initial state on $K(G, f)$.

3. We check if the strategy is winning with the help of an LTL model-checker; if it is, we stop and return that strategy, otherwise we mark the current strategy and all those equivalent to it as checked, and we repeat the steps.

We say that two strategies $f$ and $f'$ are *equivalent* if the set of states reachable from the initial state of $K(G, f)$ is equivalent to the set of states reachable from the initial state of $K(G, f')$. This may happen because some states become unreachable after the pruning process. In this case, $f$ is winning iff $f'$ is.

## Encoding fairness constraints into the Kripke model

Let $G_{\Sigma_S} = (\{u, env, s\}, M^r, m'_{in}, P', d, \tau, \equiv)$ be a derived game structure with fairness constraints, and $f$ be a strategy on $G_{\Sigma_S}$. We modify $G_{\Sigma_S}$ and construct a Kripke model $K(G, f)$ as follows. Let $P^\alpha$ be the set of atomic propositions of $K$. $P^f$ includes all the elements in $P'$. In addition, for each transition $t \in T_{env}$, we add an element to $P^f$. Abusing the notation, these propositions will be denoted with the name of their associated transitions. Finally $P^f$ includes the two propositions $env$ and $u$.

For each state $m \in M^r$, we delete all the outgoing controllable transitions that are not selected by $f$ in $[m]$. We split all the other arcs by adding for each of them an intermediate state. Let $t$ be a transition outgoing from $m$ in $G_{\Sigma_S}$, and $l$ be the intermediate state to add. All the propositions true in $m$ and all those associated with uncontrollable transitions not enabled at $m$ are true in $l$. In addition, if $t$ is controllable or $f([m]) = \emptyset$, $u$ is true in $l$; if $m$ does not enable any transition in $T_{env}$, $env$ is true in $l$; if $t \in T_{env}$ all the propositions in $\#t$ are true in $l$.

Finally, for each state $m$ in $M^r$ if $m$ does not enable any transition in $T_{env}$ and $f([m]) = \emptyset$, we add a state $l$ and two transitions so to create a cycle between $m$ and $l$. In $l$ the true propositions are those true in $m$ and all those in $P^f \setminus P'$.

**Example 50.** *Consider the* 1*-safe system and its concurrent game structure represented in Fig. 4.15. Assume that the goal of the user in* $\Sigma_S$ *is* $F(p_7 \wedge p_9)$*, and that the user follows the strategy* $\alpha$ *such that* $\alpha([\{p_1, p_2^c\}]) = \{t_1\}$*,* $\alpha([m]) = \emptyset$ *otherwise. The Kripke model* $K(G, f)$ *encoding this strategy and the fairness constraints is represented in black in Fig. 4.16. The grey part represents the parts of the game structure* $G_{\Sigma_S}$ *that must be deleted. Each transition in* $K(G, f)$ *is doubled with respect to the same transition in* $G_\Sigma$*. The states dividing each transition (*$m'_1, m'_2, m'_5$ *etc.) are used to store the information about fairness constraints. The true propositions in each of these new states are in Table 4.3. We can use this Kripke model to check if* $\alpha$ *is winning, as formalized in Lemma 21.*

**Lemma 21.** *Let* $\psi$ *be an* LTL-X *formula,* $G_{\Sigma_S}$ *be a derived structure,* $f$ *be a strategy on* $G_{\Sigma_S}$ *and* $K(G, f)$ *the derived Kripke model.* $f$ *is winning on* $G_\Sigma$ *if, and only if,* $\psi' = (\bigwedge_{x \in \{u, env\}} GF\, x \wedge \bigwedge_{t_j \in T_{env}} GF t_j) \rightarrow \psi$ *holds in the initial state on* $K(G, f)$*.*

*Proof.* We associate to every infinite computation $\sigma^f$ on $K(G, f)$ an infinite computation $\sigma$ on $G_{\Sigma_S}$ by removing from $\sigma^f$ all the states that are not in $M^r$.

Table 4.3: True propositions in the states added during the construction of $K(G,f)$

| New states | True propositions |
|---|---|
| $m'_1$ | $p_1, p_2, p_3, u, env, t_3, t_4, t_5, t_6$ |
| $m'_2$ | $p_1, p_2, p_6, u, env, t_3, t_4, t_5, t_6$ |
| $m'_5$ | $p_1, p_2^c, p_9, u, t_3, t_4, t_5, t_6$ |
| $m'_{10}$ | $p_1^c, p_2^c, p_4, p_9, u, env, t_3, t_4, t_5, t_6$ |
| $m'_{13}$ | $p_1^c, p_2^c, p_7, p_9, u, env, t_3, t_4, t_5, t_6$ |

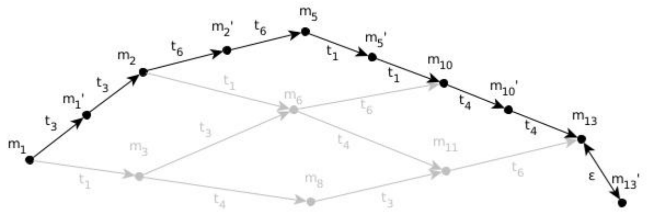


Figure 4.16: A Kripke model derived from the game structure in Fig. 4.15.

We first show that for each infinite path $\sigma^f$ in $K(G,f)$, $\sigma^f$ satisfies

$$(\bigwedge_{x\in\{u,env\}} GF\,x \wedge \bigwedge_{t_j\in T_{env}} GFt_j)$$

iff there is a computation $\sigma'$ in $G_{\Sigma_S}$ such that $\mu(\sigma) = \mu(\sigma')$, and $\sigma'$ satisfies all the fairness constraints in $G_{\Sigma_S}$. Assume such a $\sigma'$ exists, then both the user and the environment must have been selected infinitely often by the scheduler. If $\sigma'$ is obtained through the occurrence of infinitely many uncontrollable and controllable transitions, then by construction $\bigwedge_{i\in\{u,env\}} GFi$ holds in $\sigma^f$. If there is a finite number of uncontrollable transitions, there must be an index $i$ such that for each $\sigma'_j : j > i$, $\sigma'_j$ does not enable any uncontrollable transition. Since $\mu(\sigma) = \mu(\sigma')$, this property must hold also for $\sigma$. By construction, for each state in $\sigma$ where no uncontrollable transition is enabled, there is a state in $\sigma^f$ where $env$ is true. Analogously for $u$ if no controllable transition is enabled from a certain state on. In addition, there may be a finite number of controllable transitions also if the strategy for the user does not select any transition from a certain state on. By construction, for each of these states in $\sigma$ there is a state on $\sigma^f$ where $u$ is true. Vice versa, if there is no fair $\sigma'$, also the formula $\bigwedge_{i\in\{u,env\}} GFi$ does not hold on $\sigma^f$, since there are no other cases than those discussed above in which $u$ or $env$ are true in a state. If $\sigma$ satisfies also the constraints for the environment, then the formula $\bigwedge_{t_j\in T_{env}} GFt_j$ must hold in $\sigma^f$, because by construction, for each $t_j \in T_{env}$, $t_j$ is true in a state $l$ of $\sigma^f$ iff there is a state $m$ in $\sigma$ and a transition $t_i$ bringing to the next state of $\sigma$ and such that $t_j$ is not enabled at $m$ (and therefore its associated fairness constraint is also not enabled) or $t_j \in \#t_i$ (and therefore the constraint is taken in $\sigma$).

As second step, we observe that $\psi$ holds in $\sigma$ iff it holds in $\sigma^f$ (since all the propositions of $\psi$ are in $P$, this can be seen as a Corollary of Lemma 17).

Then, if $\psi'$ holds in $K(G, f)$, in all the fair paths in $G_\Sigma$ that follow $f$, $\psi$ holds, and therefore $f$ is winning on $G_{\Sigma_S}$. Vice versa, if $\psi$ does not hold in $K(G, f)$, there is at least a fair path on $G_\Sigma$ that follows $f$ and such that $\psi$ is not verified, therefore $f$ is not winning on $G_{\Sigma_S}$ □

Lemma 21 allows us to prove the correctness of the algorithmic procedure proposed at the beginning of this section to find a winning strategy, if one exists.

### 4.3.3 Complexity

**Theorem 9.** *The complexity of finding a winning strategy is at least* PSPACE *and at most* EXPSPACE *in the dimension of the net.*

*Proof.* In order to prove that finding a winning strategy is at least PSPACE, we show that this problem is at least as hard as the reachability problem. This consists in deciding whether a given marking is reachable from the initial marking, and it is proved to be PSPACE-complete for 1-safe nets [41]. Let $\Sigma = (P, T, F, m_0)$ be a net system, and $m \subseteq P$ be a marking. We associate to the reachability problem for $m$ from $m_0$ a game on the net. On $\Sigma$, we assume that all the places are observable, and all the transitions are controllable by the user. Since every transition is controllable, for each marking, the stable part is the entire marking, therefore the strategy is defined on the reachable markings of the net. Let the goal of the user be $F(\bigwedge_{p_i \in m} p_i \wedge \bigwedge_{p_i \in P \setminus m} \neg p_i)$. A winning strategy for the user exists iff $m$ is reachable:

- If $m$ is reachable, then there is a sequence of transitions that brings from $m_0$ to $m$. Let $\sigma = t_0 ... t_n$ be this sequence, and $m_0 m_1 ... m_n$ be the corresponding sequence of markings, so that $m_0[t_0\rangle m_1[t_1\rangle ... m_n[t_n\rangle m$. We can assume that for each $i \neq j$, $m_i \neq m_j$. If this is not the case, we can construct another path that from $m_0$ arrives to $m$ by observing that if $m_l = m_{l+k}$ for some $l, k$, then the sequence of transitions $t_0 t_1 ... t_{l-1} t_{l+k} ... t_n$ also reaches $m$ and the repetition between $m_l$ and $m_{l+k}$ has been deleted. Then a winning strategy is a function $\alpha$ such that $\alpha(m_i) = \{t_i\}$ for each $i \in \{0, ..., n\}$.
- If there is a winning strategy for the user, then by construction the user reaches $m$ after a finite number of steps.

We now prove the upper limit by showing that the algorithm that we proposed is at most EXPSPACE. In general, the marking graph $\textsc{mg}(\Sigma)$ has exponential size with respect to the size of $\Sigma$; computing all the unions and complements of the observable places is also exponential with respect to $\Sigma$. Computing the stable and observable parts of all the markings requires polynomial time with respect to $\textsc{mg}(\Sigma)$. Let $G_{\Sigma_S}$ be the concurrent structure derived from $\Sigma$. By construction, $\textsc{mg}(\Sigma)$ and $G_{\Sigma_S}$ have the same size. In our algorithm we guess a strategy $\alpha$, we construct a Kripke model $K(G, f)$ and we verify an LTL formula. Since constructing a strategy $f$ and $K(G, f)$ is polynomial with respect of $G_{\Sigma_S}$, and model-checking of LTL-X is PSPACE, the algorithm is PSPACE with respect to the size of $\textsc{mg}(\Sigma)$. Since $\textsc{mg}(\Sigma)$ is in general exponentially larger than $\Sigma$, the algorithm is at most EXPSPACE. $\square$

## 4.4 Implementable strategies

In the previous sections we studied how to find a strategy for the user to pursue a certain goal, in case such a strategy exists. When the user follows a strategy, it constraints the behaviour of the system, preventing some of the possible executions. In this section, we discuss how to encode a given strategy in the system itself, namely how to add constraints to the system, so that the only executions allowed in the new systems are those obtained when following the strategy. In some sense, this procedure “corrects” the system that does not necessitate anymore of an external controller in order to satisfy the behaviour imposed by the strategy.

The idea we pursue is this: a strategy is encodable within a P/T system, if its decisions can be represented as extra places, corresponding to causal relations from uncontrollable transitions to controllable transitions. We will then say that a strategy is *implementable* if we can add new places to the given P/T system, so that the maximal runs of the augmented nets are the runs of the plays consistent with the strategy in the original P/T system.

In Sec. 4.4.1, we formalize the idea of *implementable strategy* for bounded P/T systems without self-loops, and we discuss some aspects of the definitions on the basis of some examples. In Sec. 4.4.2, we restrict ourselves to the case in which the strategy is defined on observations without memory, and we propose an algorithm to find an implementation, if one exists, based on region theory.

The results discussed in this section are in [5].

### 4.4.1 Formal definitions and examples

Before giving a formal definition of implementable strategy, we need some auxiliary definitions.

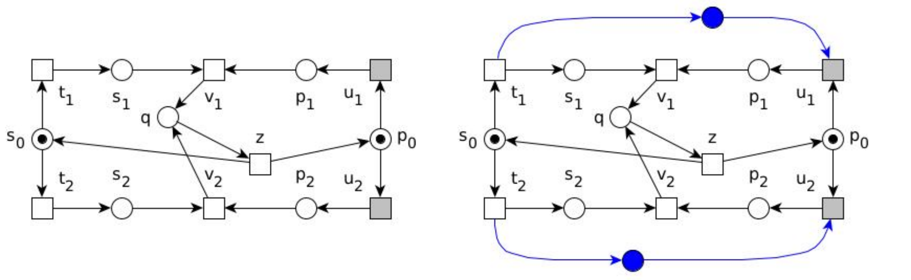


Figure 4.17: The game net of Fig. 4.1 and one of its restrictions.

**Definition 25.** *Let $\Sigma = (P, T, F, W, m_{in})$ be a bounded P/T system. A* restriction *of $\Sigma$ is a P/T system $\Sigma' = (P \cup H, T, F', W', m'_{in})$ such that $F = F'|_{(P\times T)\cup(T\times P)}$, $W(x, y) = W'(x, y)$ for each $(x, y) \in (P \times T) \cup (T \times P)$, and $m_{in} = m'_{in}|_P$, $P \cap H = \emptyset$.*

Let $\Sigma$ be a bounded P/T system, and $\Sigma'$ be a restriction of $\Sigma$. Since all the constraints in $\Sigma$ are maintained in $\Sigma'$, and $\Sigma'$ has some additional places that may add constraints to the occurrences of some transitions, a sequence of transition firings in $\Sigma'$ from $m'_{in}$ can always be reproduced in $\Sigma$ from $m_{in}$.

