Taming Confusion for Modeling and Implementing Probabilistic Concurrent Systems*

Joost-Peter Katoen[1,2] and Doron Peled[3]

[1] Software Modeling and Verification Group
RWTH Aachen University
D-52056 Aachen, Germany
[2] Formal Methods and Tools Group
University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands
[3] Department of Computer Science
Bar Ilan University
Ramat Gan 52900, Israel

Abstract. In concurrent systems, the choice of executing the next transition depends both on the timing between the agents that make independent or collaborative interactions available, and on the conflicts (nondeterministic choices) with other transitions. This creates a challenging modeling and implementation problem. When the system needs to make also probabilistic choices, the situation becomes even more complicated. We use the model of Petri nets to demonstrate the modeling and implementation problem. The proposed solution involves adding sequential observers called *agents* to the Petri net structure. Distributed probabilistic choices are facilitated in the presence of concurrency and nondeterminism, by selecting agents that make the choices, while guaranteeing that their view is temporarily stable. We provide a distributed scheduling algorithm for implementing a system that allows distributed probabilistic choice.

1 Introduction

Adding probabilities in the presence of concurrency and nondeterminism is challenging. Autonomous models in which branching probabilities and nondeterminism coexist are well understood. A prominent example is Markov decision processes (MDPs) [21]. Probabilistic automata (PAs) [23], a slight generalization of MDPs, have been equipped with parallel composition in a CSP-like fashion. They constitute a framework for concurrent systems that exhibit both nondeterministic and probabilistic behavior. Examples of such systems are randomized

* The first author is supported by the FP7 MEALS and SENSATION projects. The second author is supported by ISF grant 126/12 "Efficient Synthesis of Control for Concurrent Systems".

M. Felleisen and P. Gardner (Eds.): ESOP 2013, LNCS 7792, pp. 411–430, 2013.

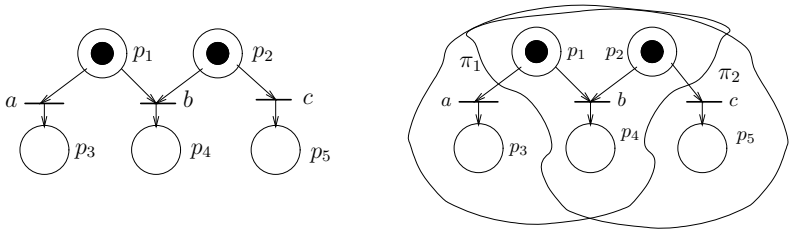

Fig. 1. A Petri net (left) covered by agents (right)

distributed algorithms and network security protocols. There is however a serious anomaly [17]: for concurrent PAs, a global scheduler may establish strong correlations between the behavior of system components and, e.g., resolve choices in one PA based on the outcome of a coin flip in the other.

An example illustrates the issue. Assume two scientists want to write a paper for a forthcoming important conference. Given the deadline, each can be involved in only one paper. Each scientist has his own idea that can be materialized into a single-authored paper a or c. They also have a joint idea for a paper b. The Petri net in Figure 1(left) depicts this situation. As they face a tough choice, they want to use some probabilistic measures to make the decision, e.g., a fair coin. The difficulty in modeling is that the available choices for such a decision may differ depending on the concurrent scheduling. If, say, the outcome of the flipped coin by the first author yields paper a, the selection of the other author is no more probabilistic; he has no choice but to write his own paper c. While writing two independent papers is concurrent, the selection by one author affects (e.g., removes) the alternative(s) for the other; this phenomenon is called *confusion* in Petri nets.

Such a subtle interplay between concurrency and probabilistic choices has recently led to various proposals to remedy or control this phenomenon, e.g., token-based schemes [7] and distributed schedulers [11], with impacts in the context of quantitative security [3] and testing theory [10]. In this paper, we start from an expressive concurrency model—Petri nets—and equip them with branching probabilities. An important advantage of Petri nets is that the artifacts that affect the aforementioned problem such as independence, conflict (or confusion), and concurrency are well studied. (Our concepts can be however also demonstrated similarly with other models.) The challenge with Petri nets is to deal with *confusion*. Indeed, various earlier proposals for probabilistic Petri nets restrict the semantics and/or their analysis to confusion-free nets [2,18,25]. Confusion describes a situation where a nondeterministic choice between transitions is affected (i.e., possibilities are added or removed) by firing an independent transition. This situation is problematic as, e.g., a probabilistic decision can be altered potentially without being observed.

Our proposal is to add information about the structure of a net, by defining what we call *agents*. The transitions of a net are covered by a set of (possibly intersecting) agents. Agents represent processes, observers, or component

automata, which can make both nondeterministic and probabilistic decisions based on their "local" information. The selection of the next agent that can resolve a choice is done nondeterministically, and is supposed to be done by a scheduler. Adding agents to Petri nets yields, what we refer to as, *covered* nets. In Figure 1(right), the two agents π_1 and π_2, indicated by contours that encapsulate their transitions and places, model the two scientists. The selection between which scientist may decide first, say π_1, is done globally and nondeterministically. Agent π_1 can control transitions a and b and observe all places except p_5, i.e., all the input and output places related to its transitions.

Contributions of this paper. The technical contributions of this paper are:

- An extension of Petri nets with *agents*. Agents resolve choices based on a local state of the net, as opposed to resolving choices based on global state information (as is usual in net theory).
- A *semantics* for these nets with a two-level control mechanism: a (global) selection of an agent followed by a (local) choice by this agent. Under this semantics, concurrency occurs when an agent's transition does not affect the choices available to another agent. In contrast with standard net semantics, it identifies confusion as an additional source of dependency.
- The applicability of this framework to a novel notion of *probabilistic* Petri nets where agents are responsible to resolve probabilistic choices locally. This provides a novel and clean treatment of concurrency in the presence of branching probabilities, and naturally yields an MDP.
- A *distributed algorithm* for selecting a set of agents that can resolve their choices in an independent, concurrent fashion. The algorithm is based on a structural analysis of the net (thus is efficient), is deadlock free, and relies on low-level atomicity assumptions. To the best of our knowledge, this is the first algorithm that implements distributed scheduling of concurrent probabilistic systems.

2 Preliminaries

Petri nets. We start by introducing some basic concepts and notations of Petri nets; for more details, see e.g., [22].

Definition 1 (Syntax). *A* 1-safe Petri net N *is a tuple* (P, T, E, s_0) *where*

- P *is a finite set of* places. *The* states *of* N *are defined as* $S = 2^P$.
- T *is a finite set of* transitions.
- $E \subseteq (P \times T) \cup (T \times P)$ *is a bipartite relation between places and transitions.*
- $s_0 \in S$ *(i.e.,* $s_0 \subseteq P$*) is the* initial state.

For a transition $t \in T$*, let the set* ${}^\bullet t$ *of* input places *be* $\{p \in P \mid (p, t) \in E\}$*, and the set* $t^\bullet$ *of* output places *be* $\{p \in P \mid (t, p) \in E\}$*. Similarly, for a place* $p \in P$*, we denote by* $p^\bullet$ *the transitions* $\{t \in T \mid (p, t) \in E\}$*, and by* ${}^\bullet p$ *the transitions* $\{t \in T \mid (t, p) \in E\}$.

