Stochastic Petri Net Semantics for Stochastic Process Algebras

Marina Ribaudo
Dipartimento di Informatica
Università degli Studi di Torino
Torino, Italy

Abstract

In this paper we define a Stochastic Petri Net (SPN) semantics for Stochastic Process Algebras (SPAs), a recently introduced formalism that offers a novel approach for performance modeling. The proposed semantics is evaluated in terms of three criteria: Concurrency and Retrievability, as defined by Olderog for untimed net semantics, and Markov Equivalence for the stochastic aspects.

1 Introduction

SPNs and SPAs have had somewhat of a common development. In both cases the original definitions (standard Petri nets and pure Process Algebra) did not include any temporal information, so that they were used only for the qualitative analysis of concurrent systems.

The timing extensions of the basic formalisms also allowed one to study the quantitative properties of systems. In particular, in both SPNs and SPAs, negative exponentially distributed delays have been associated with the elementary events (transitions and actions), and it has been proved that both formalisms can be viewed as high level formalisms for the specification of Markov processes.

Similarities and differences can be established between the two formalisms and we think that the definition of a mapping between them may constitute a starting point for investigating the existing relationship. In this paper we thus show how to map the basic operators of SPAs into SPN models. This work extends a previous paper [21] in which the semantics was given by means of small examples.

The balance of this paper is as follows. In Section 2 we define the basic concepts concerning SPAs. Being a well known formalism SPNs are not described in detail; information on SPNs may be found, for example, in [14, 15]. The SPN semantics for SPAs is defined in Section 3 and some properties that the semantics must satisfy are described in Section 4. Section 5 provides a brief comparison among the different SPA languages introduced in the literature. Finally, Section 6 concludes this work.

2 Stochastic process algebras

Process Algebras (PA) are abstract languages for the specification and understanding of concurrent systems. Actions are the building blocks of the language and they are used to describe entities that run concurrently and cooperate through communication. The models are built following a compositional approach. Systems are viewed as collections of smaller subsystems and each subsystem is modeled in isolation. Submodels are then composed using the language constructors to obtain a model that reflects the structure of the system itself.

The Calculus of Communicating Systems (CCS) [13] and the Communicating Sequential Processes (CSP) [10] can be considered as the starting point in this field. CCS and CSP cannot express time delays between events: action executions take zero time and only relative ordering is represented via the traces of the processes.

Over the years, various proposals for introducing time into PA have been developed. These attempts follow two main lines: time can be either deterministic or stochastic. In the deterministic approach the possibility of finding a calculus for real time systems in the style of CCS has been investigated (for an overview, see the paper by Nicollin and Sifakis [16]). The main purpose of the stochastic approach is that of combining system specification and performance evaluation, starting from the first steps of the design.

The languages that follow the stochastic approach are usually known as Stochastic Process Algebras (SPA) or Markovian Process Algebras [8, 7, 3, 2]. They all start from an untimed PA, extend the basic actions with exponential delays, and generate the underlying Markov process to compute performance measures on the modeled system.

In the remaining part of this section we will consider Performance Evaluation Process Algebra (PEPA) [8]

as our reference language, but most of the considerations we will discuss can also be easily extended to the other SPAs.

Systems are described in PEPA as interactions of components that can perform a set of activities. An activity $a \in \mathcal{A}ct$ is described as a pair (α, r), where $\alpha \in \mathcal{A}$ is the type of the activity and $r \in I\!R^+$ is the parameter of the negative exponential distribution governing its duration. Whenever a process P can perform an action, an instance of a given probability distribution is sampled: the resulting number specifies how long it will take to complete the action.

The syntax for PEPA terms is defined as follows

$$P ::= (\alpha, r).P \mid P + Q \mid P \underset{L}{\bowtie} Q \mid P/L \mid A$$

The intuitive meaning of the operators is the following.

$(\alpha, r).P$ is the process that performs an action of type α after a certain delay, and then behaves like P.

The component $P + Q$ enables activities of P and Q. A race condition governs the dynamic behavior of a model whenever more than one activity is enabled. This means that we may think of all the activities attempting to proceed, but only the fastest succeeding. The first activity to complete determines the future behavior of $P + Q$.

The component $P \underset{L}{\bowtie} Q$ represents a system where processes P and Q work together to perform some activities. In PEPA the cooperation between components follows the CSP style of communication [10]. The set L is called the cooperation set and defines the action types on which the components must synchronize or cooperate. P and Q proceed independently with any activities whose types do not occur in the cooperation set L. However, activities with action types in the set L are assumed to require the simultaneous involvement of both components. The resulting activity (called a shared activity) will have the same action type as the two contributing activities and a rate reflecting the rate of the synchronization. Several possibilities exist for defining this resulting rate in terms of the rates of the contributing activities (cf. Sec. 5). In PEPA, the rate of the synchronization is computed using the formula

$$R = \frac{r_1}{r_\alpha(P)} \frac{r_2}{r_\alpha(Q)} min(r_\alpha(P), r_\alpha(Q))$$