**Example 51.** *Fig. 4.17 recalls, on the left, the net already presented in Fig. 4.1, and represents one of its restrictions on the right, where the additional places are coloured in blue. In the initial P/T system, four transitions are enabled in the initial marking, namely $t_1$, $t_2$, $u_1$, and $u_2$, whereas in the restriction, only $t_1$ and $t_2$ are enabled, since the new places, initially unmarked, are preconditions of $u_1$ and $u_2$.*

**Definition 26.** *Let $\rho' = (B'_{\rho'}, E'_{\rho'}, F'_{\rho'})$ be a run in the unfolding of $\Sigma'$, and $\gamma'$ be a B-cut of $\rho'$. A* projection *of $\rho'$ on* UNF$(\Sigma)$ *is a run $\rho = (B_\rho, E_\rho, F_\rho)$ such that $\rho$ is isomorphic to $\rho' \setminus J$, with $J = \{j \in B' : \lambda(j) \in H\}$. Analogously, a* projection *of $\gamma'$ on $\rho$ is a B-cut $\gamma \subseteq B_\rho$ such that $\gamma = \gamma' \setminus J$.*

By construction, for each $\rho'$ run of UNF$(\Sigma')$, and for each B-cut $\gamma'$, there is always at least a projection on UNF$(\Sigma)$.

As discussed in Sec. 4.1, while following a strategy, the user has complete control over controllable transitions, for example it can decide to fire controllable transitions only after the occurrence of some uncontrollable transitions, to completely block their occurrence, or to order the occurrence of concurrently enabled controllable transitions in a specific way. However, it cannot force uncontrollable transitions to fire in any order and it cannot be sure to fire a controllable transition before an enabled uncontrollable transition has occurred. An implementation of a strategy must reflect these properties, therefore, we cannot allow for places that limit the behaviour of the environment. A naive solution for this would be to allow to add only places that are preconditions of controllable transitions. However, Ex. 52 shows a system in which the new places are preconditions of uncontrollable transitions, without limiting the behaviour of the environment.

**Definition 27.** *Let $\Sigma = (P, T, F, W, m_{in})$ be a P/T system, and $\Sigma' = (P \cup H, T, F', W', m'_{in})$ a restriction of $\Sigma$. A place $p \in H \setminus P$ is* environment-fair *if $p^\bullet \cap T_{env} = \emptyset$, or if the following conditions hold.*

1. *There is a transition $t \in T$ such that $p \in t^\bullet$.*
2. *For each $t_n \in T_{env}$ such that $t_n \in p^\bullet$, for each run $\rho \in \textsc{unf}(\Sigma')$ including the elements $b, e_1, e_2$ such that $\lambda(b) = p$, $\lambda(e_1) = t$, $\lambda(e_2) = t_n$, $e_1 \in {}^\bullet b, b \in {}^\bullet e_2$, then $e_1 \prec e_2$ in the projection of $\rho$ on $\textsc{unf}(\Sigma)$.*
3. *For each transition $t_n \in T_{env}$ such that $t_n \in p^\bullet$, for each transition $t_i \in p^\bullet$, and for each marking $m' \in [m'_{in}\rangle$, if $t_i$ and $t_n$ are in behavioural conflict in $m'$, then they are also in behavioural conflict also in $m = m' \cap P$.*

In Def. 27, the second condition expresses that if an uncontrollable transition is forced by a new place to occur after another one, then this same order constraint must hold also in the initial net. The third condition expresses that all the conflicts derived from a new place involving at least an uncontrollable transition must be structural and not behavioural.

We have now all the elements to define implementable strategies. In Def. 28 and Def. 29, we propose two notions: loosely speaking, $\Sigma'$ is a weak implementation of $\alpha$ on $\Sigma$ if all its maximal runs follow the behaviour prescribed by $\alpha$ on $\Sigma$, possibly restricting it; it is a strong implementation if all the behaviours allowed by $\alpha$ on $\Sigma$ are reproducible on $\Sigma'$.

Both notions have advantages and drawbacks. In order to guarantee that all the runs of a system satisfy a given property that holds when following the strategy, it is sufficient to use a notion of weak implementation. However, this notion may allow for unnecessary restrictions of the behaviour of the system, therefore in some cases the strong implementation may be preferable.

**Definition 28.** *Let $\Sigma = (P, T, F, W, m_{in})$ be a bounded P/T system. A strategy $\alpha$ is* weakly implementable *iff there is a bounded restriction $\Sigma' = (P \cup H, T, F', W', m'_{in})$ of $\Sigma$ such that the following conditions hold.*

1. *Each place in $H \setminus P$ is environment fair.*
2. *For each maximal run $\rho'$ in $\textsc{unf}(\Sigma')$, there exists a sequence $\delta'$ of B-cuts in $\rho'$, such that the projection $\rho$ of $\rho'$ on $\textsc{unf}(\Sigma)$, and the projection of the B-cuts in $\delta'$ on $\rho$ form a play $(\rho, \delta)$ on $\textsc{unf}(\Sigma)$ consistent*

*with $\alpha$, with $\delta$ the sequence of B-cuts that are projections of the B-cuts in $\delta'$.*

*If such a $\Sigma'$ exists, we say that it is a* weak implementation *of $\alpha$ on $\Sigma$.*

**Definition 29.** *Let $\Sigma = (P, T, W, m_{in})$ be a bounded P/T system. A strategy $\alpha$ is* strongly implementable *iff there is a bounded restriction $\Sigma' = (P \cup H, T, F', W', m'_{in})$ of $\Sigma$ such that $\Sigma'$ is a weak implementation of $\alpha$, and for each play $(\rho, \delta)$ consistent with $\alpha$, $\rho$ is a projection of a maximal run in $\text{UNF}(\Sigma')$.*

*If such a $\Sigma'$ exists, we say it is a* strong implementation *of $\alpha$ on $\Sigma$.*

**Remark 14.** *By definition, a weak implementation $\Sigma'$ of a strategy $\alpha$ on a P/T system $\Sigma$ has at least a maximal run in $\text{UNF}(\Sigma')$, since its initial marking cannot be empty. Hence, in the unfolding of a weak implementation there must always be at least a maximal run projecting on a play consistent with $\alpha$ in $\text{UNF}(\Sigma)$.*

To illustrate the definitions and the different cases that can occur, we discuss three examples of nets and strategies.

**Example 52.** *Consider the safe (1-bounded) P/T system on Fig. 4.18 where, in order to simplify the drawing, the output arcs of the five transitions labelled by R have been omitted. These transitions reproduce the initial marking, hence they all have as post-set the places $\{p_1, p_2, p_{10}\}$. In this system, the user has the goal to infinitely often reach a marking containing place $p_{12}$, it has no memory, and it can observe the value of the places $\{p_1, p_2, p_3, p_4, p_5, p_6, p_{10}, p_{12}\}$, which are represented with bold lines.*

*In the initial marking, there are two independent conflicts controlled by the environment (between A and B, and between C and D), and a conflict controlled by the user, between H and I. In order to fulfil its goal, the user has to wait for the environment to make its choices.*

*The system is safe, so we can take the powerset of the set of observed places as* Obs. *A winning strategy is: $\alpha(\{p_1, p_2, p_{10}\}) = \emptyset$, $\alpha(\{p_4, p_6, p_{10}\}) = \{H\}$, $\alpha(\{p_3, p_5, p_{10}\}) = \alpha(\{p_3, p_6, p_{10}\}) = \alpha(\{p_4, p_5, p_{10}\}) = \{I\}$. A strong implementation of this strategy is given by the P/T system on Fig. 4.19, where the additional places and their relations with the transitions of the P/T system are coloured in blue. Since the choice of H depends on the occurrence of both B and D, this is represented by adding place $p_{14}$ from B to H, and place $p_{15}$ from D to H. Instead, the choice of I depends on the occurrence of either A or C, hence it may be represented adding place $p_{13}$ from both A and C to I. In this way, if it is the case that both A and C occur, the added place $p_{13}$ may get two tokens and then the net system with this added place becomes 2-bounded and no more safe (1-bounded). In order to correctly reproduce the initial marking, the added places may be emptied by the uncontrollable transitions R, and this without limiting the behaviour*

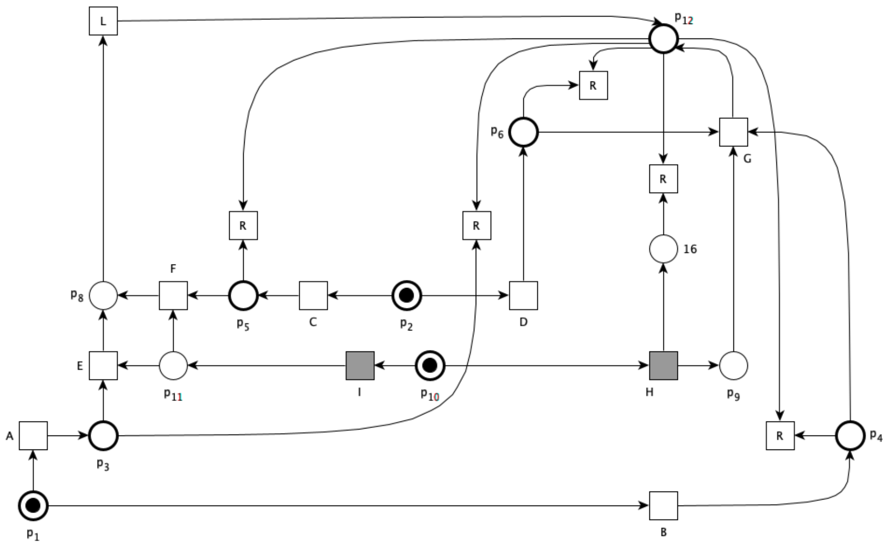


Figure 4.18: A P/T system where the user observes the thick places and has the goal to reach place $p_{12}$ infinitely often.

*of the environment, indeed it is possible to show that the added places, $p_{13}$, $p_{14}$ and $p_{15}$ are environment-fair, since they satisfy conditions* 1.*, and* 2. *of Def. 27.*

**Example 53.** *A strategy can be weakly implementable, but not strongly implementable. Consider the net system $\Sigma$ on the left of Fig. 4.20, and the following strategy, defined on markings, under the hypothesis that the user can see all places, but cannot distinguish distinct occurrences of them (braces are omitted in the arguments of the map): $\alpha(p_0, p_3) = \{t_0, t_1\}$, $\alpha(p_1, p_3) = \{t_2\}$, $\alpha(p_7, p_3) = \{t_2\}$, $\alpha(p_2, p_3) = \{t_3\}$, and $\alpha(m) = \emptyset$ for all other markings. The marking graph shown on the right of Fig. 4.20 is obtained from* MG($\Sigma$) *by removing arcs corresponding to controllable transitions which are not chosen by the strategy, and states which are no more reachable from the initial state. This transition system is not separated for bounded P/T systems without self-loops. As we will see in the next section, this means that the strategy is not strongly implementable. However, we can notice that the unsolvable separation problem is given by the absence of label $t_2$ in state $\{p_2, p_3\}$, and by the absence of label $t_3$ in the state $\{p_1, p_3\}$, which are reached if the user fires $t_0$ in the initial state. The strategy $\alpha$ gives the user the freedom to choose either $t_0$ or $t_1$ in that state, therefore a weak implementation of $\alpha$ can be realized by blocking $t_0$, so that the critical states are not reached, adding a place from $t_6$ to $t_2$, and blocking $t_3$.*

*An alternative weak implementation consists in adding a place from $t_6$ to $t_2$, and blocking the occurrence of $t_3$. In this way, if $t_0$ fires in the imple-*

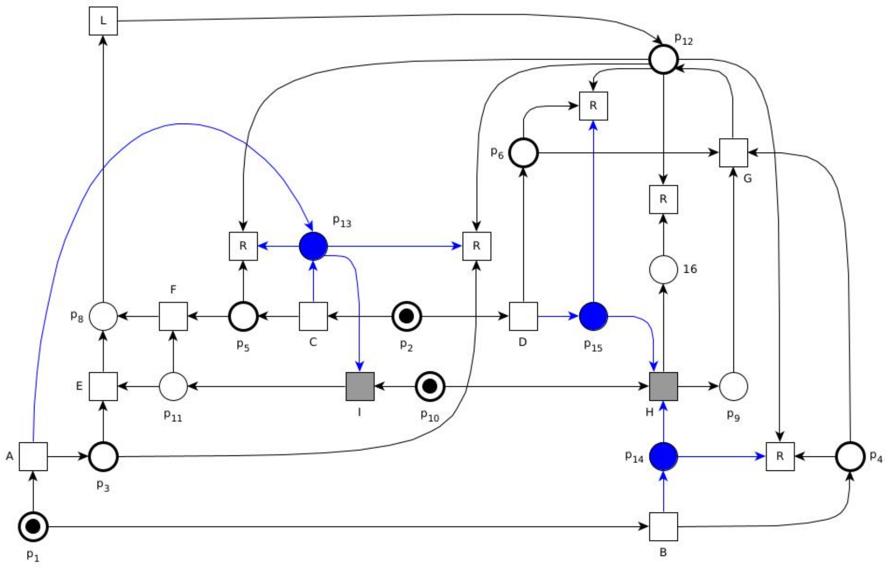


Figure 4.19: An implementation of a strategy for the P/T system in Fig. 4.18, adding preconditions also to uncontrollable transitions.