Definition 2 (Enabled Transition). *A transition $t \in T$ is* enabled *in a state s, denoted $s[t\rangle$, if ${}^\bullet t \subseteq s$ and $t^\bullet \cap s \subseteq {}^\bullet t$. The set of enabled transitions in a state s is denoted $en(s)$. A state s is in* deadlock *if $en(s) = \emptyset$.*

Definition 3 (Fired Transition). *A transition $t \in en(s)$, i.e., $s[t\rangle$, can* fire *(or* execute*) from state s to state s', denoted by $s[t\rangle s'$, if $s' = (s \setminus {}^\bullet t) \cup t^\bullet$.*

Transitions are visualized as lines, places as circles, and the relation E is represented using arrows. The net in Figure 3 has places p_1, p_2, ..., p_9 and transitions a, b, c, d, e, and f. We depict a state by putting a full circle, called a *token*, inside each place of that state, leaving the other places empty. The net in Figure 3 has the initial state $s_0 = \{p_1, p_2, p_9\}$. The transitions that are enabled in s_0 are a and b. If we fire transition a from that state, the token from place p_1 will be removed, and a token will be placed in p_3. All other tokens reside in their places.

Definition 4 (Execution). *An* execution *of a Petri net N is a maximal (i.e., it cannot be extended) alternating sequence of states and transitions $s_0 t_1 s_1 t_2 s_2 \ldots$, where s_0 is the initial state of N and for all $i \geq 0$, $s_i[t_{i+1}\rangle s_{i+1}$ holds.*

If it is clear from the context, we sometimes use just the sequence of states as executions. A state is *reachable* in a Petri net N if it occurs in at least one of its executions. The *state graph* of N is a digraph where the nodes represent reachable states of N and the edges represent the firing relation. Figure 4 depicts the state graph for the Petri net in Figure 3.

Decomposition into disjoint places. A composition of Petri nets, see Figure 2 (which can also be seen as a decomposition), was given, e.g., by Mazurkiewicz [19]. It combines different Petri nets by unifying their components. This can be seen as synchronizing the executions of transitions of the components that have the same name. The places of the components are disjoint, but this condition can be relaxed (e.g., when modeling systems with shared variables).

Definition 5 (Composition). *Let N^1, N^2, ..., N^n be Petri nets, where $N^i = (P^i, T^i, E^i, s_0^i)$. Then the Mazurkiewicz composition of these nets is the net*

$$N = \Big(\bigcup_{1 \leq i \leq n} P^i, \bigcup_{1 \leq i \leq n} T^i, \bigcup_{1 \leq i \leq n} E^i, \bigcup_{1 \leq i \leq n} s_0^i \Big)$$

This kind of decomposition is also present in the component-based platform BIP (Behavior, Interaction, Priority) [6]. There, the finite state components may be engaged in internal or collaborative transitions. For a distributed implementation, a synchronization algorithm is required for selecting an interaction among several choices, dealing with the complication of guaranteeing a consistent interaction in the presence of different choices by the different participants. Specifically, it is essential to prevent the situation where a component is committed to an interaction, while another participant has meanwhile selected a conflicting interaction. A scheduler such as α-core [20] (that algorithm contains a small

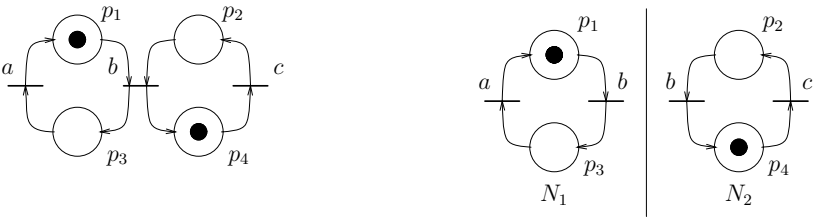

Fig. 2. Petri net composition/decomposition

error, which is corrected in [13]) provides such a guarantee by performing a two-phase exchange of messages for requesting an interaction and committing to it by all participants. In the next section, we propose another perspective on decomposing nets. The key to this decomposition is the concept of an *agent*.

3 Covering Petri Nets by Agents

This section introduces *covered* Petri nets, i.e., nets whose transitions are completely covered by agents. Agents act as entities that resolve nondeterministic (and later in Section 4, probabilistic) choices in a net. They do so on the basis of their local view of the state of the net. Executions of covered nets include in each step an agent that selects one of its enabled transition. It is nondeterministically determined when an agent (which has an enabled transition) gets its turn.

Covered Petri Nets. We first recapitulate some standard notions on transitions.

Definition 6 (Dependent, Conflicting Transition). *Transitions $t_1, t_2 \in T$ are* dependent *(see [19]) if $({}^\bullet t_1 \cup t_1{}^\bullet) \cap ({}^\bullet t_2 \cup t_2{}^\bullet) \neq \emptyset$. Let $D \subseteq T \times T$ be the* dependence *relation. Transitions $t_1, t_2 \in T$ are* independent *if $(t_1, t_2) \notin D$. Let $I = (T \times T) \setminus D$.*

Dependent transitions t_1 and t_2 are conflicting *if $({}^\bullet t_1 \cap {}^\bullet t_2) \cup (t_1{}^\bullet \cap t_2{}^\bullet) \neq \emptyset$. (We often call the transitions that are dependent but not conflicting* sequential*.)*

Dependent transitions have some common place. They are conflicting if they share some input or some output place. For example, the transitions a and b in Figure 1(left) are dependent (and conflicting), and so are transitions b and c. The transitions a and c are independent.

In order to facilitate probabilistic choices in Petri nets (see Section 4), we extend nets with agents that make decisions based on a partial view of the state of the net. This yields covered Petri nets.

Definition 7 (Covered Petri Net). *A* covered Petri net *(in short* CPN*) $\mathcal{C} = (N, \Pi)$ is a net $N = (P, T, E, s_0)$ and a set $\Pi \subset 2^T$ of nonempty sets of transitions with $T = \bigcup_{\pi \in \Pi} \pi$ satisfying:*

1. *If $t, t' \in \pi$ and $(t, t') \in I$, then for no reachable state s of N, $t, t' \in en(s)$.*

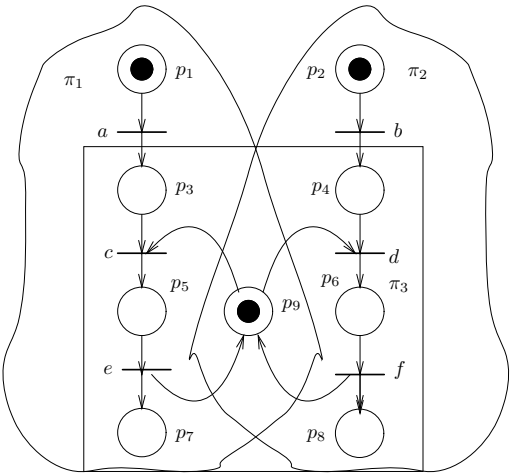

Fig. 3. A Petri net for mutual exclusion

2. *For each* $p \in P$*, there is a* $\pi \in \Pi$ *such that* ${}^\bullet p \subseteq \pi$*, and for any other* $\pi' \in \Pi$*,* $|{}^\bullet p \cap \pi'| \leq 1$*. The same holds when replacing* ${}^\bullet p$ *by* $p^\bullet$*.*

The sets of transitions in Π *satisfying the above constraints are called* agents*.*

Covering of Petri nets appears in [14]. Agents are nonempty and together cover all transitions of the net. A transition can belong to several agents, e.g., when it models a synchronization between agents. For instance, in Figure 1, transition b belongs to both agents π_1 and π_2. Let us explain the above definition in some more detail. The first constraint asserts that no two independent transitions in agent π can ever be executed from the same state[1]. Stated differently, transitions of an agent that can be simultaneously enabled are dependent. The second constraint requires that all input transitions of a place are captured by an agent; the same holds for all its output transitions. In addition, any other agent in Π contains at most one input transition (and similarly, for output transitions).