$r_\alpha(P)$ is called the apparent rate of action type α in the P component and it is the rate at which α appears to an external observer. For any component P the total capacity for carrying out activities of a given type is equal to the sum of the rates of the activities of that type which are enabled. Moreover, it is assumed that when two components carry out α in cooperation, their total capacity to complete the activity is limited to the capacity of the slowest. The apparent rate of a shared activity is then the minimum of the apparent rates in the two contributing components, i.e. $min(r_\alpha(P), r_\alpha(Q))$. Since, in general, each component may enable several α activities, we assume that each independently chooses which instance takes part in the cooperation. Thus the rate of the shared activity is adjusted to reflect these choices and it is multiplied by the product of the probabilities that each contributing activity is selected, i.e. $\frac{r_1}{r_\alpha(P)} \frac{r_2}{r_\alpha(Q)}$.

When the set L is empty, $\underset{L}{\bowtie}$ has the effect of parallel composition, allowing components to proceed concurrently without any interaction between them. In this case the concise notation $P \| Q$ is usually used instead of $P \underset{\phi}{\bowtie} Q$.

P/L is a process that cannot perform any actions belonging to the set L. These actions appear as the invisible action τ and they can be regarded as internal delays.

It is assumed that there is also a countable set of constants. Constants are components whose meaning is given by defining an equation such as $A \stackrel{\text{def}}{=} P$ that gives the constant A the behavior of P.

The formal semantics for the operators is given following the structural operational semantics style of Plotkin [20]. Applying the semantic rules, a transition diagram, called Derivation Graph (DG), may be associated with each PEPA term. The DG can be viewed as an alternative way for describing the behavior of the system [8]: the nodes are labeled by language terms, and the arcs are labeled by pairs (α, r) that cause the change of state.

The Markov process underlying any finite PEPA component can be obtained directly from the DG: a state of the Markov process is associated with each node of the graph and the transitions between states are defined by the arcs of the graph. Since all activity durations are exponentially distributed, the total transition rate between two states will be the sum of the activity rates labeling arcs connecting the corresponding nodes in the DG. Notice that some information is lost when we derive the underlying Markov chain. First, we no longer take into account action types, and second, multiple arcs between the same states in the DG are collapsed into a single one in the Markov chain.

To ensure that the underlying Markov process is ergodic, the DG of a PEPA model must be strongly connected. Some necessary conditions for ergodicity, at the syntactic level of a PEPA model, have been defined

in [8]. For example, cooperation must be the highest level combinator. The class of PEPA terms that satisfy these syntactic conditions are termed cyclic components and can be described by the following syntax

$$\begin{array}{lcl} P & ::= & S \mid P \underset{L}{\bowtie} P \mid P/L \\ S & ::= & (\alpha, r).S \mid S + S \mid A \end{array}$$

3 SPN semantics for PEPA

A semantics is a mapping that assigns to every element of a syntactic domain a meaning or interpretation, i.e. an element of a semantic domain.

Usually the semantics of PA is given by means of structural deduction rules. The application of such rules allows us to associate with every term P a transition diagram (Labeled Transition System) representing its meaning. This approach is known as an interleaving approach and it is based on a single observer model in which the occurrence of events is serialized even for independent processes.

A disadvantage of the interleaving approach is that causality information may be lost. Let us consider, for example, the two agents $A = a|b$ and $B = a.b + b.a$ (expressed using Milner's notation). Under the interleaving assumption the meaning (external behavior) of A and B is the same: an external observer will either see first an a and then a b, or first a b and then an a, and therefore will not be able to distinguish between them. The observer is not able to detect that in $a|b$ the actions a and b occur independently, while in $a.b + b.a$ there is a causal link between them: a occurs before b or b occurs before a.

Another approach to semantics is known as the true concurrency approach. In this context the two agents A and B are no longer indistinguishable because a distinction occurs between A, in which a and b can occur simultaneously, and B in which this option is not present.

Those in favor of the true concurrency approach think that the interleaving model is unrealistic and they present alternative semantic models based on Petri nets [4, 24, 17, 6, 5, 18, 23], event structures [25], or traces [10, 12].

We are interested in the work on the definition of a (untimed) Petri nets semantics for PA in which it is possible to associate a Petri net representation ($\mathcal{N}[\![P]\!]$) with each process term P. Several authors [6, 5, 24] have proposed a compositional Petri net semantics. Compositional refers to the way the semantics is defined and means that for every n-ary syntactic operator op on process terms a corresponding Petri net operator $op_{\mathcal{N}}$ is introduced such that

$$\mathcal{N}[\![op(P_1, P_2, \ldots, P_n)]\!] = op_{\mathcal{N}}(\mathcal{N}[\![P_1]\!], \mathcal{N}[\![P_2]\!], \ldots, \mathcal{N}[\![P_n]\!])$$

This says that the meaning (the net representation) of the composed term $op(P_1, P_2, \ldots, P_n)$ is inferred starting from the meanings of the component terms.

We now describe an SPN semantics [21] for the PEPA language following this compositional approach: for each operator op in the language we show the operation $op_{\mathcal{N}}$ that has to be performed at the net level to reproduce its effect. The operation $op_{\mathcal{N}}$ is based on the well known operations of transition and place superpositions.