*mentation, neither $t_2$ nor $t_3$ can fire. This execution on the implementation projects on a consistent play, since neither $t_2$ nor $t_3$ are constantly selected by $\alpha$ after the choice of $t_0$. In the run selecting $t_1$ we are in the same case discussed above, therefore the requirements for the weak implementation are satisfied.*

**Example 54.** *In some cases, neither a weak implementation nor a strong implementation exist. Consider the net $\Sigma$ in Fig. 4.21, and assume that the user can distinguish each B-cut in* UNF$(\Sigma)$. *Consider the strategy $\alpha$ such that $\alpha(\gamma_0) = \alpha(\gamma_1) = \emptyset$, $\alpha(\gamma_2) = \alpha(\gamma) = \{t_0\}$, for each $\gamma > \gamma_2$ enabling $t_0$.*

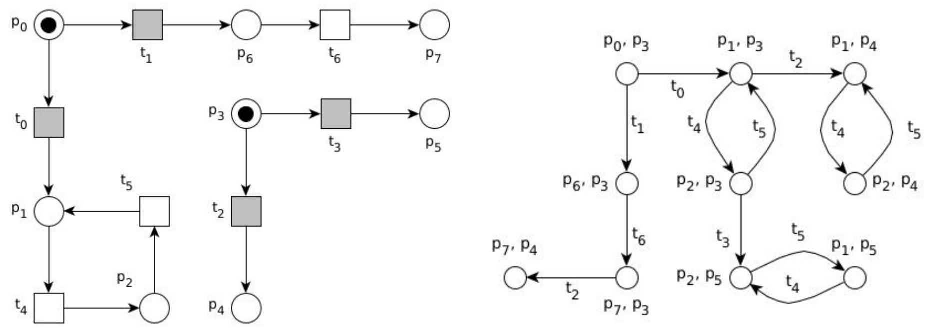


Figure 4.20: A weakly implementable strategy.

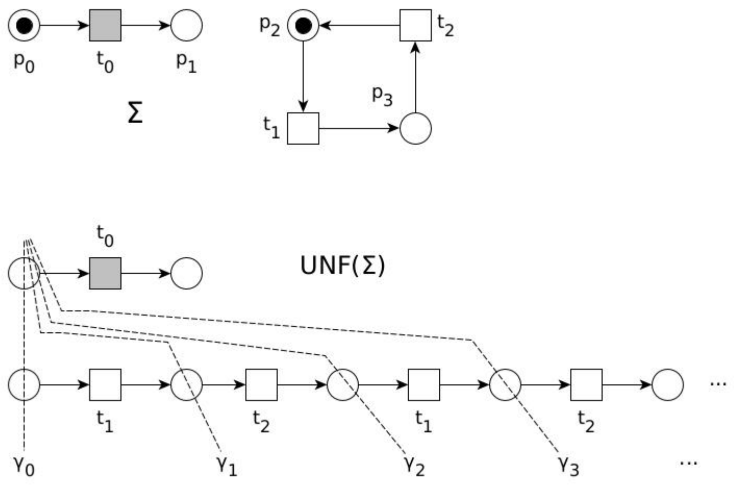


Figure 4.21: A 1-bounded net and its unfolding

*An implementation cannot block $t_0$, since in all the plays consistent with $\alpha$, $t_0$ must fire, since it is constantly enabled and selected from $\gamma_2$ on. Hence, $t_0$ needs to be blocked until the first occurrence of $t_2$, and we need to add a place from $t_2$ to $t_0$. Such a place is not bounded, therefore the resulting net is not a weak implementation of $\Sigma$.*

### 4.4.2 An algorithm to decide if a strategy is (strongly) implementable

We are interested in characterizing cases in which a given strategy is implementable. In this section we focus on P/T systems where the user cannot discriminate between two B-cuts associated to the same marking, namely, for each pair of B-cuts $\gamma_1, \gamma_2$ such that $\lambda(\gamma_1) = \lambda(\gamma_2)$, $\gamma_1 \equiv \gamma_2$. Any class of observations satisfying this condition is allowed. Hence, from now on, we consider strategies consistent with this assumption.

For this case, we propose an algorithm which, given a strategy, defined as a map from equivalence classes of markings to sets of controllable transitions, decides if the strategy is strongly implementable, and constructs an actual implementation, in the positive case. Its pseudo-code is presented in Algorithm 11.

The algorithm builds on region theory, and is conceptually simple: we first construct a reduced marking graph of $\Sigma$, where we remove all arcs corresponding to controllable transitions which are not compatible with the strategy, and remove states which are no more reachable from the initial marking. In practice, we can think of directly building the reduced marking graph, starting from the initial marking. Given a strategy $\alpha$, the reduced

**Algorithm 11** Implementable strategy

```
function IMPL($\Sigma$, $\alpha$)
                                    ▷ $\Sigma$ is a bounded P/T system, $\alpha$ a strategy.
    $\Sigma' \leftarrow \Sigma$
    build $\text{MG}_\alpha(\Sigma) = (M', T, A)$
    compute $\mathsf{SP} = \{(m,t) \mid m \in M', m[t\rangle \text{ in } \Sigma, t \notin \alpha(m)\}$
    for $(m,t)$ in $\mathsf{SP}$ do
        if there is a region $r$ solving $(m,t)$ then
            add to $\Sigma'$ a place corresponding to $r$
        else
            return failure                  ▷ $\alpha$ is not strongly implementable
        end if
    end for
    add a new place $\omega$ to $\Sigma'$, with $m'_{in}(\omega) = 0$,
    and make it an input place of each $t$ unreachable in $\text{MG}_\alpha$.
    return $\Sigma'$                        ▷ $\Sigma'$ is a strong implementation of $\alpha$.
end function
```

marking graph will be denoted $\text{MG}_\alpha(\Sigma)$.

We then try to synthesize the reduced marking graph by the calculus of regions. To this end, we first observe that, for each region $r$ of the full marking graph $\text{MG}(\Sigma)$, the restriction of $r$ to the set of states of the reduced marking graph $\text{MG}_\alpha(\Sigma)$ is a region of the latter. This implies that all the state separation problems in $\text{MG}_\alpha(\Sigma)$ are solved by restrictions of the "old" regions. Such restrictions also solve the state-transition separation problems which were already in $\text{MG}(\Sigma)$.

On the other hand, removing some transitions from $\text{MG}(\Sigma)$ creates new state-transition separation problems. The algorithm we propose looks for a set of regions solving these new problems. If no such set of regions exists, then the strategy is not strongly implementable, as will be proved below. If $U = \{r_1, \ldots, r_K\}$ is a set of separating regions for the new separation problems, then we can add to $\Sigma$ the set $H$ of corresponding places, and the arcs with the corresponding weights.

**Lemma 22.** *Let $\Sigma = (P, T, F, W, m_{in})$ be a bounded P/T system with $T = T_u \cup T_{env}$, $T_u \cap T_{env} = \emptyset$. Let $\text{MG}_R$ be a transition system obtained from $\text{MG}(\Sigma)$ by removing a subset of transitions labelled with elements in $T_u$, and the states unreachable from the initial marking. Let $U$ be a set of regions solving the new state-event separation problems, and $H$ the set of places derived from them. The restriction $\Sigma' = (P \cup H, T, F, W, m'_{in})$ of $\Sigma$ is a bounded P/T system, and all the places in $H$ are environment-fair.*

*Proof.* Boundedness follows from the fact that $\Sigma'$ is separated, and its marking graph is finite. To show that the new places are environment-fair, we proceed in two steps. First, we note that by removing arcs labelled by controllable transitions, we cannot introduce new behavioural conflicts between uncontrollable transitions. Second, suppose that in a run of $\Sigma'$, there is a B-cut $\gamma$ enabling an occurrence $e_1$ of an uncontrollable transition $t_1$; suppose moreover that in the same run there is an occurrence $e_2$ of another uncontrollable transition, $t_2$, and that $e_1$ precedes $e_2$ in this run, while $e_1$ and $e_2$ are not ordered in the projection of the run. Then, in $\Sigma$, the reachable marking $\lambda(\gamma)$ must enable $t_1$ and $t_2$, and these two transitions must be concurrent in $m$. Hence, there must be reachable markings $m_1$, $m_2$, and $m_3$, with $m[t_1\rangle m_1$, $m[t_2\rangle m_2$, $m_1[t_2\rangle m_3$, and $m_2[t_1\rangle m_3$. None of these arcs in $\textsc{mg}(\Sigma)$ can be removed by the reduction operation, hence $e_1$ and $e_2$ should be concurrent in $\gamma$. $\square$

**Lemma 23.** *Let $\Sigma = (P, T, F, W, m_{in})$ be a bounded P/T system whose transitions are partitioned into controllable ($T_u$) and uncontrollable ($T_{env}$). A strategy $\alpha : \mathsf{Obs} \to 2^{T_u}$ is strongly implementable if, and only if, the reduced marking graph $\textsc{mg}_\alpha(\Sigma)$ is separated. Furthermore, if $\textsc{mg}_\alpha(\Sigma)$ is separable, $\Sigma'$ returned by Algorithm 11 is a strong implementation of $\alpha$ on $\Sigma$.*

*Proof.* First, we prove that if a strategy $\alpha$ is strongly implementable on $\Sigma$, then $\textsc{mg}_\alpha(\Sigma)$ is separable. Let $\Sigma'$ be a strong implementation of $\alpha$ on $\Sigma$. By contradiction, assume that $\textsc{mg}_\alpha(\Sigma)$ is not separable. Then, $\textsc{mg}(\Sigma')$ cannot be isomorphic to $\textsc{mg}_\alpha(\Sigma)$ and either there is a path $\sigma'$ in $\textsc{mg}(\Sigma')$ that cannot be reproduced in $\textsc{mg}_\alpha(\Sigma)$, or there is a path $\sigma$ in $\textsc{mg}_\alpha(\Sigma)$ that cannot be reproduced in $\textsc{mg}(\Sigma')$. In the first case, there must be a run in $\textsc{unf}(\Sigma')$ that cannot be associated to any play in $\textsc{unf}(\Sigma)$; in the latter case, there must be a play on $\textsc{unf}(\Sigma)$ that is not associated to any run in $\textsc{unf}(\Sigma')$. Hence, in both cases $\Sigma'$ is not a strong implementation of $\alpha$ on $\Sigma$, contradicting the hypothesis.

Next, we show that if the marking graph is separated, $\alpha$ is strongly implementable, and $\Sigma'$ returned by Algorithm 11 is a strong implementation. Lemma 22 shows that the places added to $\Sigma'$ are environment fair. We need to show that also the other conditions in Def. 28 and Def. 29 are satisfied. The second point of Def. 28 and the condition in Def. 29 follow from the observation that, by construction, $\textsc{mg}(\Sigma')$ and $\textsc{mg}_\alpha(\Sigma)$ are isomorphic. As sequence of B-cuts, we can consider those associated to sequences of markings in $\textsc{mg}(\Sigma')$. $\square$

The computational cost of solving one state-event separation problem is polynomial in the size of the transition system, so the overall cost of our algorithm depends mainly on the construction of the reduced marking graph. If $\Sigma$ is $k$-bounded and has $h$ places, then the marking graph has at most $(k+1)^h$ reachable markings. Hence the complexity in the worst case is exponential in the number of places. For a given strategy, the number

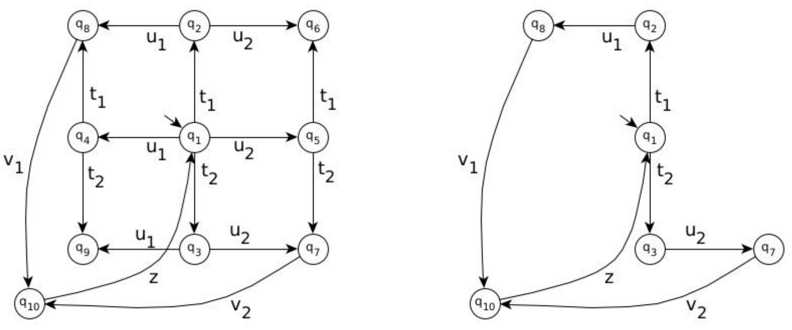


Figure 4.22: A marking graph and its reduction.

of reachable markings in the reduced graph can be much smaller, but this depends strongly on the specific strategy.

**Example 55.** *Consider the P/T system $\Sigma$ in Fig. 4.1 (recalled also in Fig. 4.17), and the associated discussion. A winning strategy is given by the following map: $\alpha(\{p_0, s_1\}) = \{u_1\}$, $\alpha(\{p_0, s_2\}) = \{u_2\}$, and $\alpha(\{m\}) = \emptyset$ for all other markings.*

*The marking graph of $\Sigma$ is shown on the left of Fig. 4.22. The reduced marking graph, obtained by removing controllable transitions not chosen by the strategy, is on the right. This reduced transition system turns out to be separated. Beyond the regions corresponding to places of $\Sigma$, we need to find regions solving four new state-transition separation problems: the initial state labelled with $q_1$ must be separated both from $u_1$ and from $u_2$; $q_3$ must be separated from $u_1$, and $q_2$ from $u_2$. The set $\{q_2\}$ is a region in the reduced marking graph, and solves all the separation problems related to $u_1$. Analogously, the region $\{q_3\}$ solves the separation problems related to $u_2$. An implementation of the strategy where the places associated to these regions has been added is represented on the right in Fig. 4.17. Where the place on the top of the figure corresponds to $\{q_2\}$, and the one of the bottom to $\{q_3\}$.*

## 4.5 Related works

Games have been used to model the interaction between agents on a system for decades. One of the most used game model considers two-player infinite games on finite graphs [67, 98]. In this context, a play is an infinite path on the graph, and the two players are opponents. The nodes of the graph are divided among the two players, that move whenever the play reaches one of their nodes. A winning condition for a player may require for example to reach or to avoid a set of nodes.

Unlike in the context of games on graph, in this thesis we considered an asynchronous game, without turn division between the two players. Our game develops the idea of game on the unfolding in introduced in [22]. In that paper, the authors consider the problem of weak observable liveness (WOL) [45, 20], consisting in forcing a target transition of a 1-safe system to fire infinitely often, by controlling some transitions in the system. In [22], the problem was modelled as a two-player game on the unfolding of a 1-safe system, on which the user could observe all the transitions. A previous attempt to model WOL made use of Streett games [67], which are infinite games on finite graphs where the winning condition is defined as the Streett acceptance condition [119]. Although this method provided a technique to model-check WOL with full observability under some restrictions, the interleaving semantics used in games on graphs prevented to easily represent asynchronicity of the system.

In [58, 59], Finkbeiner and Olderog introduced a different notion of asynchronous games on Petri nets, called Petri games. As for the game in this thesis, Petri games are defined on the unfolding of the net; the players are represented as tokens on the net, and they are divided into two teams, namely the *system* and the *environment*. One of the most interesting aspects of this game is that the players have causal memory, i.e. they are fully informed about their past, and they have no information about the rest of the system until a synchronization occur. When two or more players synchronize through the occurrence of a shared transition, they get the information about the past of the tokens participating in that transition. In [59], the authors studied how to check whether the system players have a strategy to avoid a set of bad places. Among the works extending the results in [58, 59], [64, 56] propose tools for the analysis of Petri games and the synthesis of strategies; [55, 57, 76, 75] study the complexity and decidability of Petri games by considering different classes of Petri nets, number of players in the two teams, and winning conditions; [26] proves an equivalence between Petri games and control games [62] defined on Zielonka's asynchronous automata [134].