Example 1. The net in Figure 3 is covered by $\Pi = \{\pi_1, \pi_2, \pi_3\}$ with the *left* agent $\pi_1 = \{a, c, e\}$, the *right* agent $\pi_2 = \{b, d, f\}$, and the third agent $\pi_3 = \{c, d, e, f\}$. Remark that all transitions of agent π_3 are shared with some other agent. The set of agents $\Pi' = \{\pi_1, \pi_2\}$ does not cover the net as ${}^\bullet p_9 = \{e, f\}$ is not captured by a single agent, i.e., there is no single agent π with ${}^\bullet p_9 \subseteq \pi$.

The constraint on the places of the CPN reflects the goal of agents to resolve a choice. Only if $p^\bullet \subseteq \pi$ (${}^\bullet p \subseteq \pi$, respectively), agent π can resolve the choice between transitions that require p to have (not have, respectively) a token. This

[1] This restriction, which we find natural, can be alleviated, but this requires a change in the algorithm presented in Section 5.

explains the choice of agents in Figure 1: $\pi_1 = p_1{}^\bullet = \{a, b\}$, and $\pi_2 = p_2{}^\bullet = \{b, c\}$. In Figure 5, $\pi_1 = \{b, c\}$ can execute c, while $\pi_2 = p_1{}^\bullet = \{a, b\}$ makes the choice between a and b.

Definition 8 (Neighborhood). *The* neighborhood $\mathsf{ngb}(T')$ *of a set* $T' \subseteq T$ *of transitions is the set of places* $\bigcup_{t \in T'} ({}^\bullet t \cup t^\bullet)$.

We will visually represent the separation of transitions of a Petri net into agents using a contour line that encapsulates the neighborhood of its agents. The neighborhood of agent π_1 in Figure 3 is $\{p_1, p_3, p_5, p_7, p_9\}$. Place p_9 belongs to the neighborhood of all indicated agents (i.e., π_1, π_2 and π_3), and acts as a semaphore.

Definition 9 (Local Information). *The* local information *of agent* $\pi \in \Pi$ *of a CPN* $\mathcal{C} = (N, \Pi)$ *in state* s *of* N, *denoted* $s\lceil_\pi$, *is defined by* $s\lceil_\pi = s \cap \mathsf{ngb}(\pi)$.

In the net in Figure 3, the local information of π_1 in any state s equals $s \cap \{p_1, p_3, p_5, p_7, p_9\}$. In the initial state s_0, $s_0\lceil_{\pi_1}$ equals $\{p_1, p_9\}$. After executing transition b, the local information of π_1 does not change. The subsequent execution of transition d removes p_9 from the local information of π_1. The local information of an agent represents the limited view it has regarding the state of the system. This is formalized in the following lemma:

Lemma 1. *Let CPN* $\mathcal{C} = (N, \Pi)$. *For states* s *and* s' *of* N, *and* $\pi \in \Pi$ *we have:*

1. *If* $s\lceil_\pi = s'\lceil_\pi$, *then* $en(s) = en(s')$.
2. *If* $s[t\rangle s'$ *for transition* $t \in \pi$, *then* $s \setminus \mathsf{ngb}(\pi) = s' \setminus \mathsf{ngb}(\pi)$.

CPN Executions. We now define the concept of executions for CPNs. A CPN execution involves not only the states and the enabled transitions fired from them (as for an execution of a Petri net, cf. Definition 4), but also the agents that are selected to decide which of their enabled transition will be fired. We call such a scheduling *agent centric* and in Section 5 provide an algorithm for implementing such scheduling. The basic principle of selecting agents is to give priority to agents that have a more complete view of a nondeterministic choice in the net. This is formalized by the notion of subsumption. Let CPN $\mathcal{C} = (N, \Pi)$ with $\pi, \pi' \in \Pi$.

Definition 10 (Subsumption). *An agent* π subsume*s an agent* π' *over transition* $t \in \pi \cap \pi'$, *denoted* $\pi \gg_t \pi'$, *if the following conditions hold:*

1. *For each state* s *such that* $t \in en(s)$, $|en(s) \cap \pi'| = 1$, *i.e., there is no alternative choice for* π' *in* s *besides* t, *and*
2. *There is at least one state* s *such that* $t \in en(s)$ *and* $|en(s) \cap \pi| > 1$, *i.e.,* π *has a nondeterministic choice in* s *that includes* t.

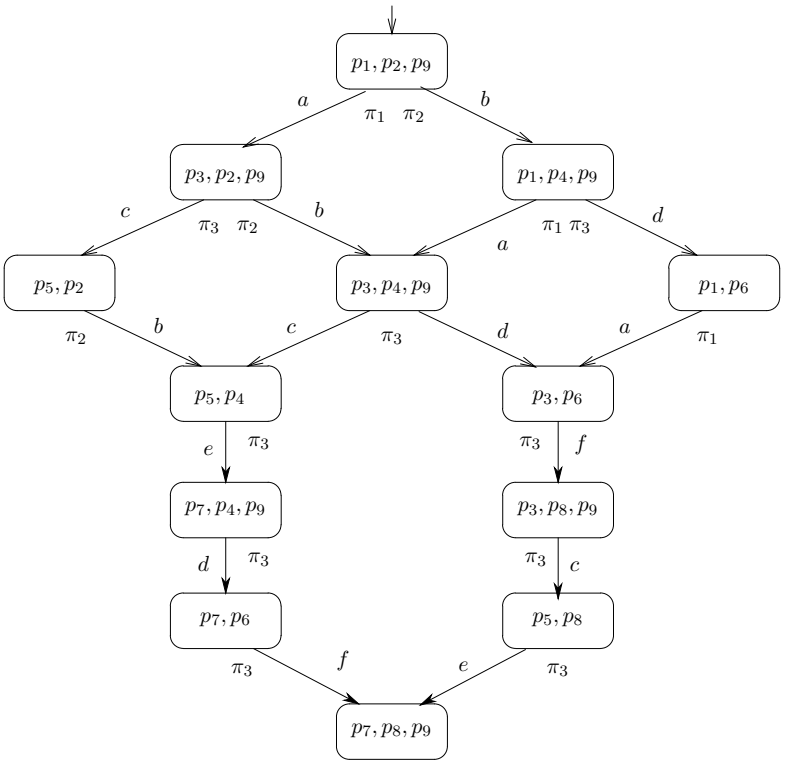

Fig. 4. State graph for the net (mutual exclusion) in Figure 3

In the following definition, an execution of a CPN only allows a transition t to be fired by an agent π' if there is no other agent π that subsumes π' over t. That is to say, if $\pi \gg_t \pi'$, agent π' will never be selected to make a decision to fire transition t. The justification for this notion is that its view is more restricted than that of agent π, which observes more alternatives to t.

In Figure 3, we have that $\pi_3 \gg_c \pi_1$ and (by symmetry) $\pi_3 \gg_d \pi_2$. In the notion of CPN execution defined below, agent π_3 resolves the nondeterministic choice between c and d, rather than agents π_1 or π_2. In a sense, we can remove the transitions c and d from the scope of agent π_1 and π_2, respectively. However, this will create a hole in the structure of the agent π_1 and π_2. (The same applies to the choice between the transitions e and f, although in this case the choice is somewhat fake as it is clear from the structure of the net that only one of these transitions can be enabled in all states.)