The semantics of each operator is given first by means of one simple example and then it is formally defined for the general case. We consider labeled SPN models defined as 6-tuples $(P, T, F, \Lambda, M_0, l)$ where P is the set of places, T is the set of transitions, F is the flow relation defined as $F \subseteq (P \times T) \cup (T \times P)$, $\Lambda : T \to I\!R^+$ is the function defining the transitions rates[1], M_0 is the set of initial places, and l is a labeling function whose meaning will be clear while reading the next paragraph.

Basic element The basic element in PEPA is the activity $a = (\alpha, r)$ composed of two pieces of information: α is the action type and r is the rate of the exponential distribution of the associated delay. Its translation is shown in Fig. 1: p_1 and p_2 are the input and the output places of the basic element and the transition t models action α.

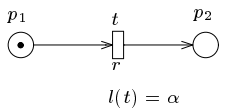

Figure 1: Translation of the basic element.

In this translation context, we need to emphasize the difference between transitions and actions: in PEPA different terms can perform actions of the same type and so we can have in the corresponding net several transitions representing the same action type. On the other hand, by definition, transitions need to be distinguishable, i.e. they must have different names. Therefore, we define a labeling function $l : T \to \mathcal{A}ct$ that maps transition names into action names[2]. Note that it is possible that $l(t_i) = l(t_j) = \alpha$ but $t_i \neq t_j$.

We can formally define the net representation of the basic element as follows.

Definition 1 Basic element
The net representation of the basic element (α, r) is

[1] In the formal definitions we will use the vector notation $\{(t_1, \lambda_1), (t_2, \lambda_2), \ldots, (t_m, \lambda_m)\}$.

[2] In the next figures we associate with each transition t the corresponding action name $l(t)$.

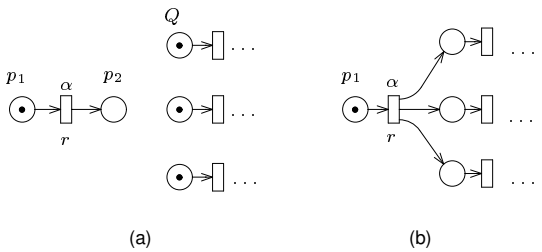

Figure 2: Translation of the prefixing operator.

$\mathcal{N} = (P, T, F, \Lambda, M_0, l)$ where

$$\begin{array}{lcl} P & = & \{p_1, p_2\} \\ T & = & \{t\} \\ F & = & \{(p_1, t), (t, p_2)\} \\ \Lambda & = & \{(t, r)\} \\ M_0 & = & \{p_1\} \\ l(t) & = & \alpha \end{array}$$

Prefixing Prefixing is the basic mechanism to build expressions in the language. As an example let us consider the term $P \stackrel{\text{def}}{=} (\alpha, r).Q$ whose translation can be formalized as

$$\mathcal{N}[\![(\alpha, r).Q]\!] = \cdot_{\mathcal{N}} (\mathcal{N}[\![(\alpha, r)]\!], \mathcal{N}[\![Q]\!])$$

First (see Fig. 2(a)) we translate the components inside round brackets, i.e. the action (α, r) and the term Q, and then we translate the "$\cdot$" operator superposing place p_2 with all the places that are initially marked in the net modeling Q (see Fig. 2(b)).

Definition 2 Prefixing
Let $\mathcal{N}_1$=$(\{p_1, p_2\}$,$\{t\}$,$\{(p_1, t), (t, p_2)\}$,$\{(t, r)\}$, $\{p_1\}, l_1)$ be the net representing (α, r) and $\mathcal{N}_2 = (P_2, T_2, F_2, \Lambda_2, M_{0,2}, l_2)$ be the net representing Q, with $\{p_1, p_2\} \cap P_2 = \{t\} \cap T_2 = \emptyset$. The net modeling $P = (\alpha, r).Q$ is $\mathcal{N}_{(.)} = (P, T, F, \Lambda, M_0, l)$ where

$$\begin{array}{lcl} P & = & \{p_1\} \cup P_2 \\ T & = & \{t\} \cup T_2 \\ F & = & F_2 \cup \{(p_1, t)\} \cup \{(t, q) : q \in M_{0,2}\} \\ \Lambda & = & \{(t, r)\} \cup \Lambda_2 \\ M_0 & = & \{p_1\} \\ l(t) & = & \begin{cases} l_1(t) & if \;\; t \notin T_2 \\ l_2(t) & if \;\; t \in T_2 \end{cases} \end{array}$$

Cooperation The translation of the term $P \underset{L}{\bowtie} Q$ is

$$\mathcal{N}[\![P \underset{L}{\bowtie} Q]\!] = \underset{L}{\bowtie}_{\mathcal{N}} (\mathcal{N}[\![P]\!], \mathcal{N}[\![Q]\!])$$

We first map into SPN models the two components P and Q and then we define a proper scheme to translate the $\underset{L}{\bowtie}$ operator. For the cooperation we distinguish two cases:

1. for action types $\alpha \notin L$ we consider the union of the nets representing P and Q;

2. for action types $\alpha \in L$ we need to superpose transitions with the same label α and to compute the synchronization rates.