The idea of games on concurrent structures such as Petri nets, VASS, and event structures was defined also in several other works. In [84], the authors consider a two-player game on Petri nets in which players strictly alternate moves, and one of them has the goal of reach a subset of states. In [69], the authors study decidability conditions on a concurrent game on event structures. Finally, [18] presents an asynchronous two-player game on a VASS based model, studies the decidability and complexity class of some safety and reachability goal, and propose algorithms to find strategies, when the problem is decidable.

Another line relevant in the study of games and control problems is given by game-based logics. In Sec. 4.3.1, we have already discussed the relation of our game with ATL [9].

Finally, the research line on supervisory control of discrete event systems (DES) started by Ramadge and Wonham in the late 80s [108, 109] is also very relevant for our work, and in particular for the results presented in Sec. 4.4. In Chap. 3 and Chap. 4 of their book [36], Cassandras and Lafortune collect some results about the control of a DES with a supervisor. Supervising a DES means to limit its behaviours, so to avoid that it can generate unwanted strings of its language. A supervisor is a function that, for each given observation, chooses some controllable events to disable. In particular, Chap. 4 shows how these results can be adapted when DES are modelled as labelled Petri nets.

The idea of using region theory to construct a supervisor in a Petri net was introduced in [110], where the goal of the supervisor is to avoid a set of bad states. In a later work [63], the authors extended their work by requiring that the supervised system need to be able to always restore the initial marking, in addition to the avoidance of bad states. In [91, 111, 103], the authors improve the efficiency of the search for regions and consider applications of this approach in the control of flexible manufacturing systems.

# Chapter 5

# Toward multi-agent system analysis with Petri nets

In Chap. 4, we considered systems with a centralized controller, and we studied how it could control and influence the rest of the system. However, in a distributed system, there may also be the interaction of several agents, each of them with its own goal, not necessarily in conflict with the goals of the others. Most of the results in the formal analysis of strategic abilities in such systems consider agents as transition systems. The main challenge that these models need to tackle is the high computational complexity, that partly depends on the huge size of the global model.

Using Petri nets to model agents may help to produce smaller models, since they may be much smaller than the transition system describing the same behaviour. In this chapter, we present a translation from a multi-agent system defined with transition systems to a multi-agent system modelled with 1-safe systems, we show their behavioural equivalence and we consider the problem of finding transitions in the global model which are not 1-live. In future works we plan to use this Petri net framework for model-checking game-based logics.

The results in this chapter generalize those published in [8]. In particular, in [8] we considered a naive construction of the agents as 1-safe systems, in which each labelled arc in the transition system was transformed into a transition of the 1-safe system. Here, we show that the results in [8] also hold when we construct 1-safe systems through region theory.

The chapter is structured as follows. Sec. 5.1 discusses related works, with a special focus on the multi-agent model defined in [79], that we want to translate into a Petri net model. Sec. 5.2 defines the translation and shows the equivalence between the two models. Finally, Sec. 5.3 discusses an algorithm to find transitions that are not 1-live in the global model.

## 5.1 Related works

Synthesis and analysis of multi-agent systems is a well known topic in the literature, also in the context of Petri nets. Pujari and Mukhopadhyay in [107] discuss MAS as discrete-event dynamic systems (DEDS), and use Petri nets as a modeling tool to assess the structural properties of the system. Similarily, following DEDS concept, Lukomski and Wilkosz [94] show rules of modeling and analyzing the considered multi-agent system with use of Petri nets. In [53], the authors propose to use the compositional specification power of Petri nets in application to a multi-agent system; highly distributed air traffic operations system is considered and modelled using Stochastically and Dynamically Coloured Petri Nets. The approach of Galan and Baker [60] focuses on specifying and analyzing the conversations in a multi-agent system. Conversations are specified using an automata model and converted into a Petri net representation. Using a Petri net analyzer, the conversations are checked for consistency and coherency by testing liveness and safety of the resulting net. Hiraishi in [77] proposes $PN^2$ model, an extension of P/T nets, for the design and the analysis of multi-agent systems.

Another branch of the literature on the subject shows a number of approaches using Petri nets to coordinate, organize or plan MAS behaviors. In [135] a representation and execution framework for high level multirobot plan design, called Petri Net Plans (PNP), is proposed. PNP is based on Petri nets with a domain specific interpretation. Places and transitions are partitioned into several classes of different interpretations. A special case is a Petri net that has at most one token per place and edges of weight one. As a central feature, PNPs allow for a formal analysis of plans based on standard Petri net tools. Scheduling by hierarchical structuring of the tasks performed by agents is one of key ideas in [99]. They propose a multi-agent system that allows the user to define a hierarchical structuring of the tasks that these agents perform, to plan a schedule involving parallel and sequential calling of the agents. The agents are atomic or complex. Atomic agents are simple Petri nets performing a task, while complex agents are used to gather atomic (and/or other complex agents) to conglomerate their individual behaviour, and arrange their working order. The authors of [88] propose a framework for specifying multi-agent systems based on Synchronized Petri Nets. It is an extension of Recursive Petri nets, facilitating multi-agent system specifications by concepts like: typed places, transitions and tokens, synchronization points, synchronization conditions, synchronization relations and binding functions.

Finally, an approach that seems to be very close to ours uses Nested Unit Petri Nets (NUPN) [61]. One can see multi-agent systems as safe NUPNs of height 1 (taking every agent as a leaf unit and the whole system as the root unit). The agents that we construct in this work are not necessarily unit-safe; however, they would become unit-safe if we decompose them into sequential components as we plan to do in future works. The considered problem of transitions disabled by synchronizations corresponds to the usual weak-liveness check for such nets.

### 5.1.1 Asynchronous multi-agent systems (AMAS)

In this section we focus on the model presented in [79]. This model is particularly relevant for the results discussed in this chapter, since it is the starting point of the definition of multi-agent system based on Petri nets presented in the next section.

In [79], the authors define an asynchronous multi-agent system as a network of automata, and they use it as basis for model-checking a fragment of ATL* [9]. They define a notion of partial information on the system for the agents, and apply a partial order reduction for model-checking strategic abilities of a group of agents. Interestingly, their technique cannot be applied when the agents have perfect information on the global system.

In what follows, we recall some formal definitions of the model in [79] that will be used to present the result of this chapter.

**Definition 30.** *An* asynchronous multi-agent system (AMAS) *consists of $n$ agents $\mathcal{A} = \{\text{AG}_1, \dots, \text{AG}_n\}$. Each agent is associated with a tuple $\text{AG}_i = (Q_i, U_i, A_i, q_{i,0}, \mathcal{PV}_i, V_i)$, where*

- $Q_i = \{q_i^1, \dots, q_i^{n_i}\}$ *is a set of local states;*
- $U_i = \{\alpha_i^1, \dots, \alpha_i^{m_i}\}$ *is a set of labels representing the events in which agent $\text{AG}_i$ can choose to participate;*
- $A_i \subseteq Q_i \times U_i \times Q_i$ *is a set of arcs or transitions local with respect to agent $\text{AG}_i$;*
- $q_{i,0} \in Q_i$ *is an initial state;*
- $\mathcal{PV}_i$ *is a set of local propositions;*
- $V_i : Q_i \to 2^{\mathcal{PV}_i}$ *is the valuation of local propositions in local states.*

The sets of labels of different agents may not be disjoint. This is because different agents may participate in the same events; events which are present in more than one label set $U_i$ require participation of all the agents having them in their $U_i$, needing to synchronize for the execution.

**Example 56.** *Fig. 5.1 represents an* AMAS *with three agents. The events $n_1, m_1, n_2, m_2$ are shared by two agents, and require the participation of both of them to occur, whereas the events $n_3$ and $m_3$ are local, and depend on a single agent.*

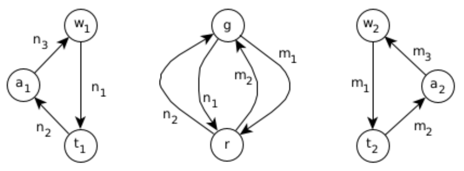


Figure 5.1: Example of three local models for agents related to Train-Gate-Controller benchmark from [79].

The composition of agents in an AMAS happens with synchronizations on common events. The resulting structure is the *canonical interleaved interpreted system*, which is formally defined below.

Since the model checking of AMAS is not in the scope of this thesis, we simplify the definition of agents and of their composition by only considering the structure of their transition systems, namely the tuples $(Q_i, U_i, A_i, q_{i,0})$ (omitting the propositional variables).

**Definition 31.** *A* canonical interleaved interpreted system (canonical IIS*) is an* AMAS *extended with a tuple* $(Q, U, A, q_0)$ *where:*

- $Q \subseteq Q_1 \times \ldots \times Q_n$ *is a set of global states;*
- $U = \bigcup_{i \in \{1,\ldots,n\}} U_i$ *is a set of events;*
- $A \subseteq Q \times U \times Q$ *is a set of arcs or transitions in the global system such that* $a = ((q_1, ...q_n), \alpha(q'_1, ..., q'_n)) \in A$ *iff for each agent* $\text{AG}_i$ *with* $\alpha \in U_i$*,* $(q_i, \alpha, q_i) \in A_i$*.*
- $q_0 = (q_{1,0}, \ldots, q_{n,0})$ *is an initial state.*

Given a canonical IIS $I$, some of its states may not be reachable through any execution, due to the restrictions given by the synchronizations, and therefore also the transitions outgoing from these states can never be executed. We will denote with $I_r$ the canonical system where these unreachable states and transitions have been pruned.

By definition, the number of states in the IIS grows exponentially with the number of agents, therefore limiting the number of compositions when studying the properties of the system may help in the analysis.

**Example 57.** *Fig. 5.2 represents the reachable part of the canonical IIS of the* AMAS *in Fig. 5.1.*

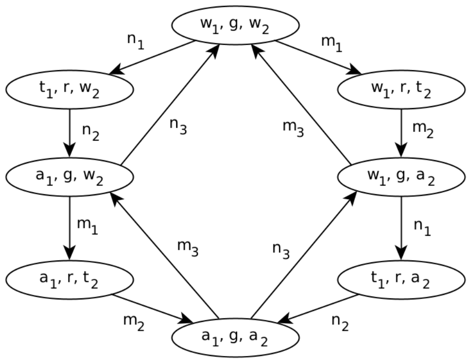


Figure 5.2: IIS of the AMAS in Fig. 5.1

## 5.2 Modelling asynchronous multi-agent systems with 1-safe systems

In Sec. 5.1.1 we recalled the formal definition of AMAS and IIS introduced in [79]. In this section we show how to translate such a model into a multi-agent system modelled by 1-safe systems.

Since each agent in an AMAS is an LTS, we can use region theory to synthesize a P/T system for each of them. In particular, in this section we consider the synthesis of 1-safe systems. As we discussed in Sec. 2.5, not always the synthesis of an LTS through region theory is possible. However, we can always obtain a labelled 1-safe system (see Sec. 2.1). The representation of agents with 1-safe systems is at most large as the representation with transition systems, but, if the same label appears multiple times in the agent, the 1-safe system may be smaller. This is particularly important at the composition stage: even in the worst case, the composed 1-safe system can be exponentially smaller than the composition of transition systems; in the best case in which each agent can be synthesized with region theory, the composed 1-safe system has only one transition for each label of the AMAS.

**Remark 15.** *In general, we can have different labelled 1-safe systems realising an LTS, based on which criteria we choose for splitting labels.*

Formally, we can represent each agent $\text{AG}_i$ as a labelled 1-safe system. In particular, for each agent $\text{AG}_i = (Q_i, U_i, A_i, q_{i,0})$, the associated labelled 1-safe system is defined as $\Sigma_i = (D_i, T_i, F_i, r_{i,0})$ together with a map $\beta : T_i \to U_i$, where:

- $D_i$ is a set of regions solving the separation problems in $\text{AG}_i$, possibly by splitting some of the labels in $U_i$;
- $T_i$ is the set of transitions; there must be at least one for each element in $U_i$, and there could be more if splitting transitions is necessary for

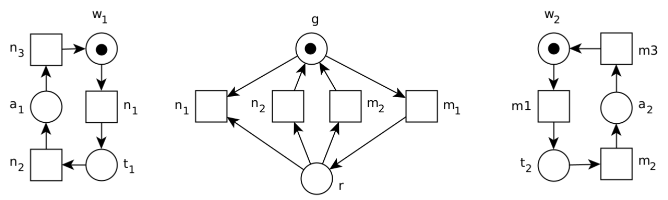


Figure 5.3: 1-safe systems for the transition systems depicted in Fig. 5.1.

the synthesis;

- $F_i \subseteq (D_i \times T_i) \cup (T_i \times D_i)$ is constructed as described in Sec. 2.5;
- $r_{i,0} \subseteq D_i$ is the set of region including $q_{i,0}$;
- $\beta$ is the function associating to each transition in $T_i$ its label $U_i$ on AG$_i$; if AG$_i$ can be synthesized without splitting of transitions, $\beta$ is injective.

**Example 58.** *Consider the* AMAS *in Figure 5.1. In Figure 5.3, each agent has been synthesized into a* 1*-safe system. In this case, each agent is trivially synthesizable in an exact way: since each action is never repeated inside the same agent, each of their states is a minimal region, and the set of minimal regions satisfies all the separation properties.*

### 5.2.1 Synchronization on common actions

Let $\Sigma_1 = (P_1, T_1, F_1, m_{1,in}), ..., \Sigma_n = (P_n, T_n, F_n, m_{n,in})$ be a set of labelled 1-safe systems, and $\beta_1, ..., \beta_n$ their labelling functions with $\beta_i$ labelling function of $\Sigma_i$ for each $i \in \{1, ..., n\}$.