Definition 11 (CPN Execution). *An execution of a CPN $\mathcal{C} = (N, \Pi)$ is a maximal sequence $s_0[\pi_1|t_1\rangle s_1[\pi_2|t_2\rangle s_2 \ldots$ where s_0 is the initial state of N, for all $i \geq 0$, $s_i[t_{i+1}\rangle s_{i+1}$, $\pi_i \in \Pi$, $t_i \in \pi_i$ and there is no $\pi \in \Pi$ such that $\pi \gg_{t_i} \pi_i$.*

The choice between admissible agents (agents that currently have an enabled transition t and that are not subsumed by another one over t) is performed in a nondeterministic way.

The subsumption relation $\pi \gg_t \pi'$ is static: it does not depend on the current state. For simplicity of the presentation, we will remove henceforth from the description of an agent the transitions that it cannot execute due to subsumption. Thus, this leaves, for the CPN in Figure 3, $\pi_1 = \{a\}$ and $\pi_2 = \{b\}$.

Confusion and Weak Places. In the sequel of this section, we recall some standard notions from Petri net theory (such as confusion and concurrency), and introduce some new notions that become relevant for the scheduling algorithm in Section 5.

Definition 12 (Confusion). *The quadruple (t_1, t_2, t_3, s) is a* confusion occurrence *in state s if transitions t_1 and t_2 are conflicting, $(t_1, t_3) \in I$, $t_1, t_3 \in en(s)$, and the execution of t_3 (from s) changes the enabledness of t_2.*[2] *The pair $(t, t') \in T \times T$ is a* confusion *if there exists some transition $t'' \in T$ and a state s where (t, t'', t', s) is a confusion occurrence.*

Confusion appears in the nets in Figures 1 and 5. These are two classical examples of confusion, where the first one is symmetric and the second one is asymmetric. In Figure 1, (a, b, c, s_0) and (c, b, a, s_0) are confusion occurrences for the initial state $s_0 = \{p_1, p_2\}$. The former is due to the fact that a and b are conflicting and the firing of c disables b; the second is due to the conflict between c and b and firing a disables b. This gives (a, c) and (c, a) as (symmetric) confusions, respectively. In Figure 5, (a, b, c, s_0) is a confusion occurrence for the initial state $s_0 = \{p_1, p_2\}$ as firing c enables b. Thus, (a, c) is a confusion. Since (c, a) is not a confusion, this confusion is asymmetric. A confusion occurrence may not be detectable by considering the local information of an agent; e.g., in the net of Figure 5, neither agent π_1 nor agent π_2 can locally detect that the current (initial) state is a confusion occurrence.

Definition 13 (Confusion Pivot). *A place $p \in P$ is* pivotal *for a confusion (t, t'), if there is a confusion occurrence $(t, \hat{t}, t', s)$ such that firing t' from s changes the value of a place $p \in {}^\bullet\hat{t} \cup \hat{t}^\bullet$. We denote the pivotal places for a confusion (t, t') by $\mathsf{pivotal}(t, t')$. For $T' \subseteq T$, let $\mathsf{pivot}(T') = \bigcup_{t' \in T', t \in T} \mathsf{pivotal}(t', t)$.*

Intuitively, the pivot places are the places that are changing during the occurrence of the confusion, to create or to eliminate some choice, by the firing of an independent transition. In Figure 1, $\mathsf{pivot}(\pi_1) = \{p_2\}$ and $\mathsf{pivot}(\pi_2) = \{p_1\}$. In Figure 5, p_5 is pivotal for the confusion (a, c), resulting in $\mathsf{pivot}(\pi_2) = \{p_5\}$.

Definition 14 (Concurrent Transitions). *Independent transitions t and t' are* concurrent *if neither (t, t') nor (t', t) is a confusion. Let $C \subseteq T \times T$ be the symmetric and irreflexive concurrency relation.*

[2] That is, $t_2 \in en(s)$ and $t_2 \notin en(s')$, or $t_2 \notin en(s)$ and $t_2 \in en(s')$, where $s[t_3\rangle s'$.

The notion of concurrent transitions is pessimistic: two transitions t and t' may be in confusion, yet t' can execute independently of t without affecting its conflicts.

Definition 15 (Weak Places). *For CPN* $\mathcal{C} = (N, \Pi)$ *and* $\pi \in \Pi$*, let* $\mathsf{weak}(\pi) = \mathsf{ngb}(\pi) \cap \bigcup_{\pi' \in \Pi \setminus \{\pi\}} \mathsf{ngb}(\pi')$.

Thus, the places in $\mathsf{weak}(\pi)$ are in the part of the neighborhood of an agent π that can be changed by transitions fired by agents other than π. For example, in Figure 1, $\mathsf{weak}(\pi_1) = \{p_1, p_2, p_4\}$, and in Figure 5, $\mathsf{weak}(\pi_1) = \{p_5\}$.

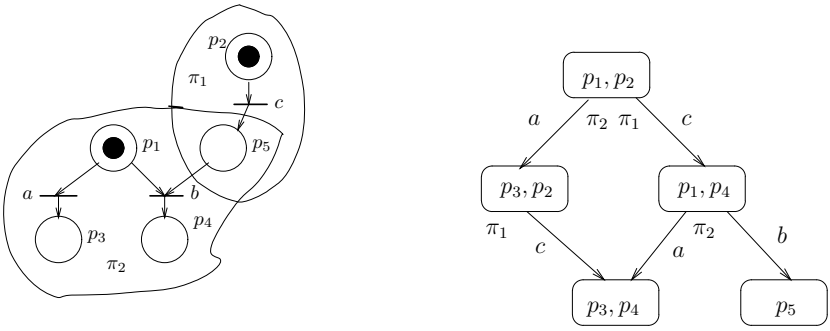

Fig. 5. A Petri net with confusion and its state graph

4 Probabilistic Covered Petri Nets

In this section, we extend the notion of covered Petri nets with branching probabilities. This naturally gives rise to a Petri net model that can be used to describe Markov decision processes [21] at a high level of abstraction. The nondeterministic choices in the MDPs correspond to the selection of (enabled) agents whereas the agents are responsible for resolving the probabilistic choices (based on their local view).

Probabilistic CPNs. In the sequel, for countable set T, let $\mathit{Dist}(T)$ be the set of probability distributions over T, and $\mathit{Dist}_\perp(T)$ be the set of distribution functions that for some elements in T may be undefined, i.e., yield the value $\perp$. Functions $\mu \in \mathit{Dist}_\perp(T)$ thus are of type $T \mapsto [0,1] \cup \{\perp\}$ and satisfy $\sum_{t \in T, \mu(t) \neq \perp} \mu(t) = 1$.

Definition 16 (Probabilistic CPN). *A* probabilistic CPN $\mathcal{D} = (N, \Pi, f)$ *is a CPN* (N, Π) *equipped with a function* $f : \Pi \times S \to \mathit{Dist}_\perp(T)$ *satisfying for all* $\pi \in \Pi$ *and* $s \in S$*:*

1. $f(\pi, s)(t) = \perp$ *iff* $t \notin \mathit{en}(s) \cap \pi$.
2. *For each* $s' \in S$, $f(\pi, s) = f(\pi, s')$ *whenever* $s\lceil_\pi = s'\lceil_\pi$.

Intuitively speaking, the function f assigns to a pair (π, s) a probability distribution over the enabled transitions in state s (of N) that are "visible" by the agent π. The first clause asserts that f is undefined only for transitions that are disabled for agent π. The second clause requires that an agent π, whose local views in states s and s' coincide, chooses a given transition in these states with equal probability. That is to say, the probability distribution over the enabled transitions only depends on the local information of the agent.