When the set L is empty, we have the parallel composition of independent components (denoted using the concise notation $\|$) that is translated considering the union of the starting nets.

The net depicted in Fig. 3(a) shows the translation of $P \stackrel{\text{def}}{=} (\alpha, r_1).P_1 \underset{\{\beta\}}{\bowtie} (\alpha, r_2).Q_1$ where both processes execute their actions independently.

A different situation arises in the case of the term $P \stackrel{\text{def}}{=} (\alpha, r_1).P_1 \underset{\{\alpha\}}{\bowtie} (\alpha, r_2).Q_1$ because both processes execute an activity of type α that belongs to the cooperation set. The cooperation between the two processes is obtained by merging the two α transitions, representing the two different views of the same activity, into a single one. The resulting transition models the shared activity as shown in Fig. 3(b).

For the computation of the rate R to be associated with the shared activity we need to apply the formula described in Sec. 2. Since in this simple example each component enables a single activity of type α, the rate R is equal to $\frac{r_1}{r_\alpha(P)} \frac{r_2}{r_\alpha(Q)} min(r_\alpha(P), r_\alpha(Q)) = \frac{r_1}{r_1} \frac{r_2}{r_2} min(r_1, r_2) = min(r_1, r_2)$.

The operation that has to be performed at the net level to reproduce the PEPA cooperation and to compute the synchronization rates can be formally expressed as follows.

Definition 3 Cooperation
Given $\mathcal{N}_1 = (P_1, T_1, F_1, \Lambda_1, M_{0,1}, l_1)$ and $\mathcal{N}_2 = (P_2, T_2, F_2, \Lambda_2, M_{0,2}, l_2)$ with $P_1 \cap P_2 = T_1 \cap T_2 = \emptyset$ modeling the components P and Q, we define the net modeling the component $R = P \underset{L}{\bowtie} Q$ as $\mathcal{N}_{(\underset{L}{\bowtie})} =$

$(P, T, F, \Lambda, M_0, l)$ where, for each $\alpha \in L$

$$
\begin{array}{lcl}
P & = & P_1 \cup P_2 \\
T & = & (T_1 - T_1^L) \cup (T_2 - T_2^L) \cup \\
 & & \{t_i \times t_j : t_i \in T_1 \wedge t_j \in T_2 \wedge l_1(t_i) = l_2(t_j) = \alpha \in L\} \\
 & & T_1^L = \bigcup_{\alpha \in L} \{t_i \in T_1 : l_1(t_i) = \alpha \in L\} \\
 & & T_2^L = \bigcup_{\alpha \in L} \{t_j \in T_2 : l_2(t_j) = \alpha \in L\} \\
F & = & (F_1 - \{(p,t) \in F_1 : l_1(t) = \alpha \in L\} \\
 & & \quad -\{(t,p) \in F_1 : l_1(t) = \alpha \in L\}) \cup \\
 & & (F_2 - \{(q,t) \in F_2 : l_2(t) = \alpha \in L\} \\
 & & \quad -\{(t,q) \in F_2 : l_2(t) = \alpha \in L\}) \cup \\
 & & \{(p, t_i \times t_j) : l_1(t_i) = l_2(t_j) = \alpha \in L \wedge \\
 & & \qquad ((p,t_i) \in F_1 \vee (p,t_j) \in F_2)\} \cup \\
 & & \{(t_i \times t_j, p) : l_1(t_i) = l_2(t_j) = \alpha \in L \wedge \\
 & & \qquad ((t_i,p) \in F_1 \vee (t_j,p) \in F_2)\} \\
\Lambda & = & (\Lambda_1 - \{(t_i, \lambda_i) : t_i \in T_1 \wedge l_1(t_i) = \alpha \in L\}) \cup \\
 & & (\Lambda_2 - \{(t_j, \lambda_j) : t_j \in T_2 \wedge l_2(t_j) = \alpha \in L\}) \cup \\
 & & \{(t_i \times t_j, R_{ij}), R_{ij} = \frac{\lambda_i}{r_\alpha(P)} \frac{\lambda_j}{r_\alpha(Q)} min(r_\alpha(P), r_\alpha(Q)) : \\
 & & \quad t_i \in T_1 \wedge t_j \in T_2 \wedge l_1(t_i) = l_2(t_j) = \alpha \in L\} \\
M_0 & = & M_{0,1} \cup M_{0,2} \\
l(t) & = & \begin{cases} l_1(t) & if \;\; t \in T_1 - T_1^L \\ l_2(t) & if \;\; t \in T_2 - T_2^L \\ \alpha & if \;\; t = t_i \times t_j : t_i \in T_1 \wedge t_j \in T_2 \wedge \\ & \qquad l_1(t_i) = l_2(t_j) = \alpha \in L \end{cases}
\end{array}
$$

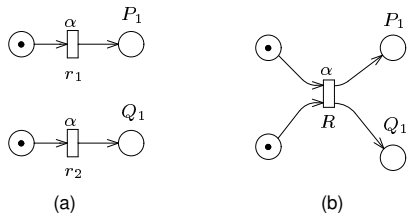

Figure 3: Translation for the cooperation operator.