We construct a *global* 1-safe system $\Sigma = (P, T, F, m_{in})$ with labelling function $\beta$ by synchronizing the agents on transitions with the same label. The set of places $P$ of $\Sigma$ is the union of the sets of places $P_i$, assumed to be pairwise disjoint. For each label $\alpha \in \beta(T_i)$, for each agent $\Sigma_i$, let $T_i^\alpha = \{t \in T_i : \beta(t) = \alpha\}$ be the set of transitions labelled with $\alpha$. The set of transitions of $\Sigma$ is defined as $T = \bigcup_{\alpha \in \beta(T_i)} \bigotimes_{i \in \{1,...,n\}} T_i^\alpha$. The flow relation is determined in this way: for each transition $t \in T$, and each place $p \in P$ there is an arc from $p$ to $t$ iff there is a $\Sigma_i$ and $t_j \in T_i$ such that $p \in P_i$, $t_j$ is a component in $t$, and $(p, t_j) \in F_i$; analogously for the arcs from $t \in T$ to $p \in P$. The initial marking $m_{in}$ is the union of all $m_{i,in}$, with $i \in \{1, ..., n\}$. The labelling function $\beta$ associates every transition $t \in T$ to the label of all its component, that is unique by construction.

By construction, each place in $\Sigma$ belongs at most to one agent, whereas the transitions can be shared. Note that some of the transitions may be enabled in no reachable marking, and therefore are not 1-live. The same

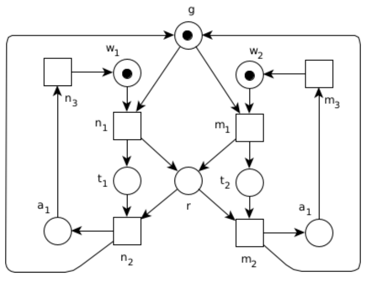


Figure 5.4: Global 1-safe system model resulting from the composition of the nets depicted in Fig. 5.3.

problem happens when we consider the composition of AMAS, because some states may not be reachable from the initial state $q$, due to the synchronization constraints. The problem of finding these transitions is discussed in detail in Sec. 5.3.

**Example 59.** *Fig. 5.4 represents the composition of the three agents from Fig. 5.3 based on synchronizations on transitions. Since in this case every agent is synthsizable, in the global* 1*-safe system in Fig. 5.4 each transition has a different label.*

In order to prove the equivalence between the behaviour of the global 1-safe system and of the IIS as defined in [79] and recalled in Def. 31, we show that synthesis and composition commute, namely synthesizing 1-safe system agents from the AMAS and then composing them is equivalent to construct the composition of AMAS and then synthesizing a 1-safe system. As already noted in Remark 15, the synthesis of labelled 1-safe systems may produce different sets of agents. However, for all the possible sets the commutativity holds.

**Theorem 10.** *Let $\mathcal{A} = (\text{AG}_1, ..., \text{AG}_n)$ be an* AMAS, *$\Sigma_1, ..., \Sigma_n$ be a set of* 1*-safe systems and $\beta_1, ..., \beta_n$ be a set of labelling functions such that for each $i \in \{1, ..., n\}$, $\Sigma_i$ labelled with $\beta_i$ synthesizes $\text{AG}_i$. Let $\Sigma = (P, T, F, m_{in})$ be a global* 1*-safe system labelled with $\beta$ constructed as described above, and $I$ the canonical IIS of $\mathcal{A}$. The transition system of $\Sigma$ is isomorphic to the reachable part $I_r$ of $I$.*

*Proof.* The proof is based on the fact that the set of places $P$ of the global 1-safe system can be partitioned into the places of the agents $\Sigma_i$, and each agent has a marking graph isomorphic to an agent $\text{AG}_i$.

We first show that the set of labels of the transitions enabled in $m_{in}$ in $\Sigma$ is the same as the set of labels of the transitions occurring in the initial state $q_0$ of $I_r$, and that for each label $\alpha$, the cardinality of the set of transitions associated to $\alpha$ is the same.

Assume that $q_0$, initial state of $\mathcal{A}$, enables a transition with label $\alpha$. Let $\text{AG}_\alpha = \{\text{AG}_j : j \in J \subseteq \{1, ..., n\}, \alpha \in \beta(U_j)\}$ be the set of agents in the AMAS with $\alpha$ in their alphabet. All the $\text{AG}_j \in \text{AG}_\alpha$ must enable a transition labelled with $\alpha$ in their initial state. By contradiction, assume that no transition labelled with $\alpha$ is enabled in $m_{in}$; since $m_{in}$ is the union of the initial states of the $\Sigma_i$, there must be a $\Sigma_i$ with $\alpha$ in its alphabet that does not enable any transition labelled with $\alpha$ in its initial state. This is impossible, since by construction, its marking graph is isomorphic to the marking graph of a $\text{AG}_i \in \text{AG}_\alpha$. Similarly, if there is a transition with a label $\alpha$ enabled in $m_{in}$, then each 1-safe system agent having it in its alphabet must have it enabled in the initial state, since by construction, the marking graph of the 1-safe system agents are isomorphic to the agents in $\mathcal{A}$, $\alpha$ must be enabled in the initial state also in $I_r$.

For each label $\alpha$, the number of transitions labelled $\alpha$ and enabled in $q_0$ is the product of the numbers of transitions enabled in the initial states of each agent in $\text{AG}_\alpha$; since the marking graphs of the 1-safe system agents are isomorphic to the agents in $\text{AG}_\alpha$, in $m_{in}$ the same number of transitions labelled with $\alpha$ is enabled.

Next, we show that two transitions bring to different markings in $\Sigma$, iff they bring to different states in $I_r$. Assume that two transitions $t_1$, $t_2$ enabled in $q_0$ arrive in different states of $I_r$. By construction, there is at least an agent $\text{AG}_i$ participating in the action and such that the local state after the occurrence of $t_1$ differs from the local state after the occurrence of $t_2$. This must happen also in $\Sigma_i$, since its marking graph is isomorphic to $\text{AG}_i$. Therefore, the markings reached from $m_{in}$ in $\Sigma$ differs at least for the places belonging to $\Sigma_i$. Analogously, if $t_1$ and $t_2$ lead from $q_0$ to the same state in $I_r$, then for all the agents participating in them, the local states after $t_1$ and $t_2$ must be the same, and this is true also for all the $\Sigma_i$ participating in $t_1$ and $t_2$, and therefore also for their union.

This same reasoning can be applied recursively to the states reached from the initial marking, until considering all the reachable states. With the same argument we can also show that a cycle is closed on the marking graph of $\Sigma$ iff it is closed in $I_r$. $\square$

## 5.3 1-liveness of transitions in the global model

In this section we discuss how to find transitions that are not 1-live on the global 1-safe system. This is known to be a PSPACE-complete problem [81, 50]. We propose an algorithm that, in some cases, does not need to construct the global 1-safe system in order to verify whether a transition is 1-live, but uses a smaller subnet. If this is possible, some computation is saved, since the complexity of the problem depends on the size of the net.

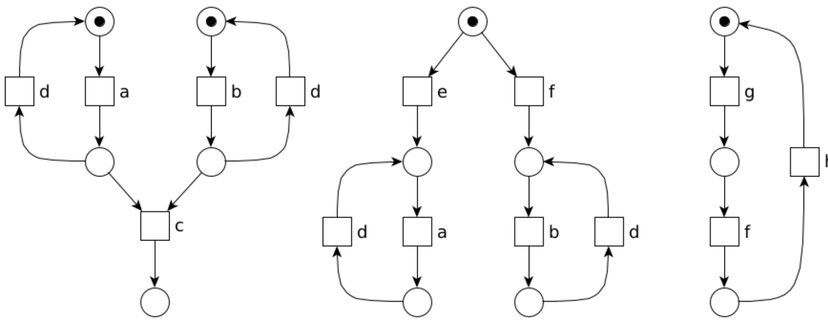


Figure 5.5: Multi-agent system with three agents.

In the worst case, the algorithm reconstructs the global 1-safe system, and checks 1-liveness on it.

**Example 60.** *Fig. 5.5 shows three agents modelled with* 1*-safe systems. In each of them, every transition is 1-live. However, in the global* 1*-safe system obtained by synchronizing the transitions with the same labels, c is not* 1*-live anymore.*

Although we can verify which transitions are 1-live in the global 1-safe system by computing its marking graph, this can be computationally very expensive, since having all the agents may increase the level of concurrency, and therefore the size of the marking graph.

A first alternative idea could be to find some of the transitions that will never be enabled by composing, for each label, all the agents sharing it. For example, the labels $a$ and $b$ in the multi-agent system in Fig. 5.5 are shared by the first and the second agent from the left in the figure. Fig. 5.6 shows the composition of the two agents, and the marking graph of this 1-safe system. If a transition cannot occur in the 1-safe system obtained composing only the agents sharing its label, then it cannot occur in the global 1-safe system, since adding new components can only increase the number of constraints, due to the synchronization requirements.

Unfortunately, this is only a necessary condition to identify transitions that cannot fire, but it is not sufficient. Transition $c$ is not 1-live in the global 1-safe system, but it appears only in the first agent, where it is 1-live.

This happens because $c$ must occur after the occurrences of both $a$ and $b$. This is not possible in the global system, since transitions $a$ and $b$ in the first agent must synchronize with transitions $a$ and $b$ in the second agent, but in the second agent $a$ and $b$ are in conflict and cannot occur in the same execution.

This suggests us another element to check whether a transition can be enabled without constructing the transition system of the global 1-safe system, consisting in composing all the agents sharing a certain label, and all

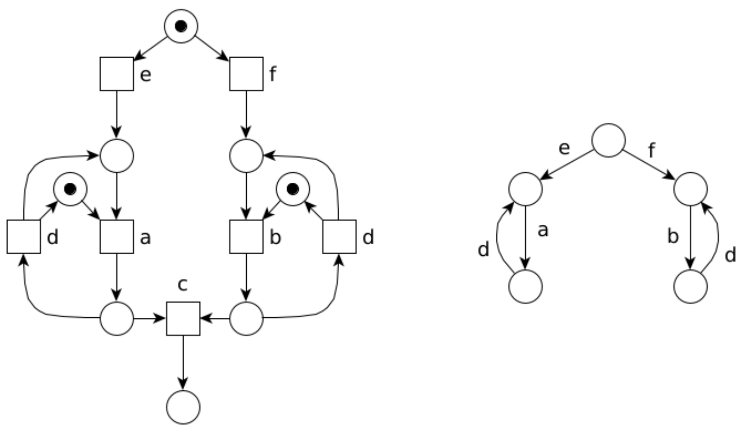


Figure 5.6: 1-safe system obtained by composing the first and the second agents in Fig. 5.5, and its marking graph.

the labels which appear in a minimal path leading to it from the initial state. In case of label $c$, this means to compose the agents with labels $a, b, c$, namely, the two on the left in Fig. 5.5. The marking graph of this composition is on the right in Fig. 5.6, and it does not include any transition labelled with $c$.

**Definition 32.** *Let* $\text{MG}(\Sigma)$ *be the marking graph of* $\Sigma = (P, T, F, m_{in})$. *The firing sequence* $t_1...t_n$ *is* minimal *if for each* $i, j < n$, $m_i \neq m_j$, *where* $m_0 = m_{in}$, *and* $m_{i-1}[t_i\rangle m_i$ *for each* $i \in \{1, ..., n\}$.

Let $L \subseteq \beta(T)$ be any subset of labels in the global net $\Sigma$. We will denote with $\Sigma^L$ the net obtained by composing all the agents with at least an element of $L$ in their alphabet.

Let $\alpha$ be any label on the system, and $T_\alpha = \{t_j \in T : \beta(t_j) = \alpha\}$. Algorithm 12 shows how to check that $t_j \in T_\alpha$ is 1-live in $\Sigma$, without computing the entire system. By applying the algorithm to each $t_j \in T_\alpha$, we can discover which transitions of $\Sigma$ are not 1-live, and therefore can be removed without changing the behaviour of the net.

The algorithm takes as input a transition $t_j$, the set of all the agents in the system, and a set of labels $L$, and returns true if there is a firing sequence of transitions that enables $t_j$, false otherwise. In addition, if one exists, the algorithm returns the sequence $\pi$ of transitions leading to $t_j$. In the first call $L = \{\alpha\}$.

The algorithm has a recursive structure. The first step consists in computing the minimal paths from the initial state of $\Sigma^L$ to $t_j$ (this is done by the function comp_min_paths). If there are no minimal paths in $\text{MG}(\Sigma^L)$, then it returns false, since $\Sigma^L$ does not need to be further explored. Otherwise, it selects a minimal path $\pi$, through the function select_path. Let

$\beta(\pi)$ be the set of labels of the transitions in $\pi$. If $\Sigma^{\beta(\pi)} = \Sigma^L$, then the algorithm returns true, since we found a path that can be executed on $\Sigma$ and enables $t_j$. When this happens, there is no need to look for alternative paths, and the computation can stop and return true. This is not the case if a recursive call returns false, since the unreachability of $t_j$ may be due to a wrong choice of the path in one of the previous steps. Then, we need to check if, in previous calls, other paths could have been chosen, leading to different subsystems, and check if $t_j$ is reachable in them. If $t_j$ is not reachable from any path, then we can conclude that $t_j$ is not 1-live in $\Sigma$.

**Algorithm 12** Check if $t_j$ is 1-live

```
function CHECK_1_LIVENESS($t_j, \{\Sigma_i : i \in \{1, ..., n\}\}, L) \in \{$true, false$\} \times \Pi)$
    $\Pi \leftarrow$ comp_min_paths(MG($\Sigma^L$), $t_j$)
    if $\Pi = \emptyset$ then
        return false, $\emptyset$
    end if
    while $\Pi \neq \emptyset$ do
        $\pi \leftarrow$ select_path($\Pi$)
        $\Pi \leftarrow \Pi \setminus \{\pi\}$
        if $\Sigma^L = \Sigma^{\beta(\pi)}$ then
            $\pi' \leftarrow \pi$
            return true, $\pi$
        end if
        $r, \pi' \leftarrow$ check_1liveness($t_j, \{\Sigma_i : i \in \{1, ..., n\}\}, \beta(\pi)$)
        if $r =$ true then
            return true, $\pi'$
        end if
    end while
    return false, $\emptyset$
end function
```

By construction, for each transition $t_i$ in the sequence $\pi$ returned by Algorithm 12, the set of preconditions and the set of postconditions are the same in $\Sigma$ and in $\Sigma^{\beta(\pi)}$.