Example 2. Consider the CPN in Figure 1(right) and agent π_1 with $\mathsf{ngb}(\pi_1) = P \setminus \{p_5\}$ and let state $s = \{p_1, p_2\}$. Assume π_1 has a fair coin, yielding $f(\pi_1, s)(a) = f(\pi_1, s)(b) = \frac{1}{2}$. The same distribution for π_1 applies to the state $s' = \{p_1, p_2, p_5\}$, since $s\lceil_{\pi_1} = s'\lceil_{\pi_1}$. Now, consider agent π_2 with $\mathsf{ngb}(\pi_2) = P \setminus \{p_3\}$. Agent π_2 may select an enabled transition in s by flipping a biased coin, say, $f(\pi_2, s)(b) = \frac{1}{3}$ and $f(\pi_2, s)(c) = \frac{2}{3}$. Note that the transition b is common to the two agents, but its firing probabilities may differ, depending on which agent selects b. If π_1 is selected first to resolve the choice between a and b, and it selects to fire a, subsequently, we obtain the state $s'' = \{p_2, p_3\}$ and π_2 has a new local information view, namely $\{p_2\}$. Thus, it now only has the possibility to choose c, yielding $f(\pi_2, s'')(c) = 1$.

Example 3. Figure 3 represents a simple randomized mutual exclusion algorithm [4] where access to the critical section is arranged by an arbiter. In the initial state, agent π_1 can decide to fire transition a. By symmetry, agent π_2 can do the same for b. However, to fire transition c or d—acquiring access to the critical section—we use a third agent π_3 that acts as arbiter. If only c (or only d) is enabled, then π_3 decides to fire this transition. In case both c and d are enabled, i.e., in the state $\{p_3, p_4, p_9\}$, the agent π_3 flips a fair coin (say) yielding probability $\frac{1}{2}$ for transition c and d.

From probabilistic CPNs to MDPs. In the following, we show that probabilistic CPNs naturally give rise to MDPs (Markov Decision Processes [21]). In the sequel we also show that there is a one-to-one relationship between probabilistic CPN adversaries and (traditional) adversaries for MDPs. Let us start by recalling the notion of MDPs [21]. As we consider 1-safe Petri nets, it suffices to consider finite-state MDPs.

Definition 17 (Markov Decision Process). *A* Markov decision process (MDP) *is a tuple* $(Q, Act, \mathbb{P}, q_0)$ *where*

- Q *is a finite set of* states *with* initial state $q_0 \in Q$.
- Act *is a finite set of* actions.
- $\mathbb{P} : Q \times Act \times Q \mapsto [0, 1]$ *with for each* $q \in Q, \alpha \in Act$, $\sum_{q' \in Q} \mathbb{P}(q, \alpha, q') \in \{0, 1\}$.

An action α *is* enabled *in state* q *iff* $\mathbb{P}(q, \alpha, q') > 0$ *for some* $q' \in Q$.

The intuitive semantics of an MDP is as follows. In state q, one of its enabled actions is selected nondeterministically. As usual, we assume that for every state

this set is nonempty. After having selected action α, say, in state q, the next state is randomly determined. More precisely, the probability of moving to state q' (which may equal q) is $\mathbb{P}(q, \alpha, q')$. An MDP execution is thus an alternating sequence of states and actions $q_0\alpha_1 q_1\alpha_2 \ldots$ such that α_{i+1} is enabled in state q_i. Probabilistic CPNs can be viewed as a modeling formalism for MDPs in the following way.

Definition 18 (The MDP of a probabilistic CPN). *Let $\mathcal{D} = (N, \Pi, f)$ be a probabilistic CPN with $N = (P, T, E, s_0)$. The MDP of $\mathcal{D}$, denoted $\mathsf{mdp}(\mathcal{D})$, is the tuple $(S, Act, \mathbb{P}, s_0)$, where $S = 2^P$, $Act = \Pi$, and*

$$\mathbb{P}(s, \pi, s') = \sum\{f(\pi, s)(t) \mid t \in \pi, s[t\rangle s' \text{ and for no } \pi' \in \Pi, \pi' \gg_t \pi\}.$$

Stated in words, the states of $\mathsf{mdp}(\mathcal{D})$ are the states of the net N. Its actions are the agents. This corresponds to the intuition that an agent is selected nondeterministically, which resolves the probabilistic choice. The transition probabilities in $\mathsf{mdp}(\mathcal{D})$ correspond to the function f, provided the selected agent has the privilege to resolve the probabilistic choice. As several transitions in a given state of the net may result in the same target state, we take the sum over all individual probabilities of these transitions.

Adversaries. As the former of the previous examples showed, the probability of a transition occurrence may depend on the agent that has been selected. This suggests to define a probability measure over the behaviours of a probabilistic CPN that is subject to a given selection of agents. In order to do so, we resort to the standard notion of an adversary [21] (sometimes also called scheduler or strategy) and adapt this to our setting.

Definition 19 (MDP Adversary). *An adversary (strategy) for an MDP is a function A that maps execution fragments $q_0\, q_1 \ldots q_n$ of the MDP such that the action $A(q_0\, q_1 \ldots q_n)$ is enabled in q_n.*

An adversary thus selects an enabled action in the final state of a given execution fragment of the MDP.[3] The basic idea of an adversary for a probabilistic CPN is that it takes as argument a prefix of an execution and maps this onto an agent that can extend this prefix.

Definition 20 (Adversary for a probabilistic CPN). *An* adversary *A for a probabilistic CPN $\mathcal{C} = (N, \Pi, f)$ is a function that maps a prefix $\rho = s_0[\pi_1|t_1\rangle s_1 \ldots s_{n-1}[\pi_n|t_n\rangle s_n$ of an execution of $\mathcal{C}$ onto an agent $\pi_{n+1} \in \Pi$ such that $\rho[\pi_{n+1}|t_{n+1}\rangle s_{n+1}$ is a prefix of an execution of $\mathcal{C}$ for some $t_{n+1} \in \pi_{n+1}$.*

An adversary thus selects after a finite execution fragment of the covered net which agent is to resolve the next probabilistic choice. (The random choice,

[3] These are also called deterministic adversaries [21]. Our setting can easily be generalized to randomized adversaries that select agents according to a probability distribution. This falls however outside the scope of this paper.

i.e., the selection of transition t_{n+1} is done by the selected agent, not by the adversary.) An *A-execution* is an execution $s_0[\pi_1|t_1\rangle s_1[\pi_2|t_2\rangle s_2 \ldots$ of the probabilistic CPN such that for all $i \geq 0$, $\pi_{i+1} = A\,(s_0[\pi_1|t_1\rangle s_1 \ldots s_{i-1}[\pi_i|t_i\rangle s_i)$. That is to say, an A-execution is the execution fragment of the probabilistic CPN in which adversary A decides on every step which agent is to make a selection. A probability measure can now be defined on A-executions in the following way. The probability of the execution fragment s_0 is one, and the probability of $s_0[\pi_1|t_1\rangle s_1 \ldots s_{n-1}[\pi_n|t_n\rangle s_n$ is defined as the product $f(s_0, \pi_1)(t_1) \cdot \ldots \cdot f(s_{n-1}, \pi_n)(t_n)$. An alternative way of looking at this, is that an adversary A imposed on a probabilistic CPN yields an infinite Markov chain in which states correspond to finite execution fragments and transition probabilities are determined by the agent selected by adversary A in the current state.