Choice The translation of $P + Q$ can be expressed as

$$\mathcal{N}[\![P + Q]\!] = +_{\mathcal{N}} (\mathcal{N}[\![P]\!], \mathcal{N}[\![Q]\!])$$

First we build the nets representing the components P and Q, and then we translate the "+" operator considering the Cartesian product of their input places. As an example, consider $R \stackrel{\text{def}}{=} P + Q$ where

$$P \stackrel{\text{def}}{=} (\alpha, r_1).P_1 \| (\beta, r_2).P_2 \qquad Q \stackrel{\text{def}}{=} (\gamma, r_3).Q_1$$

whose translation is depicted in Fig. 4. The two components in isolation are shown in Fig. 4(a) and the net representing R is shown in Fig. 4(b). The initial places in the composed net are obtained considering the Cartesian product of the input places in the starting nets (see the places A, B and C in Fig. 4(a) and the places $A \times C$ and $B \times C$ in Fig. 4(b)).

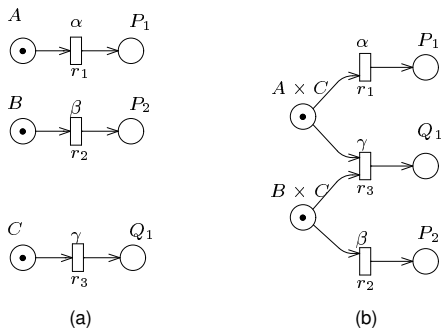

Figure 4: Cartesian product of the input places.

Definition 4 Choice
Let $\mathcal{N}_1 = (P_1, T_1, F_1, \Lambda_1, M_{0,1}, l_1)$ and $\mathcal{N}_2 = (P_2, T_2, F_2, \Lambda_2, M_{0,2}, l_2)$ with $P_1 \cap P_2 = T_1 \cap T_2 = \emptyset$ be the nets modeling two components P and Q. We define the net modeling the component $R = P + Q$ as $\mathcal{N}_{(+)} = (P, T, F, \Lambda, M, l)$ where

$$
\begin{array}{lcl}
P & = & P_1 \cup P_2 - \{M_{0,1} \cup M_{0,2}\} \cup \{M_{0,1} \times M_{0,2}\} \\
T & = & T_1 \cup T_2 \\
F & = & (F_1 - \{(p,t) : p \in M_{0,1} \wedge (p,t) \in F_1\}) \cup \\
 & & (F_2 - \{(q,t) : q \in M_{0,2} \wedge (q,t) \in F_2\}) \cup \\
 & & \{(p \times q, t) : p \in M_{0,1} \wedge q \in M_{0,2} \wedge \\
 & & \qquad ((p,t) \in F_1 \vee (q,t) \in F_2)\} \\
\Lambda & = & \Lambda_1 \cup \Lambda_2 \\
M_0 & = & M_{0,1} \times M_{0,2} \\
l(t) & = & \begin{cases} l_1(t) & if \;\; t \in T_1 \\ l_2(t) & if \;\; t \in T_2 \end{cases}
\end{array}
$$

Hiding The translation of the term P/L is given by

$$\mathcal{N}[\![P/L]\!] = /_{\mathcal{N}} (\mathcal{N}[\![P]\!])$$

and its net representation is shown Fig. 5. For hiding, we also have two different possibilities depending on the fact that action type α belongs to the set L or not. Fig. 5(a) shows the translation of $P \stackrel{\text{def}}{=} (\alpha, r)/\{\beta\}$ and Fig. 5(b) shows the translation of $P \stackrel{\text{def}}{=} (\alpha, r)/\{\alpha\}$, in which the action α belongs to the hiding set. This second net is obtained by simply renaming the transition α with the label τ. Notice that different τ transitions have to be distinguished since they model internal delays.

Figure 5: Translation for the hiding operator.

Definition 5 Hiding
Let $\mathcal{N}_1 = (P_1, T_1, F_1, \Lambda_1, M_{0,1}, l_1)$ be the net representing P. The net modeling the component $Q = P/L$

is given by $\mathcal{N}_{(/)} = (P, T, F, \Lambda, M_0, l)$ where

$$\begin{array}{lcl} P & = & P_1 \\ T & = & T_1 \\ F & = & F_1 \\ \Lambda & = & \Lambda_1 \\ M_0 & = & M_{0,1} \\ l(t) & = & \begin{cases} \tau & if \;\; l_1(t) \in L \\ l_1(t) & if \;\; l_1(t) \notin L \end{cases} \end{array}$$

Recursion In PEPA there is no explicit recursion operator and infinite behaviors can be obtained using constants. For example, the expression $P \stackrel{\text{def}}{=} (\alpha, r).P$ describes a component that can perform an infinite number of α actions.