**Theorem 11.** *Algorithm 12 is correct: for each transition $t_j \in T$, the algorithm returns true iff $t_j$ is 1-live in $\Sigma$.*

*Proof.* As the first step, we show that if the algorithm returns true, then $t_j$ is executable in $\Sigma$, and in particular the path $\pi = t_1 ... t_j$ returned by the algorithm is a firing sequence of $\Sigma$. We proceed by induction, starting to show that $t_1$ is enabled in $m_{in}$. By contradiction, let us suppose that $t_1$ is not enabled in $m_{in}$. Then, there must be a precondition $p \in {}^\bullet t_1$, such that $p \notin m_{in}$. By the construction, all the elements in ${}^\bullet t_1$ come from agents that have transitions labelled with $\beta(t_1)$, and all these agents are included in $\Sigma^{\beta(\pi)}$; hence, if $t_1$ is enabled in the initial state of $\Sigma^{\beta(\pi)}$, it must be enabled also in $m_{in}$. Let $\pi_i = t_1...t_i$, $i < j$, be a prefix of the firing sequence $\pi$ and $m_i$ the state reached in $\Sigma$ after executing $\pi_i$. We show that $t_{i+1}$ is enabled in $m_i$. By contradiction, let us suppose that $t_{i+1}$ is not enabled in $m_i$. Then there must be a place $p \notin m_i$ and such that $p \in {}^\bullet t_{i+1}$. This place must be also in $\Sigma^{\beta(\pi)}$, since, by construction, $\Sigma^{\beta(\pi)}$ includes all the agents with transitions labelled $\beta(t_{i+1})$, and the preconditions of $t_{i+1}$ on $\Sigma$ cannot belong to any other agent. Let $m_{in}^{\beta(\pi)} t_1 m_1^{\beta(\pi)} ... t_i m_i^{\beta(\pi)}$ the sequence of states and transitions obtained by firing the sequence $t_1...t_i$ in $\Sigma^{\beta(\pi)}$; $p \in m_i^{\beta(\pi)}$, since $t_{i+1}$ can fire after $\pi_i$ in $\Sigma^{\beta(\pi)}$, and there must be an index $k \leq i$ such that $p \in m_r^\alpha$ for each $r \geq k$. If $k = 0$, then $p \in m_0 = m_{in}$, since $m_{in}^{\lambda(\pi)} \subseteq m_{in}$, by the construction. If $k > 0$, then $p \in t_k^\bullet$, and $p \in m_k$. Since the set of preconditions and postconditions of $t_{k+1}...t_i$ is the same in $\Sigma$ and $\Sigma^{\beta(\pi)}$, if $p \in m_i^{\beta(\pi)}$ after the execution of $t_{k+1}...t_i$, then $p \in m_i$ after firing the same sequence.

As the second step, we need to prove that if the algorithm returns false, then $t_j$ is not 1-live in $\Sigma$. This follows from the observation that adding agents to the system can only restrict the possibility of the transitions in $\Sigma^\alpha$ to occur by adding synchronizations constraints. Therefore, if $t_j$ is a transition in $\Sigma^L$, but there is no sequence in $\Sigma^L$ enabling $t_j$, a fortiori there cannot be any sequence in $\Sigma$. □

**Theorem 12.** *Algorithm 12 terminates after a finite number of steps.*

*Proof.* The thesis follows from the finiteness of the number of agents in the system, and of the number of minimal paths. □

Algorithm 12 does not guarantee that the system $\Sigma^{\beta(\pi)}$ to check will be smaller than $\Sigma$, since the two systems may coincide. However, in distributed systems in which each agent interacts with a small subset of other agents of the entire system, it may become a convenient technique. An example of such a system could be represented by a social network, where the number of users is huge, but each of them has a limited number of connections. A toy example is represented in Fig. 5.7. In this case concurrency enlarges the size of the global transition system, without affecting the reduced systems. The required computation can also be reduced by choosing proper heuristics for the function select_path, so that the more convenient paths are selected to be analysed first. A possible criterion could be to select first the paths requiring to add the smallest number of new agents, even when they are longer and with more labels than others. Consider for example the system of agents in Fig. 5.8. The four agents are represented at the top of the figure, and two of their compositions at the bottom. Transition $d$ belongs only to agent AG$_2$ (the second from the left), and there are two minimal sequences

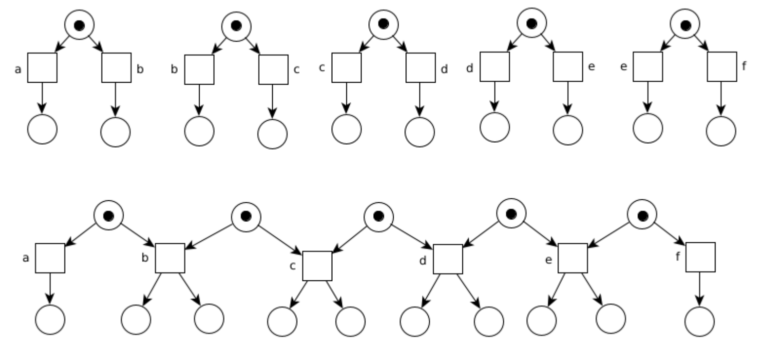


Figure 5.7: Example of a multi-agent system where each agent interacts with a small subset of other agents.

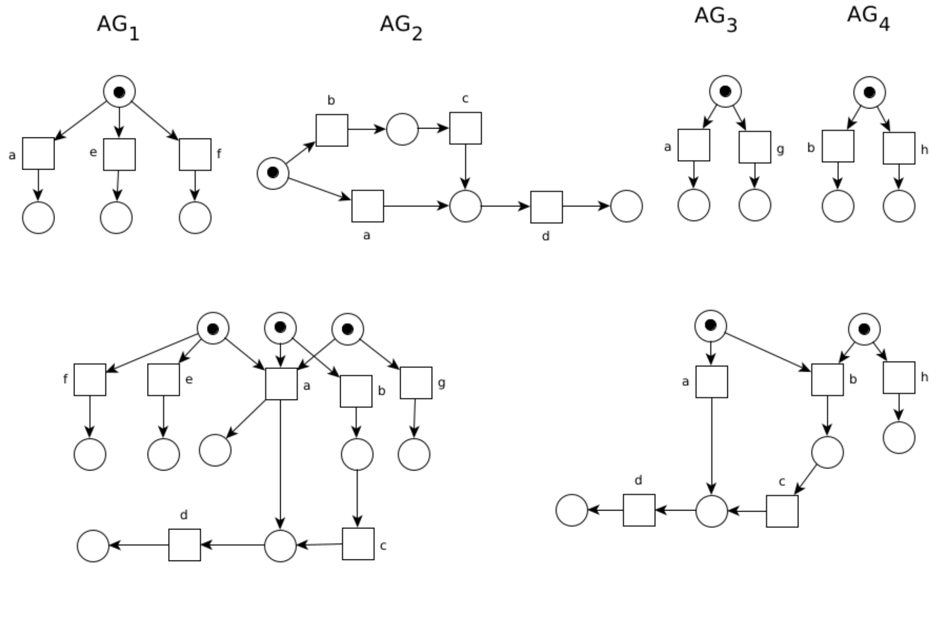


Figure 5.8: Above, four agents part of a multi-agent system; below, composition of $\mathrm{AG}_1$, $\mathrm{AG}_2$, $\mathrm{AG}_3$ on the left, and of $\mathrm{AG}_2$ and $\mathrm{AG}_4$ on the right.

reaching it: $ad$, and $bcd$. Although $ad$ is shorter, the label $a$ is shared by two more agents: $\textsc{ag}_1$ and agent $\textsc{ag}_3$, whereas $c$ belongs only to $\textsc{ag}_2$, and $b$ is shared with $\textsc{ag}_4$ only. The two compositions of agents represented in Fig. 5.8 show $\Sigma^{\{a,d\}}$ (on the left), and $\Sigma^{\{b,c,d\}}$ (on the right). It is easy to see that $\Sigma^{\{b,c,d\}}$ has less reachable states than $\Sigma^{\{a,d\}}$, and it is sufficient to decide the 1-liveness of the transition labelled with $d$.

Note that, for simplicity, we have considered compositions of the entire 1-safe systems modelling the agents. However, an optimization of Algorithm 12 consists in decomposing the agents into sequential components (see [112]) as first step, and then consider their composition. This possibly allows for a further reduction of the resulting composition, since parts of some agents may not be necessary for some labels.

# Chapter 6

# Conclusion and future works

In this thesis, we studied formal methods for the analysis of distributed systems that are not fully observed and/or controlled by a single agent. In particular, we studied how to check which information an agent may obtain on the system through its partial observations of it, and which behaviours the agent is able to force on a system, when it can partially control it. Most of the work focusses on goals achievable by a single agent, despite the possibly opposing behaviour of the rest of the system. In the last part of the thesis we put the basis for the analysis of systems in which several agents interact, each of them with its own observations, control, and goals.

The problems considered in this thesis may have applications in the analysis of noninterference, diagnosis and control of distributed systems.

We now recall the main results presented in the previous chapters and discuss their possible developments.

First, we proposed some formal methods for the analysis of information flow between transitions of a P/T system. We studied the information flow on the basis of two relations: *reveals* and *excludes*. These relations were used in the literature to verify diagnosis and noninterference properties. We defined three new relations generalizing reveals, by assuming that the observing agent may have the ability of counting a certain number of occurrences of a group of transitions. We then provided a group of algorithms to compute both the relations defined in the literature and the new relations on bounded equal-conflict P/T systems. The algorithms are based on the equivalence between maximal-step semantics and step semantics in equal-conflict P/T systems, and on the finiteness of markings in bounded nets. We plan to generalize them in order to allow also to model-check these relations on nets with confusion. We also plan to complete the study of the complexity class of these relations, and to assess whether they are decidable on unbounded nets.

The second problem that we considered consisted in determining whether a certain property can be forced by a user controlling a subset of transitions and observing a subset of places. We modelled this problem as an asynchronous two-player game on the unfolding of the P/T system, where the player *user* has the goal to enforce the property, and the player *environment* works against it. The behaviour of the user is formalized by a strategy, that is winning if the user wins all the plays consistent with it. The strategy of the user is based on its observations on the unfolding. We discussed different notions of observation, and their relation with memory and concurrency, on the basis of some examples. In particular, we defined a notion of stable part of a marking, that we consider particularly useful to model observations in asynchronous systems. This notion keeps into account that some information on the system may change from the time of actual occurrence to the time of receiving it, and therefore, a strategy for the user should not depend on such information. In our use of stable parts, we considered net extended with implicit places. Finding implicit places may be computationally hard: in the general case we need to compute the marking graph and find the regions on it. We plan to consider classes of 1-safe system in which this computation may be avoided, such as systems that can be decomposed in sequential components.

We studied in detail some examples of games, providing algorithms to find winning strategies both on the unfolding and on the marking graph, and comparing our game with the ones defined on concurrent game structures (CGS) [9]. In particular, we showed that when we consider strategies without memory, and the goal of the user is to satisfy an LTL-X formula, our game can be translated into a game on concurrent game structure. When we consider strategies with memory, this relation may not hold, due to the different semantics (interleaving and concurrent) on the two models. We conjecture that when we consider observations based on stable parts of markings, and the user has a memory based on the partial order of the places that it could observe, the game on the unfolding can be translated into a game on CGS. We plan to formalize this notion of observation and to prove our conjecture. We also plan to develop new algorithms to deal with the notions of memory using the elements of the unfolding, and to determine whether we can find a finite bound of memory that is always sufficient to find a winning strategy, if one exists under the assumption of infinite memory.

In addition, we want to extend the work for finding strategies by using a prefix of the unfolding both by considering wider classes of goals and by considering partial observations. Since the marking graph suffers of the state explosion problem, finding solutions on the unfolding may be more convenient; we plan to make experiments to compare the two methods, when possible. Finally, we also plan to merge the approach on the unfolding and the one on the marking graph to obtain more efficient algorithms: a small prefix of the unfolding, such as the complete prefix in [52], can give information about choices that the user should not make; although deriving a full strategy from it seems quite hard, the partial strategy may be used to reduce the number of states of the marking graph, by avoiding transitions that are known to be losing.

Once we have a strategy defining the behaviour of the user, we studied how it can be implemented on the system. In particular, we defined a strong and a weak notion of *implementable strategy*, both consisting in adding some places to the system to constraint the P/T system so that all the maximal runs are associated with plays consistent with the strategy in the original P/T system. In addition, the strong notion requires the vice versa, namely for all plays consistent with the strategy, there must be a maximal run associated to it on the constrained P/T system. We then proposed an algorithm based on region theory to check whether a strategy without memory is strongly implementable on a bounded system, and, in case of positive answer, to return an implementation. In future works, we plan to develop an algorithm also for checking weak implementation, possibly based on the techniques of synthesis approximation. We also plan to characterize implementable strategies, so to recognize them before running the algorithm. We conjecture that strategies defined on stable parts of markings are always strongly implementable. The idea behind this conjecture is that when we consider stable parts of marking, either a controllable transition is chosen in each marking where concurrent uncontrollable transitions are enabled, or it is chosen after their occurrence. In both cases, this can be represented in a Petri net: in the first case in the implementation the transitions remain concurrent, whereas in the second case the transitions will be in sequence, and the occurrence of the controllable transition will depend on the occurrence of the uncontrollable ones.

Another line that we want to follow consists in considering games where the user needs to guarantee some liveness goal, while at the same time keeping some information secret from the environment. In this scenario, the environment observes a subset of transitions in the system, and part of the transitions of the user are secret for it; the user is the central controller on the system, and needs to guarantee some service (expressed with a liveness condition) and avoid that the observations of the environment can reveal some secret transition. This line would join the analysis of information flow with the search of winning strategies.