Example 4. Consider the CPN of Figure 3 and let $s_0 = \{p_1, p_2, p_9\}$ with $A(s_0) = \pi_1$. As π_1 can only select transition a, this yields the execution fragment $\rho_1 = s_0[\pi_1|a\rangle s_1$ with $s_1 = s_0 \setminus \{p_1\} \cup \{p_3\}$. Let $A(\rho_1) = \pi_2$. As π_2 has a single choice, it selects transition b yielding $\rho_2 = s_0[\pi_1|a\rangle s_1[\pi_2|b\rangle s_2$ with $s_2 = \{p_3, p_4, p_9\}$. Now only π_3 has a choice between enabled transitions. If π_3 randomly selects c we obtain $\rho_3 = s_0[\pi_1|a\rangle s_1[\pi_2|b\rangle s_2[\pi_3|c\rangle s_3$. Assuming as before that π_3 flips a fair coin to resolve the choice between c and d, we obtain that the probability of execution fragment $\rho_3 = f(s_0, \pi_1)(a) \cdot f(s_1, \pi_2)(b) \cdot f(s_2, \pi_3)(c)$ which equals $1 \cdot 1 \cdot \frac{1}{2}$.

It is not difficult to see that an adversary of a probabilistic CPN corresponds directly to an adversary for its MDP. This immediately yields:

Lemma 2. *The Markov chain induced by adversary A on probabilistic CPN $\mathcal{D}$ is isomorphic to the Markov chain induced by A on the MDP* $\mathsf{mdp}(\mathcal{D})$.

A probabilistic CPN can thus be considered as a high-level (and possibly succinct) representation of an MDP. The MDP loses the structural information of the CPN, much as the state graph of a net. A measure over sets of infinite A-executions can be defined in the standard way using a cylinder set construction, see, e.g., [4, Ch. 10]. A measurable set of A-executions of a probabilistic CPN for a given adversary A is called an *event*. Based on the probability measure over A-executions one can now define the maximal, and dually the minimal, probability of certain events of interest. For instance, for set $G \subseteq S$ of states, let $\Diamond G$ denote the set of executions of a CPN that at some point reach some state in G. The set $\Diamond G$ is measurable (and thus an event). The maximal probability of $\Diamond G$ stands for the supremum over all possible agent selections (by any kind of adversary defined above) of eventually reaching G. In a similar way, the minimal probability is defined as the infimum over all possible agent selections to reach G. This can be generalized towards arbitrary LTL-formulas rather than simply reachability properties. Due to the above lemma, there is a direct relation between the occurrence probabilities in a probabilistic CPN to those in its MDP. Model-checking algorithms for MDPs [4, Ch. 10] can thus be exploited to calculate quantitative bounds such as the minimal and maximal probability of a reachability property $\Diamond G$, or of a temporal logic formula in LTL (or probabilistic CTL). The details of

these algorithms fall outside the scope of this paper; it suffices here that the key numerical component is solving a system of linear inequations, whereas for LTL model checking of MDPs the construction of an automaton on infinite words is an additional important ingredient.

5 A Distributed Scheduling Algorithm for Making Probabilistic Choices

We described in this paper a Petri net based model that allows concurrency, non deterministic choice (among agents) and probabilistic choices. In order to show how distributed scheduling of probabilistic choices, as required by our model, can be achieved, we provide now an algorithm for implementing systems based on CPNs. The algorithm concretizes the possible schedulers of Def. 20.

Our algorithm is by no means the only possible way of implementing a system described as a probabilistic CPN, or the most efficient one. As identified in [7], it is important to provide a temporarily stable view for an agent that makes a probabilistic decision; the choices that this agent has must not change *after* the agent has finished collecting the information about its choices and *before* a probabilistic choice is made. In component-based systems [6,20] there is a similar difficulty in synchronization algorithms that guarantee a selection of an interaction. However, here the problem is to stabilize the interaction in which agents participate, rather than selecting a single interaction. Thus the approach is *agent centric* instead of *interaction centric*.

Requirements. We impose the following requirements on the distributed scheduling algorithm.

Concurrency. The algorithm allows concurrent probabilistic choices. (The algorithm in [7] allows only a single agent to make a choice; this is established by passing a token among the agents.)

Semaphores. The scheduling is implemented using semaphores. Only standard lock and free operators of semaphores are allowed. If needed, semaphore operations can be imitated by message passing.

Efficiency. A simple analysis of the structure of the Petri net, i.e., the graph between transition and places, and the partitioning into agents, in time quadratic in the size of the net, is performed once. This establishes the interactions that will be needed at run time.

Fine granularity. Atomicity is not assumed at a coarse granularity. Realistically we cannot assume that the local information of an agent needs to be examined atomically. While gathering this information, some of the checked places may have gained or lost a token. Thus, several actions may be needed in setting up the conditions for the correct firing of a transition according to the semantics of the CPN.

Liveness. No deadlock is introduced. In fact, the algorithm does not limit the executions and admits exactly the set of executions of the CPN.

Finite memory. The scheduling decisions for agents are based only on the current state of the execution and the value of the semaphores.

Partial view. The scheduling is based on the local information of agents and not on the global states [8].

The scheduling algorithm. The idea is to assign a semaphore to certain places of the net. We will henceforth relate interchangeably in notation, when clear from context, semaphores and the places they are associated with. Prior to firing a transition, a phase of locking semaphores associated with a set of places is carried out. This set is precisely defined below. The capturing of semaphores provides a temporarily stable view of an agent regarding the (probabilistic) choices that it needs to make. When an agent π makes a probabilistic decision, it needs to stabilize some tokens of $\mathsf{weak}(\pi)$ (this subset is defined precisely below), as the value of these places can affect π's set of choices. For the scheduling algorithm, we use a set of semaphores related to the weak places:

$$\mathsf{sem}(\Pi) = \bigcup_{\pi \in \Pi} \mathsf{weak}(\pi).$$

It is of course important to minimize the number of semaphores an agent is required to lock. Capturing $\mathsf{weak}(\pi)$ before firing a transition by π would indeed guarantee a stable environment for a probabilistic choice, but may incur needless overhead and severely restricts the concurrency by locking semaphores that are not relevant in the current state. We therefore propose a smaller subset of semaphores needed to be locked by agent π in state s so as for π to make a (probabilistic) choice:

$$\mathsf{capture}(\pi, s) = \mathsf{weak}(\pi) \cap (\mathsf{ngb}(en(s) \cap \pi) \cup \mathsf{pivot}(en(s) \cap \pi)).$$

Thus, $\mathsf{capture}(\pi, s)$ includes two sets of places:

- $\mathsf{weak}(\pi) \cap \mathsf{ngb}(en(s) \cap \pi)$ are the places that π may change by firing the next transition and can affect other agents; also, changing these places by firing a transition by another agent would affect their enabledness for π, and
- $\mathsf{weak}(\pi) \cap \mathsf{pivot}(en(s) \cap \pi)$ are the places that other agents can change and may alter the choices available to π.

For our algorithm we assume the existence of a partial order, denoted $\prec$, on the set of semaphores $\mathsf{sem}(\Pi)$ such that no two semaphores that are in $\mathsf{weak}(\pi)$ for some $\pi \in \Pi$ are unordered. We do neither assume that an agent collects its local information or acquires all needed semaphores atomically, nor that it changes all places involved in firing a transition atomically. However, we assume the existence of a mechanism by which an agent π can set an interrupt that informs it if a place was changed after its value has been inspected. By requiring that changing the value of a (weak) place p is done only after p's semaphore is acquired, we can safely assume that an agent knows the correct value of p when it holds its semaphore. Our algorithm now proceeds in two phases:

Phase 1. Each agent checks which of its transitions are enabled. It is not necessary that this is done atomically; rather, a lookup through the places (represented by variables, message queues, etc.) is performed. An interrupt that reports a change in the value of a place that has been already checked in this phase, causes a restart of this phase.