The translation of recursive terms like P can be performed in two steps. First we translate P as shown in Fig. 6(a) and then we recognize that we have to repeat the same behavior. Instead of repeating the same net structure, we can close the net as shown in the lower part of Fig. 6(b), obtaining an SPN model in which the transition labeled α can fire infinitely many times.

Here we restrict ourselves to a simple form of recursion in which the recursive term is composed of any sequence of prefixing operators. Goltz [5] has defined a more general translation for the CCS recursion introducing places labeled with variables. Such a translation cannot be directly applied to PEPA terms because of the lack of an explicit recursive operator [22].

Definition 6 Recursion
The net modeling the term $P \stackrel{\text{def}}{=} (\alpha_1, r_1) \ldots (\alpha_n, r_n).P$ is obtained starting from the net $\mathcal{N}_1 = (P_1, T_1, F_1, \Lambda_1, M_{0,1}, l_1)$ modeling $(\alpha_1, r_1) \ldots (\alpha_n, r_n)$ and
1) removing the set of places with no outgoing arcs defined as $P_f = \{p \mid \nexists t : (p, t) \in F_1\}$
2) replacing each arc entering in a place $p \in P_f$ by an arc to the initial place in $M_{0,1}$.
The net is defined as $\mathcal{N}_{(A)} = (P, T, F, \Lambda, M_0, l)$ where

$$\begin{array}{lcl} P & = & P_1 - P_f \\ T & = & T_1 \\ F & = & F_1 - \{(t,p) : p \in P_f\} \cup \\ & & \{(t, M_{0,1}) : (t,p) \in F_1 \wedge p \in P_f\} \\ \Lambda & = & \Lambda_1 \\ M_0 & = & M_{0,1} \\ l(t) & = & l_1(t) \end{array}$$

As pointed out in [6], the translation scheme of Definition 6 does not work when combined with choice. The term

$$P + Q \quad \text{where} \quad P \stackrel{\text{def}}{=} (\alpha, r_1).P \qquad Q \stackrel{\text{def}}{=} (\beta, r_2).Q$$

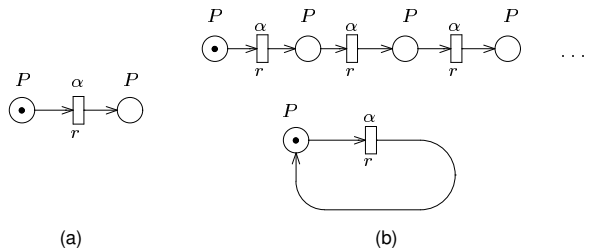

Figure 6: Translation of a recursive term.

represents a system that can witness an infinite number of α or an infinite number of β. The nets modeling the two components P and Q in isolation are shown in Fig. 7(a) and the net of $P + Q$, obtained superposing the input places, is shown in Fig. 7(b). We must notice, however, that this net can execute arbitrary interleavings of α and β action types. The correct translation of $P + Q$ is shown in Fig. 7(c) and it is obtained by "unwinding" the loop by one step. The first transition that fires determines the choice and after that, the net enters into a loop structure in which the same transition is fired infinitely many times. In this way the interleaving of α and β is no longer possible.

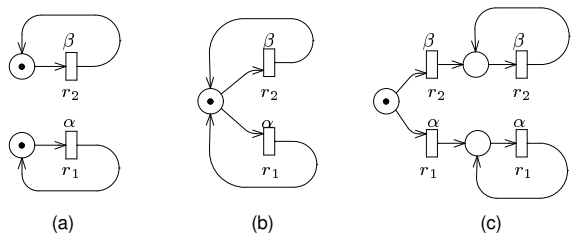

Figure 7: Translation of the term $P + Q$.

Other problems arise when recursion generates parallelism [5], as in the term $P = (\alpha, r).P \underset{\{\alpha\}}{\bowtie} (\alpha, r).P$. After an action α is performed, each instance of P is replaced by two copies of itself so that the number of P components doubles at each recursive step. The rate R at which the cooperation takes place remains the same for each unfolding of the recursion: the apparent rate of action α in each replica of P is equal to r and the minimum of apparent rates is also equal to r.

Fig. 8 shows two SPN models that behave like P. The net in Fig. 8(a) is one-safe but it has an infinite number of places and transitions. The net in Fig. 8(b) has a finite structure but two unbounded places due to the multiplicities 2 on the backward arcs, that are necessary to model pairs of P components introduced

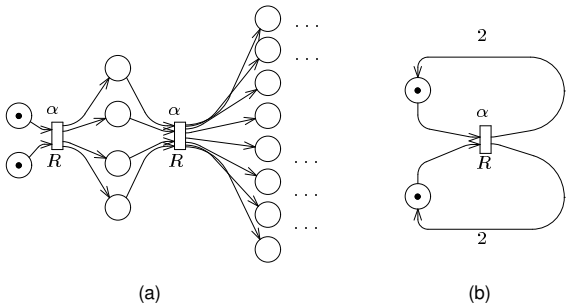

Figure 8: Unbounded parallelism.

at each recursive step.