Finally, we also plan to extend our two-player game setting to a real multi-agent game, both by considering the model-checking of LTL and ATL formulas in which two teams plays against each other, and by considering other game-based logics such as strategy logic (SL) [39], where each agent can have its own goal, not necessarily in competition with the goals of the others. In addition to the model-checking, we plan to study the satisfiability of game-based logics, namely, given a formula in ATL or SL, we want to check whether a multi-agent system satisfying that formula exists. The relatively few works on this topic focusses on the synthesis of multi-agent systems modelled as transition systems (for example, [129] provides results on CGS, whereas [102, 83] works on synchronous multi-agent systems). We plan to extend these results on multi-agent systems modelled as Petri nets. For these lines, we plan to use the framework that we presented in Chap. 5. In that chapter, we showed how to construct a multi-agent system modelled with 1-safe systems with the same behaviour of an AMAS as defined in [79] and how to check if a transition is 1-live in the global model, possibly avoiding to construct it completely. We plan to implement our method with the help of synthesis tools such as Petrify [43], and to compare the efficiency of our approach for 1-liveness with the one used in other existing tools (see for example Evaluator [95]). In addition, we plan to extend our model by allowing for the synthesis of bounded P/T systems, to obtain smaller models by splitting less transitions. Finally, we plan to extend our framework to the multi-agent semantics defined in [93] in the context of assume-guarantee reasoning. In this model transitions are local for each agent, but their occurrence may depend on the value of local states of external agents. This would provide a common framework based on Petri nets for the analysis of asynchronous multi-agent systems with action-oriented and data-oriented synchronizations.

# Bibliography

[1] Federica Adobbati, Luca Bernardinello, Görkem Kılınç, and Lucia Pomello. Computing a parametric reveals relation for bounded equal-conflict Petri nets. *submitted to Transactions on Petri Nets and Other Models of Concurrency*, 2022.

[2] Federica Adobbati, Luca Bernardinello, and Lucia Pomello. Asynchronous games on Petri nets and ATL. *CoRR*, abs/2107.06866, 2021.

[3] Federica Adobbati, Luca Bernardinello, and Lucia Pomello. A two-player asynchronous game on fully observable Petri nets. *Transactions on Petri Nets and Other Models of Concurrency*, 15:126–149, 2021.

[4] Federica Adobbati, Luca Bernardinello, and Lucia Pomello. Looking for winning strategies in two-player games on Petri nets with partial observability. *CoRR*, abs/2204.01603, 2022.

[5] Federica Adobbati, Luca Bernardinello, Lucia Pomello, and Riccardo Stramare. Implementable strategies for a two-player asynchronous game on Petri nets. In *ATAED@ Petri Nets*, pages 69–75, 2022. *Extended version submitted to Transactions on Petri Nets and Other Models of Concurrency.*

[6] Federica Adobbati, Luca Bernardinello, Görkem Kılınç Soylu, and Lucia Pomello. Information flow among transitions of bounded equal-conflict Petri nets. *PNSE@ Petri Nets*, pages 60–79, 2022.

[7] Federica Adobbati, Görkem Kılınç Soylu, and Adrián Puerto Aubel. A finite prefix for analyzing information flow among transitions of a free-choice net. *IEEE Access*, 10:38483–38501, 2022.

[8] Federica Adobbati and Łukasz Mikulski. Analysing multi-agent systems using 1-safe Petri nets. In *PNSE@ Petri Nets*, volume 3170 of *CEUR Workshop Proceedings*, pages 139–155. CEUR-WS.org, 2022.

[9] Rajeev Alur, Thomas A. Henzinger, and Orna Kupferman. Alternating-time temporal logic. *Journal of ACM*, 49(5):672–713, 2002.

[10] Rajeev Alur, Thomas A. Henzinger, Freddy Y.C. Mang, Shaz Qadeer, Sriram K. Rajamani, and Serdar Tasiran. Mocha: Modularity in model checking. In *International Conference on Computer Aided Verification*, pages 521–525. Springer, 1998.

[11] Eric Badouel, Luca Bernardinello, and Philippe Darondeau. *Petri Net Synthesis*. Springer, 2015.

[12] Sandie Balaguer, Thomas Chatain, and Stefan Haar. Building tight occurrence nets from reveals relations. In Benoît Caillaud, Josep Carmona, and Kunihiko Hiraishi, editors, *Proc. 11th ACSD, Newcastle Upon Tyne, UK*, pages 44–53. IEEE, 2011.

[13] Paolo Baldan and Alberto Carraro. Non-interference by unfolding. In *Application and Theory of Petri Nets and Concurrency*, pages 190–209. Springer, 2014.

[14] Francesco Basile. Overview of fault diagnosis methods based on Petri net models. In *2014 European Control Conference (ECC)*, pages 2636–2642. IEEE, 2014.

[15] Francesco Basile, Gianmaria De Tommasi, and Claudio Sterle. Noninterference enforcement via supervisory control in bounded Petri nets. *IEEE Transactions on Automatic Control*, 66(8):3653–3666, Sep 2021.

[16] Albert Benveniste, Eric Fabre, Stefan Haar, and Claude Jard. Diagnosis of asynchronous discrete-event systems: a net unfolding approach. *IEEE Transactions on Automatic Control*, 48(5):714–727, 2003.

[17] Béatrice Bérard, Stefan Haar, Sylvain Schmitz, and Stefan Schwoon. The complexity of diagnosability and opacity verification for Petri nets. *Fundamenta Informaticae*, 161(4):317–349, 2018.

[18] Béatrice Bérard, Serge Haddad, Mathieu Sassolas, and Nathalie Sznajder. Concurrent games on vass with inhibition. In *CONCUR*, volume 12, pages 39–52. Springer, 2012.

[19] Luca Bernardinello. Synthesis of net systems. In Marco Ajmone Marsan, editor, *Application and Theory of Petri Nets 1993*, pages 89–105, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[20] Luca Bernardinello, Görkem Kılınç, and Lucia Pomello. Weak observable liveness and infinite games on finite graphs. In *International Conference on Application and Theory of Petri Nets and Concurrency*, pages 181–199. Springer, 2017.

[21] Luca Bernardinello, Görkem Kılınç, and Lucia Pomello. Non-interference notions based on reveals and excludes relations for Petri

nets. *Transactions on Petri Nets and Other Models of Concurrency*, 11:49–70, 2016.

[22] Luca Bernardinello, Lucia Pomello, Adrián Puerto Aubel, and Alessandro Villa. Checking weak observable liveness on unfoldings through asynchronous games. In *PNSE@ Petri Nets/ACSD*, pages 15–34, 2018.

[23] Eike Best. Structure theory of Petri nets: the free choice hiatus. In *Proc. Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part I, Proc. of an Advanced Course, Bad Honnef, Germany*, volume 254 of *LNCS*, pages 168–205. Springer, 1986.

[24] Eike Best and Philippe Darondeau. Petri net distributability. In *International Andrei Ershov Memorial Conference on Perspectives of System Informatics*, pages 1–18. Springer, 2011.

[25] Eike Best and Raymond Devillers. Sequential and concurrent behaviour in Petri net theory. *Theoretical Computer Science*, 55(1):87–136, 1987.

[26] Raven Beutner, Bernd Finkbeiner, and Jesko Hecking-Harbusch. Translating asynchronous games for distributed synthesis. In *International Conference on Concurrency Theory CONCUR*, 2019.

[27] Blai Bonet, Patrik Haslum, Victor Khomenko, Sylvie Thiébaux, and Walter Vogler. Recent advances in unfolding technique. *Theoretical Computer Science*, 551:84–101, 2014.

[28] Pierre Bouvier and Hubert Garavel. Efficient algorithms for three reachability problems in safe Petri nets. In *International Conference on Applications and Theory of Petri Nets and Concurrency*, pages 339–359. Springer, 2021.

[29] Thomas Brihaye, Arnaud Da Costa, François Laroussinie, and Nicolas Markey. Atl with strategy contexts and bounded memory. In *International Symposium on Logical Foundations of Computer Science*, pages 92–106. Springer, 2009.

[30] Jeremy Bryans, Maciej Koutny, and Peter Y. A. Ryan. Modelling opacity using Petri nets. *Electronic Notes in Theoretical Computer Science*, 121:101–115, 2005.

[31] Jeremy W. Bryans, Maciej Koutny, Laurent Mazaré, and Peter Y.A. Ryan. Opacity generalised to transition systems. *International Journal of Information Security*, 7(6):421–435, 2008.

[32] Nadia Busi and Roberto Gorrieri. A survey on non-interference with Petri nets. In *Advanced Course on Petri Nets*, pages 328–344. Springer, 2003.

[33] Nadia Busi and Roberto Gorrieri. Structural non-interference in elementary and trace nets. *Mathematical Structures in Computer Science*, 19(6):1065–1090, 2009.

[34] Maria Paola Cabasino, Alessandro Giua, Marco Pocci, and Carla Seatzu. Discrete event diagnosis using labeled Petri nets. an application to manufacturing systems. *Control Engineering Practice*, 19(9):989–1001, 2011.

[35] Josep Carmona. The label splitting problem. In *Transactions on Petri Nets and Other Models of Concurrency VI*, pages 1–23. Springer, 2012.

[36] Christos G. Cassandras and Stéphane Lafortune. *Introduction to discrete event systems.* Springer, 2021.

[37] Jérémie Chalopin and Victor Chepoi. 1-safe Petri nets and special cube complexes: equivalence and applications. *ACM Transactions on Computational Logic (TOCL)*, 20(3):1–49, 2019.

[38] Thomas Chatain and Loïc Paulevé. Goal-driven unfolding of Petri nets. In Roland Meyer and Uwe Nestmann, editors, *International Conference on Concurrency Theory, CONCUR*, volume 85 of *LIPIcs*, pages 18:1–18:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[39] Krishnendu Chatterjee, Thomas A Henzinger, and Nir Piterman. Strategy logic. *Information and Computation*, 208(6):677–693, 2010.

[40] Allan Cheng. *Complexity results for model checking.* Citeseer, 1995.

[41] Allan Cheng, Javier Esparza, and Jens Palsberg. Complexity results for 1-safe nets. *Theoretical Computer Science*, 147(1-2):117–136, 1995.

[42] Edmund M Clarke Jr, Orna Grumberg, Daniel Kroening, Doron Peled, and Helmut Veith. *Model checking.* MIT press, 2018.

[43] Jordi Cortadella, Michael Kishinevsky, Alex Kondratyev, Luciano Lavagno, and Alexandre Yakovlev. Petrify: a tool for manipulating concurrent specifications and synthesis of asynchronous controllers. *IEICE Transactions on information and Systems*, 80(3):315–325, 1997.

[44] Wojciech Czerwiński, Sławomir Lasota, Ranko Lazić, Jérôme Leroux, and Filip Mazowiecki. The reachability problem for Petri nets is not elementary. *Journal of the ACM (JACM)*, 68(1):1–28, 2020.

[45] Jörg Desel and Görkem Kılınç. Observable liveness of Petri nets. *Acta Informatica*, 52(2):153–174, 2015.

[46] Jörg Desel and Javier Esparza. *Free Choice Petri Nets*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 1995.

[47] Mariagrazia Dotoli, Maria Pia Fanti, Agostino Marcello Mangini, and Walter Ukovich. On-line fault detection in discrete event systems by Petri nets and integer linear programming. *Automatica*, 45(11):2665–2672, 2009.

[48] Andrzej Ehrenfeucht and Grzegorz Rozenberg. Partial (set) 2-structures. part II: state spaces of concurrent systems. *Acta Informatica*, 27(4):343–368, 1990.

[49] Joost Engelfriet. Branching processes of Petri nets. *Acta Informatica*, 28(6):575–591, 1991.

[50] Javier Esparza. Decidability and complexity of Petri net problems—an introduction. *Advanced Course on Petri Nets*, pages 374–428, 1996.

[51] Javier Esparza and Keijo Heljanko. Implementing LTL model checking with net unfoldings. In Matthew Dwyer, editor, *Model Checking Software*, pages 37–56, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[52] Javier Esparza, Stefan Römer, and Walter Vogler. An improvement of mcmillan's unfolding algorithm. *Formal Methods Syst. Des.*, 20(3):285–310, 2002.

[53] Mariken H.C. Everdij, Margriet B. Klompstra, Henk A.P. Blom, and Bart Klein Obbink. Compositional specification of a multi-agent system by stochastically and dynamically coloured Petri nets. In *Stochastic hybrid systems*, pages 325–350. Springer, 2006.

[54] Dirk Fahland, Cédric Favre, Barbara Jobstmann, Jana Koehler, Niels Lohmann, Hagen Völzer, and Karsten Wolf. Instantaneous soundness checking of industrial business process models. In *International Conference on Business Process Management*, pages 278–293. Springer, 2009.

[55] Bernd Finkbeiner, Manuel Gieseking, Jesko Hecking-Harbusch, and Ernst-Rüdiger Olderog. Global winning conditions in synthesis of distributed systems with causal memory. In *30th EACSL Annual Conference on Computer Science Logic (CSL)*, volume 216, page 20. Schloss Dagstuhl—Leibniz-Zentrum für Informatik, 2022.

[56] Bernd Finkbeiner, Manuel Gieseking, and Ernst-Rüdiger Olderog. Adam: Causality-based synthesis of distributed systems. In *Computer Aided Verification: 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I 27*, pages 433–439. Springer, 2015.

[57] Bernd Finkbeiner and Paul Gölz. Synthesis in distributed environments. In Satya V. Lokam and R. Ramanujam, editors, *37th IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS*, volume 93 of *LIPIcs*, pages 28:1–28:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017.

[58] Bernd Finkbeiner and Ernst-Rüdiger Olderog. Petri games: Synthesis of distributed systems with causal memory. In Adriano Peron and Carla Piazza, editors, *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.

[59] Bernd Finkbeiner and Ernst-Rüdiger Olderog. Petri games: Synthesis of distributed systems with causal memory. *Information and Computation*, 253:181–203, 2017.

[60] Alan K. Galan and Albert D. Baker. Multi-agent communication in jafmas. *Trans: Propose to Q3*, 5(S6):S7, 1999.

[61] Hubert Garavel. Nested-unit Petri nets. *Journal of Logical and Algebraic Methods in Programming*, 104:60–85, 2019.

[62] Blaise Genest, Hugo Gimbert, Anca Muscholl, and Igor Walukiewicz. Asynchronous games over tree architectures. In *Automata, Languages, and Programming: 40th International Colloquium, ICALP 2013, Riga, Latvia, July 8-12, 2013, Proceedings, Part II 40*, pages 275–286. Springer, 2013.