Phase 2. Agent π locks the semaphores $\mathsf{capture}(\pi, s)$ in an ascending order according to the partial order $\prec$. If there is an interrupt announcing a change in the value of a place p that was checked in Phase 1 before p's semaphore is locked, all the semaphores locked by π so far are released in descending order (according to $\prec$), and Phase 1 is restarted.

If agent π has acquired all semaphores in $\mathsf{capture}(\pi, s)$, it randomly selects one of its enabled transition. It is important to note that $\mathsf{capture}(\pi, s) = \mathsf{capture}(\pi, s')$ when $s\lceil_\pi = s'\lceil_\pi$. That is, $\mathsf{capture}(\pi, s)$ depends only on the local information of π. This allows calculating $\mathsf{capture}(\pi, s)$ during the execution of the algorithm locally by π.

Example 5. Consider the net in Figure 5. Agent π_2 has one weak place: p_5. In the initial state $s_0 = \{p_1, p_2\}$, $\mathsf{weak}(\pi_2) \cap \mathsf{ngb}(en(s_0) \cap \pi_2) = \emptyset$ because the places $\mathsf{ngb}(en(s_0) \cap \pi_2) = \mathsf{ngb}(\{a\}) = \{p_1, p_3\}$ are not weak (as they belong only to π_2). However, as p_5 is pivotal to the confusion (a, c), $\mathsf{capture}(\pi_2, s_0)$ is in this case $\mathsf{weak}(\pi_2) \cap \mathsf{pivot}(en(s_0) \cap \pi_2) = \{p_5\}$. Thus, in order for π_2 to maintain a stable situation with respect to the choices it can make (in this case, just firing a), π_2 must lock the semaphore for place p_5. Now agent π_1 cannot fire transition c: in order to do that from s_0, it needs to lock $\mathsf{capture}(\pi_1, s_0)$, which is, in this case, $\mathsf{weak}(\pi_1) \cap \mathsf{ngb}(en(s_0) \cap \pi_1) = \{p_5\}$. Note that π_2 needs to lock p_5 because it is pivotal to a confusion with transition a, whereas π_1 needs to lock p_5 because it may want to change it by firing c. The set of semaphores that an agent needs to lock is dependent on its local information; when p_1 does not have a token, agent π_2 does not need to lock the semaphore for p_5.

Lemma 3. *An agent π cannot change a place by firing a transition without capturing the corresponding semaphore for that place.*

Proof. Follows immediately from the need for an agent to lock, before firing the next transition, the semaphores for $\mathsf{capture}(\pi, s)$. This set includes $\mathsf{weak}(\pi) \cap \mathsf{ngb}(en(s) \cap \pi)$. □

While no change occurs to the semaphores in $\mathsf{capture}(\pi, s)$, agent π has a stable set of choices, as proved by the following result.

Lemma 4. *A change to the set of currently enabled transitions of an agent π in a state s, i.e., $en(s) \cap \pi$, by firing a transition t by another agent π' (possibly $t \in \pi \cap \pi'$) in state s involves a change to some place in $\mathsf{capture}(\pi, s)$.*

Proof. Distinguish two cases.

1. t depends on some transition $t' \in en(s) \cap \pi$. By Def. 6, a state-change by t implies a change of a common place with $\mathsf{ngb}(t') \subseteq \mathsf{ngb}(en(s) \cap \pi)$. As t is executed by agent $\pi' \neq \pi$, this place also belongs to $\mathsf{weak}(\pi)$.

2. t is independent of all the transitions in $en(s) \cap \pi$. As (by assumption) t changes the enabled transitions of π in $en(s) \cap \pi$ while being independent of (all of) them, firing t enables some new transition of π, which, by the definition of agents, depends on some already enabled transition of π. Thus, by Def. 12, (t', t) is a confusion for some $t' \in en(s) \cap \pi$. By Def. 13, some place $p \in \mathsf{pivot}(en(s) \cap \pi)$ is altered by t to cause the enabledness of a transition of π in conflict with t'. As $t \notin \pi$, p is in $\mathsf{weak}(\pi)$. □

Note how the need to lock a semaphore associated with a pivotal place due to a confusion can reduce concurrency between independent transitions. This is in accordance with Definition 14.

Lemma 5. *The scheduling algorithm does not introduce a deadlock.*

Proof. Capturing semaphores in ascending order and releasing them in descending order is a solution for a generalized mutual exclusion problem, suggested originally by Dijkstra for the dining philosophers problem. See [24] for its correctness, including deadlock freeness. Note that if Phase 1 restarts, some progress must have occurred, as a place was changed by firing some transition. □

Lemma 6. *The scheduling algorithm admits exactly the set of CPN executions.*

Proof. Clearly, any execution of the net under the obtained scheduler must conform to the semantics of execution of the Petri nets. Conversely, for each execution of the CPN, we have a behavior of the scheduling algorithm in which the two phases related to the firing of each transition are clearly separated, where the capturing of the semaphores and the firing of a transition do not interleave (although the scheduling algorithm also allows interleaving of semaphore capturing, checking places and firing transitions in other ways). □

The above algorithm determines a possible schedule of the agents to fire enabled transitions: any agent that has acquired the necessary semaphores can randomly choose one of its enabled transitions. Note that it is possible that several agents are in a position to carry out a random selection, in case they all acquired their semaphores. In this case, an agent can be picked nondeterministically. That is to say, the algorithm determines a possible adversary (scheduler) A for the probabilistic CPN at hand. The following result asserts that the computed schedule yields indeed probabilities (for LTL formulas) that fall inside the scope of the minimal and maximal probabilities that are typically determined by model-checking algorithms for MDPs.

Theorem 1. *For probabilistic CPN $\mathcal{D}$ and LTL formula φ, the probability of satisfying φ for any obtained schedule by our tho-phase algorithm is within the probability bounds of the MDP $\mathsf{mdp}(\mathcal{D})$ satisfying φ.*

Proof. Based on Lemmas 3, 4 and 5, the obtained adversaries for $\mathcal{D}$ by our algorithm correspond to a subset of the adversaries of the MDP $\mathsf{mdp}(\mathcal{D})$. □

For various events of interest, such as the earlier mentioned reachability events of the form $\Diamond G$, the minimal and maximal probabilities are attained by a simple class of adversaries, the so-called *memoryless* adversaries. An adversary A

of a probabilistic CPN is memoryless whenever its decision for execution fragment $s_0[\pi_1|t_1\rangle \ldots [\pi_n|t_n\rangle s_n$ only depends on s_n. That is, a memoryless adversary selects for every visit to state s_n the same action regardless of the execution fragment before reaching s_n.

Theorem 2. *Our scheduling algorithm admits any memoryless adversary.*

Proof (sketch). An agent can only get its turn whenever it has acquired all semaphores. Acquiring the semaphores is based on the local information of an agent in a given state of the net. The decision whether an agent can perform a transition or not is memoryless. In case several agents can perform a transition (i.e., they have all acquired their necessary semaphores), the order between these agents is to be determined by the adversary. Our algorithm does not restrict this ordering: any memoryless order of admissible agents is allowed. A selected enabled agent then performs a (local) probabilistic choice. □

6 Related Work

The most well known extensions of Petri nets with randomness are (generalized) stochastic Petri nets (GSPNs) [18]. There, transitions are equipped with rates, i.e., parameters of exponential distributions. In stochastic Petri nets (SPNs) all transitions have a rate—concurrency becomes a random phenomenon and confusion is absent. Due to immediate transitions in GSPNs, confusion re-appears. This is partially tackled using weights (resolving choices probabilistically based on a global state), but the analysis of GSPNs is basically restricted to confusion-free nets. Recently, a semantics of GSPNs with confusion has been proposed using stochastic real-time games [9,12].