Notice that Definition 6 does not cover this case. On the other hand a term like P, although syntactically correct, generates an infinite Markov chain in which all the states are transient, and therefore it is not interesting for steady-state analysis purposes.

A simple translation example Let us now consider the expression $P \underset{\{\alpha\}}{\bowtie} Q$ where

$$P \stackrel{\text{def}}{=} (\alpha, r_1).P_1 + (\alpha, r_2).P_2 \quad Q \stackrel{\text{def}}{=} (\alpha, r_3).Q_1 + (\alpha, r_4).Q_2$$

to describe all the operations that have to be performed to derive the corresponding SPN representation. The translation of $P \underset{\{\alpha\}}{\bowtie} Q$ is obtained carrying out the following steps.

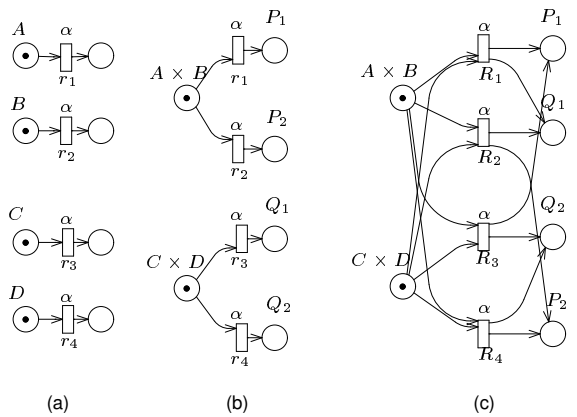

Figure 9: Example of SPN semantics.

1. First we need to translate the four components in isolation as shown in Fig. 9(a);

2. The models of P and Q are then obtained considering the Cartesian product of the input places in the starting nets (see the labels $A \times B$ and $C \times D$ in Fig. 9(b)).

3. The net of $P \underset{\{\alpha\}}{\bowtie} Q$ (see Fig. 9(c)) is obtained by merging pairs of transitions with the same label into single ones to model all the potential cooperations.

4. The synchronization rates are computed considering the apparent rates of P and Q. The apparent rate of α in P is $r_\alpha(P) = r_1 + r_2$, the probability of completing the first instance of α is equal to $\frac{r_1}{r_1+r_2}$ and the probability of completing the second instance is equal to $\frac{r_2}{r_1+r_2}$. For Q we have $r_\alpha(Q) = r_3 + r_4$ and probabilities $\frac{r_3}{r_3+r_4}$ and $\frac{r_4}{r_3+r_4}$ for the first and the second instances, respectively. The four transition rates R_i are then equal to:

$$\begin{aligned} R_1 &= \tfrac{r_1}{r_1+r_2}\tfrac{r_3}{r_3+r_4} min(r_1+r_2, r_3+r_4) \\ R_2 &= \tfrac{r_2}{r_1+r_2}\tfrac{r_3}{r_3+r_4} min(r_1+r_2, r_3+r_4) \\ R_3 &= \tfrac{r_1}{r_1+r_2}\tfrac{r_4}{r_3+r_4} min(r_1+r_2, r_3+r_4) \\ R_4 &= \tfrac{r_2}{r_1+r_2}\tfrac{r_4}{r_3+r_4} min(r_1+r_2, r_3+r_4) \end{aligned}$$

4 Criteria for a good SPN semantics

We analyze in this section three properties that the SPN semantics must satisfy to be considered a good net semantics. The first two criteria called Concurrency and Retrievability have been introduced by Olderog [17] and consider only the functional behavior of the model.

For performance modeling, however, we are interested in the timing aspects of the systems so that the exhibition of the same functional behavior is no longer enough to establish a good net semantics. We therefore need to introduce a new criteria, called Markovian Equivalence, that guarantees that the temporal behaviors of the models are also the same.

We start by considering the two principles concerning the functional behavior. They have been defined for PN models but they can be applied to SPN models as well, since the infinite support of the delay distribution guarantees that the RG of the SPN model is equal to that of the corresponding untimed net.

1) Concurrency: The SPN semantics should represent the intended concurrency of process terms.

In the interleaving semantics of process algebras concurrency is reduced to the non-deterministic choice between all the possible interleavings. Let us consider the timing extension of the processes $A = a|b$ and $B = a.b + b.a$ that we introduced at the beginning of Sec. 3, i.e.

$$\begin{aligned} A' &= (\alpha, r_1) \| (\beta, r_2) \\ B' &= (\alpha, r_1).(\beta, r_2) + (\beta, r_2).(\alpha, r_1) \end{aligned}$$

whose net representations are shown in Fig. 10.

Since SPNs, like Petri nets, are non interleaving models, it is possible to distinguish between concurrency (Fig. 10(a)) and non-deterministic interleaving (Fig. 10(b)) and, hence, our SPN semantics satisfies the first criteria.

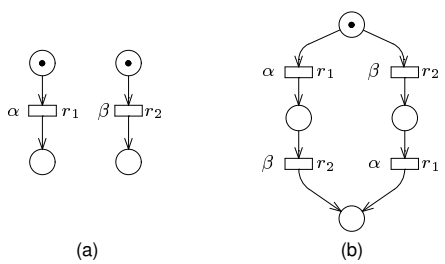

Figure 10: Concurrency and non-deterministic choice.