[63] A. Ghaffari, N. Rezg, and Xiaolan Xie. Design of a live and maximally permissive Petri net controller using the theory of regions. *IEEE Transactions on Robotics and Automation*, 19(1):137–141, 2003.

[64] Manuel Gieseking, Jesko Hecking-Harbusch, and Ann Yanich. A web interface for Petri nets with transits and Petri games. In *Tools and Algorithms for the Construction and Analysis of Systems: TACAS*, pages 381–388. Springer, 2021.

[65] Alessandro Giua and Manuel Silva. Petri nets and automatic control: A historical perspective. *Annual Reviews in Control*, 45:223–239, 2018.

[66] Joseph A. Goguen and José Meseguer. Security policies and security models. In *IEEE Symposium on Security and Privacy*, pages 11–20, 1982.

[67] Erich Grädel, Wolfgang Thomas, and Thomas Wilke. *Automata, logics, and infinite games: a guide to current research*, volume 2500. Springer Science & Business Media, 2002.

[68] Ye Guo, Xiaoning Jiang, Chen Guo, Shouguang Wang, and Oussama Karoui. Overview of opacity in discrete event systems. *IEEE Access*, 8:48731–48741, 2020.

[69] Julian Gutierrez and Glynn Winskel. On the determinacy of concurrent games on event structures with infinite winning sets. *Journal of Computer and System Sciences*, 80(6):1119–1137, 2014.

[70] Stefan Haar. Unfold and cover: Qualitative diagnosability for Petri nets. In *Proc. 46th IEEE Conf. Decis. Control*, 2007.

[71] Stefan Haar, Christian Kern, and Stefan Schwoon. Computing the reveals relation in occurrence nets. *Theoretical Computer Science*, 493:66–79, 2013.

[72] Stefan Haar, César Rodríguez, and Stefan Schwoon. Reveal your faults: It's only fair! In *Proc. 13th ACSD, Barcelona, Spain*, pages 120–129, 2013.

[73] Michel H.T. Hack. Analysis of production schemata by Petri nets. Technical report, Massachusetts Inst. of Tech. Cambridge Project MAC, 1972.

[74] Michel H.T. Hack. The equality problem for vector addition systems is undecidable. *Theoretical Computer Science*, 2(1):77–95, 1976.

[75] Paul Hannibal and Ernst-Rüdiger Olderog. The synthesis problem for repeatedly communicating Petri games. In *Application and Theory of Petri Nets and Concurrency*, pages 236–257. Springer, 2022.

[76] Jesko Hecking-Harbusch. *Synthesis of asynchronous distributed systems from global specifications.* PhD thesis, Saarländische Universitäts-und Landesbibliothek, 2021.

[77] Kunihiko Hiraishi. A Petri-net-based model for the mathematical analysis of multi-agent systems. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, 84(11):2829–2837, 2001.

[78] Thomas Hujsa, Jean-Marc Delosme, and Alix Munier-Kordon. On liveness and reversibility of equal-conflict Petri nets. *Fundamenta Informaticae*, 146(1):83–119, 2016.

[79] Wojciech Jamroga, Wojciech Penczek, Teofil Sidoruk, Piotr Dembinski, and Antoni W. Mazurkiewicz. Towards partial order reductions for strategic ability. *Journal of Artificial Intelligence Research*, 68:817–850, 2020.

[80] Ryszard Janicki, Peter E. Lauer, Maciej Koutny, and Raymond Devillers. Concurrent and maximally concurrent evolution of nonsequential systems. *Theoretical Computer Science*, 43:213–238, 1986.

[81] Neil D. Jones, Lawrence H. Landweber, and Edmund Y. Lien. Complexity of some problems in Petri nets. *Theoretical Computer Science*, 4(3):277–299, 1977.

[82] Guy-Vincent Jourdan and Gregor Bochmann. On testing 1-safe Petri nets. In *2009 Third IEEE International Symposium on Theoretical Aspects of Software Engineering*, pages 275–281. IEEE, 2009.

[83] Magdalena Kacprzak, Artur Niewiadomski, and Wojciech Penczek. Satisfiability checking of strategy logic with simple goals. In *Proceedings of the International Conference on Principles of Knowledge Representation and Reasoning*, volume 18, pages 400–410, 2021.

[84] Shyam Lal Karra, Kim Guldstrand Larsen, Marco Muniz, and Jiří Srba. On-the-fly synthesis for strictly alternating games. In *Applications and Theory of Petri Nets and Concurrency*, pages 109–128. Springer, 2020.

[85] Victor Khomenko, Maciej Koutny, and Walter Vogler. Canonical prefixes of Petri net unfoldings. *Acta Informatica*, 40(2):95–118, 2003.

[86] Görkem Kılınç. *Formal Notions of Non-interference and Liveness for Distributed Systems.* PhD thesis, Univ. Milano-Bicocca, DISCo, Milano, Italy, 2016.

[87] Kais Klai, Laure Petrucci, and Michel Reniers. An incremental and modular technique for checking LTL\ X properties of Petri nets. In *International Conference on Formal Techniques for Networked and Distributed Systems*, pages 280–295. Springer, 2007.

[88] Sofia Kouah, Djamel-Eddine Saïdouni, and Jean-Michel Ilié. Synchronized Petri net: A formal specification model for multi agent systems. *Journal of Software*, 8(3):587–602, 2013.

[89] Damian Kurpiewski, Witold Pazderski, Wojciech Jamroga, and Yan Kim. Stv+ reductions: Towards practical verification of strategic ability using model reductions. In *Proceedings of the 20th International Conference on Autonomous Agents and MultiAgent Systems*, pages 1770–1772, 2021.

[90] Stéphane Lafortune, Feng Lin, and Christoforos N. Hadjicostis. On the history of diagnosability and opacity in discrete event systems. *Annual Reviews in Control*, 45:257–266, 2018.

[91] Zhiwu Li, MengChu Zhou, and MuDer Jeng. A maximally permissive deadlock prevention policy for fms based on Petri net siphon control and the theory of regions. *IEEE Transactions on Automation Science and Engineering*, 5(1):182–188, 2008.

[92] Alessio Lomuscio, Hongyang Qu, and Franco Raimondi. MCMAS: an open-source model checker for the verification of multi-agent systems. *Int. J. Softw. Tools Technol. Transf.*, 19(1):9–30, 2017.

[93] Alessio Lomuscio, Ben Strulo, Nigel Walker, and Peng Wu. Assume-guarantee reasoning with local specifications. *International Journal of Foundations of Computer Science*, 24(04):419–444, 2013.

[94] Robert Lukomski and Kazimierz Wilkosz. Modeling of multi-agent system for power system topology verification with use of Petri nets. In *2010 Modern Electric Power Systems*, pages 1–6. IEEE, 2010.

[95] Radu Mateescu. Specification and analysis of asynchronous systems using CADP, 2008.

[96] Laurent Mazaré. Using unification for opacity properties. In *Proc. WITS'04, Barcelona, Spain*, pages 165–176, 2004.

[97] Model Checking Contest 2023. `https://mcc.lip6.fr/2023/models.php`. Last visited on 2022-11-28.

[98] Robert McNaughton. Infinite games played on finite graphs. *Annals of Pure and Applied Logic*, 65(2):149–184, 1993.

[99] Carlos Molinero and Manuel Núñez. Planning of work schedules through the use of a hierarchical multi-agent system. *Automation in Construction*, 20(8):1227–1241, 2011.

[100] Madhavan Mukund. Petri nets and step transition systems. *International Journal of Foundations of Computer Science*, 3(04):443–478, 1992.

[101] Tadao Murata. Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580, 1989.

[102] Artur Niewiadomski, Magdalena Kacprzak, Damian Kurpiewski, Michał Knapik, Wojciech Penczek, and Wojciech Jamroga. Msatl: A tool for sat-based atl satisfiability checking. In *Proceedings of 19th International Conference on Autonomous Agents and Multiagent Systems AAMAS 2020*, 2020.

[103] Yen-Liang Pan, Yi-Sheng Huang, MuDer Jeng, and Sheng-Luen Chung. Enhancement of an efficient control policy for fmss using the theory of regions and selective siphon method. *The International Journal of Advanced Manufacturing Technology*, 66(9):1805–1815, 2013.

[104] James L. Peterson. Petri nets. *ACM Comput. Surv.*, 9(3):223–252, Sep. 1977.

[105] Carl Adam Petri. *Kommunikation mit Automaten.* PhD thesis, Hamburg Univ., Germany, 1962.

[106] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57. IEEE, 1977.

[107] Shiladitya Pujari and Sripati Mukhopadhyay. Petri net: A tool for modeling and analyze multi-agent oriented systems. *International Journal of Intelligent Systems and Applications*, 4(10):103, 2012.

[108] Peter J.G. Ramadge and W. Murray Wonham. Supervisory control of a class of discrete event processes. *SIAM journal on control and optimization*, 25(1):206–230, 1987.

[109] Peter J.G. Ramadge and W. Murray Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1):81–98, 1989.

[110] Nidhal Rezg, Xiaolan Xie, and Asma Ghaffari. Supervisory control in discrete event systems using the theory of regions. In *Discrete Event Systems*, pages 391–398. Springer, 2000.

[111] Sadok Rezig, Chekib Ghorbel, Zied Achour, and Nidhal Rezg. PLC-based implementation of supervisory control for flexible manufacturing systems using theory of regions. *International Journal of Automation and Control*, 13(5):619–640, 2019.

[112] Grzegorz Rozenberg and Joost Engelfriet. Elementary net systems. In *Advanced Course on Petri Nets*, pages 12–121. Springer, 1996.

[113] Meera Sampath, Raja Sengupta, Stéphane Lafortune, Kasim Sinnamohideen, and Demosthenis Teneketzis. Diagnosability of discrete-event systems. *IEEE Transactions on automatic control*, 40(9):1555–1575, 1995.

[114] Uli Schlachter and Harro Wimmel. Relabelling lts for Petri net synthesis via solving separation problems. In *Transactions on Petri Nets and Other Models of Concurrency XIV*, pages 222–254. Springer, 2019.

[115] Pierre-Yves Schobbens. Alternating-time logic with imperfect recall. *Electronic Notes in Theoretical Computer Science*, 85(2):82–93, 2004.

[116] Claus Schröter and Victor Khomenko. Parallel LTL-X model checking of high-level Petri nets based on unfoldings. In *International Conference on Computer Aided Verification*, pages 109–121. Springer, 2004.

[117] SMART: Stochastic Model-checking Analyzer for Reliability and Timing. `https://asminer.github.io/smart/`. Last visited on 2022-11-28.

[118] Einar Smith. On the border of causality: Contact and confusion. *Theoretical Computer Science*, 153(1&2):245–270, 1996.

[119] Robert S. Streett. Propositional dynamic logic of looping and converse. In *Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 375–383, 1981.

[120] Enrique Teruel and Manuel Silva. Liveness and home states in equal conflict systems. In Marco Ajmone Marsan, editor, *Application and Theory of Petri Nets 1993*, pages 415–432, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[121] P. S. Thiagarajan. Elementary net systems. In Wilfried Brauer, Wolfgang Reisig, and Grzegorz Rozenberg, editors, *Petri Nets: Central Models and Their Properties, Advances in Petri Nets 1986, Part I, Proceedings of an Advanced Course, Bad Honnef, Germany, 8-19 September 1986*, volume 254 of *Lecture Notes in Computer Science*, pages 26–59. Springer, 1986.

[122] Yin Tong, Zhiwu Li, Carla Seatzu, and Alessandro Giua. Current-state opacity enforcement in discrete event systems under incomparable observations. *Discrete Event Dynamic Systems*, 28(2):161–182, 2018.

[123] W. van der Aalst, T. Weijters, and L. Maruster. Workflow mining: discovering process models from event logs. *IEEE Trans. Knowl. Data Eng.*, 16(9):1128–1142, 2004.

[124] Wil M. P. van der Aalst. *Process Mining - Data Science in Action, 2nd ed.* Springer, 2016.

[125] Wil M. P. van der Aalst. Free-choice nets with home clusters are lucent. *Fundam. Inform.*, 181(4):273–302, 2021.

[126] Wil M. P. van der Aalst. Reduction using induced subnets to systematically prove properties for free-choice nets. In Didier Buchs and Josep Carmona, editors, *Proc. Petri NETS 2021, Paris, France*, volume 12734 of *LNCS*, pages 208–229. Springer, 2021.

[127] Wil M. P. van der Aalst. Using free-choice nets for process mining and business process management. In Maria Ganzha, Leszek A. Maciaszek, Marcin Paprzycki, and Dominik Slezak, editors, *Proc. 16th FedCSIS*, pages 9–15, 2021.

[128] Rob J. van Glabbeek, Ursula Goltz, and Jens-Wolfhard Schicke-Uffmann. On characterising distributability. *Logical Methods in Computer Science*, 9(3), 2013.

[129] Moshe Y Vardi, Giuseppe Perelli, Aniello Murano, and Fabio Mogavero. Reasoning about strategies: On the satisfiability problem. *Logical Methods in Computer Science*, 13, 2017.

[130] Michael Weber and Ekkart Kindler. The Petri net markup language. In *Petri Net Technology for communication-based systems*, pages 124–144. Springer, 2003.

[131] Marcin Wojnakowski, Remigiusz Wiśniewski, Grzegorz Bazydło, and Mateusz Popławski. Analysis of safeness in a Petri net-based specification of the control part of cyber-physical systems. *International Journal of Applied Mathematics and Computer Science*, 31(4), 2021.

[132] Karsten Wolf. How Petri net theory serves Petri net model checking: A survey. In *Transactions on Petri Nets and Other Models of Concurrency XIV*, pages 36–63. Springer, 2019.

[133] Janan Zaytoon and Stéphane Lafortune. Overview of fault diagnosis methods for discrete event systems. *Annual Reviews in Control*, 37(2):308–320, 2013.

[134] Wieslaw Zielonka. Notes on finite asynchronous automata. *RAIRO-Theoretical Informatics and Applications*, 21(2):99–135, 1987.

[135] Vittorio A. Ziparo, Luca Iocchi, Pedro U. Lima, Daniele Nardi, and Pier Francesco Palamara. Petri net plans. *Autonomous Agents and Multi-Agent Systems*, 23(3):344–383, 2011.