The few works on probabilistic Petri nets treat probabilistic branching quite differently. In [2], probabilities are attached to outgoing edges of places. Alternatively, weights are assigned to edges [25]; here, choices are resolved in a probabilistic way whereas independent transitions fire in any order. A relation is shown between confusion-free weighted nets and Mazurkiewicz equivalence. Kudlek [15] focuses, instead, on the expressive power of the formalism, whereas [1] proposes truly concurrent probabilistic Petri nets. Here, the likelihood of processes is defined on partial orders, not on firing sequences. An MDP interpretation to nets is given in [5]. There, an extended Petri net model includes explicit transitions that indicate where a nondeterministic choice and where a probabilistic choice starts. Processes subscribe to such a choice, and the choice is made globally. To our knowledge, the treatment of probabilities in nets using the concept of agents—resolving probabilistic choices locally—is new.

The fact that global schedulers establish strong correlations between the behavior of system components (i.e., agents) has been observed earlier in [7,17]. To get around this problem [7] proposes *switched* probabilistic I/O automata. By passing a token between agents, one of the agents may make a probabilistic decision. This token-based scheme however restricts concurrency. In our model, concurrent nondeterministic and probabilistic decisions are possible. Concurrency is

restricted only by confusions, which correspond to potential changes to the available choices. Also in testing theory and security analysis, it has been recognized that the resolution of local choices within a component using global knowledge yields undesirable and counterintuitive behavior. Our two-level scheduling mechanism in which agents are selected based on global state information, whereas agents select based on their local perspective, is closely related to that of *distributed* schedulers [11]. In contrast to our case, the selection of components there can be probabilistic. (As mentioned earlier, our framework can be easily extended to random agent selection.) Agent scheduling is also a principle used in the setting of quantitative security [3]. We are unaware of any concrete distributed algorithms realizing this kind of scheduling. In our case, we complement the theoretical setting with such algorithm.

7 Epilogue

In this paper, we enhanced Petri nets with agents covering the net transitions. The local view of an agent in a covered net consists of the neighborhood (input and output places) of its transitions. In a step of a net execution, an agent is nondeterministically selected which—based on its local view—resolves a (probabilistic) decision. This provides an elegant and robust basis for resolving probabilistic choices in a nondeterministic setting. It is shown that probabilistic covered nets can be viewed as high-level descriptions of MDPs. Finally, we presented a distributed scheduling algorithm (based on semaphores) for implementing such nets. Our algorithm is obtained by a simple structural analysis of net. Confusions, in our view, are no longer an obstacle for implementing Petri nets. Rather, they are an artifact reducing concurrency, similar to the notion of dependency in trace theory [19]. In fact, the identification and analysis of confusions provides a basis for our algorithm.

We used in this paper Petri nets to demonstrate the main concepts involved in modeling and implementing distributed probabilistic scheduling. In particular, confusion was originally observed in Petri nets and has a simple and clean formal presentation within this model. Nevertheless, our modeling concepts and scheduling algorithm can be easily adapted to other models that include concurrency and probabilistic choice.

Acknowledgments. The authors thank Barbara Jobstmann and Gadi Taubenfeld for valuable discussions.

References

1. Abbes, S.: The (True) Concurrent Markov Property and Some Applications to Markov Nets. In: Ciardo, G., Darondeau, P. (eds.) ICATPN 2005. LNCS, vol. 3536, pp. 70–89. Springer, Heidelberg (2005)
2. Albanese, M.: A constrained probabilistic Petri net framework for human activity detection in video. IEEE Trans. on Multimedia 10(6), 982–996 (2008)

3. Andrés, M.E., Palamidessi, C., van Rossum, P., Sokolova, A.: Information hiding in probabilistic concurrent systems. TCS 412(28), 3072–3089 (2011)
4. Baier, C., Katoen, J.-P.: Principles of Model Checking. MIT Press (2008)
5. Beccuti, M., Franceschinis, G., Haddad, S.: Markov Decision Petri Net and Markov Decision Well-Formed Net Formalisms. In: Kleijn, J., Yakovlev, A. (eds.) ICATPN 2007. LNCS, vol. 4546, pp. 43–62. Springer, Heidelberg (2007)
6. Bliudze, S., Sifakis, J.: The algebra of connectors - structuring interaction in BIP. IEEE Trans. Computers 57(10), 1315–1330 (2008)
7. Cheung, L., Lynch, N.A., Segala, R., Vaandrager, F.W.: Switched PIOA: Parallel composition via distributed scheduling. TCS 365(1-2), 83–108 (2006)
8. de Alfaro, L.: The verification of probabilistic systems under memoryless partial-information policies is hard. In: PROBMIV, pp. 19–32 (1999)
9. Eisentraut, C., Hermanns, H., Zhang, L.: Concurrency and Composition in a Stochastic World. In: Gastin, P., Laroussinie, F. (eds.) CONCUR 2010. LNCS, vol. 6269, pp. 21–39. Springer, Heidelberg (2010)
10. Georgievska, S., Andova, S.: Probabilistic may/must testing: retaining probabilities by restricted schedulers. Formal Asp. Comput. 24(4-6), 727–748 (2012)
11. Giro, S., D'Argenio, P.R.: On the expressive power of schedulers in distributed probabilistic systems. ENTCS 253(3), 45–71 (2009)
12. Katoen, J.-P.: GSPNs revisited: Simple semantics and new analysis algorithms. In: Application of Concurrency to System Design, pp. 6–11 (2012)
13. Katz, G., Peled, D.: Code Mutation in Verification and Automatic Code Correction. In: Esparza, J., Majumdar, R. (eds.) TACAS 2010. LNCS, vol. 6015, pp. 435–450. Springer, Heidelberg (2010)
14. Katz, G., Peled, D., Schewe, S.: Synthesis of Distributed Control through Knowledge Accumulation. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 510–525. Springer, Heidelberg (2011)
15. Kudlek, M.: Probability in Petri nets. Fund. Inf. 67(1-3), 121–130 (2005)
16. Lehmann, D.J., Rabin, M.O.: On the advantages of free choice: A symmetric and fully distributed solution to the dining philosophers problem. In: POPL, pp. 133–138 (1981)
17. Lynch, N.A., Segala, R., Vaandrager, F.W.: Observing branching structure through probabilistic contexts. SIAM J. Comp. 37(4), 977–1013 (2007)
18. Ajmone Marsan, M., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets. Wiley (1995)
19. Mazurkiewicz, A.: Introduction to trace theory. In: Diekert, V., Rozenberg, G. (eds.) The Book of Traces. World Scientific (1995)
20. Pérez, J.A., Corchuelo, R., Toro, M.: An order-based algorithm for multiparty synchronization. Concurrency - Practice and Experience 16(12), 1173–1206 (2004)
21. Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley (2005)
22. Rozenberg, G., Thiagarajan, P.S.: Petri Nets: Basic Notions, Structure, Behaviour. In: Rozenberg, G., de Bakker, J.W., de Roever, W.-P. (eds.) Current Trends in Concurrency. LNCS, vol. 224, pp. 585–668. Springer, Heidelberg (1986)
23. Segala, R., Lynch, N.A.: Probabilistic simulations for probabilistic processes. Nord. J. Comput. 2(2), 250–273 (1995)
24. Taubenfeld, G.: Synchronization Algorithms for Concurrent Programming. Prentice Hall (2006)
25. Varacca, D., Nielsen, M.: Probabilistic Petri nets and Mazurkiewicz equivalence (2003) (unpublished manuscript)