2) Retrievability: The interleaving semantics for process terms should be retrievable from the SPN semantics.

The interleaving semantics of a PEPA component P is given by its DG while the interleaving semantics of an SPN model is given by its RG. We say that the interleaving semantics is retrievable from the SPN semantics when the DG of any PEPA model and the Reachability Graph (RG) of the corresponding SPN model are strongly bisimilar [19]. The strong bisimulation relation between the DG of a PEPA term and the RG of the corresponding SPN model is derived from the strong bisimulation relation over process terms [13] and is defined as follows[3]

Definition 7 A binary relation $\mathcal{R} \subseteq DG \times RG$ is a strong bisimulation if $(P, M) \in \mathcal{R}$ implies, for all activities $a = (\alpha, r)$, for all transitions $t \in T$ such that $l(t) = \alpha$ and $\Lambda(t) = r$

1. whenever $P \xrightarrow{a} P'$, then for some $M' : M\ [\ t\ \rangle M'$ and $(P', M') \in \mathcal{R}$;

2. whenever $M\ [\ t\ \rangle M'$, then for some $P' : P \xrightarrow{a} P'$ and $(P', M') \in \mathcal{R}$.

For any pair of states (P, M) that are in the relation $\mathcal{R}$[4], any transition $a = (\alpha, r)$ in the DG must be matched by a transition labeled (t, r) in the RG. Moreover, the reached states (P', M') are still in the relation $\mathcal{R}$. This means that any action in the PEPA term is matched by one transition in the SPN model and hence their functional behaviors are the same.

The existence of such a relation $\mathcal{R}$ between the DG and the RG does not imply that they are isomorphic. However, the structure of the RG of a SPN model obtained applying the semantic rules of Sec. 3 ensures that it is possible to partition its states into equivalence classes so that a one-to-one correspondence between the states in the DG and the equivalence classes in the RG may be established [22].

Theorem 1 (Retrievability) The SPN semantics satisfies the Retrievability Principle, i.e.

$$\mathrm{DG(P)} \sim RG(\mathcal{N}[\![P]\!])$$

The proof of this theorem is done by induction on the structure of the term P and it requires a detailed analysis of the reachable markings in $\mathcal{N}[\![P]\!]$.
• Base of the induction: (basic element)
$DG((\alpha, r)) \sim RG(\mathcal{N}[\![(\alpha, r)]\!])$
• Inductive step:
inductive hypothesis:
$DG(P) \sim RG(\mathcal{N}[\![P]\!])$ and $DG(Q) \sim RG(\mathcal{N}[\![Q]\!])$
inductive thesis: we need to prove that

1) $DG((\alpha, r).P) \sim RG(\mathcal{N}[\![(\alpha, r).P]\!])$
2) $DG(P + Q) \sim RG(\mathcal{N}[\![P + Q]\!])$
3) $DG(P \underset{L}{\bowtie} Q) \sim RG(\mathcal{N}[\![P \underset{L}{\bowtie} Q]\!])$
4) $DG(P/L) \sim RG(\mathcal{N}[\![P/L]\!])$
5) $DG(P \stackrel{\text{def}}{=} (\alpha_1, r_1) \ldots (\alpha_n, r_n).P) \sim RG(\mathcal{N}[\![P \stackrel{\text{def}}{=} (\alpha_1, r_1) \ldots (\alpha_n, r_n).P]\!])$

Let us consider the case of the prefixing operator and prove that $DG((\alpha, r).P) \sim RG(\mathcal{N}[\![(\alpha, r).P]\!])$
• For the inductive hypothesis we know that $DG(P) \sim RG(\mathcal{N}[\![P]\!])$.
• Starting from $DG(P)$ we can derive $DG((\alpha, r).P)$ adding a new state $(\alpha, r).P$ and a new transition labeled (α, r) connected to the initial state of $DG(P)$.
• Starting from $\mathcal{N}[\![P]\!]$ we can derive the SPN model $\mathcal{N}[\![(\alpha, r).P]\!]$ with a new initial place p and a transition t, with label α and rate r, connected to all the places initially marked in $\mathcal{N}[\![P]\!]$. $RG(\mathcal{N}[\![(\alpha, r).P]\!])$ is the same as $RG(\mathcal{N}[\![P]\!])$ plus a new state (the initial marking M_0 in which place p is marked) and a new arc connecting M_0 to the initial state of $RG(\mathcal{N}[\![P]\!])$.
• $(\alpha, r).P$ and M_0 satisfy Definition 7 by construction: they both enable a single activity (transition) of type α and the states reached after the completion of such

[3] In Definition 7, P denotes a process term and $P \xrightarrow{a} P'$ represents the evolution of P into process P' after executing activity $a = (\alpha, r)$. M represents a reachable marking and $M\ [\ t\ \rangle M'$ means that the SPN model evolves from marking M to marking M' after the firing of transition t.

[4] Any relation $\mathcal{R}$ that satisfies Definition 7 is a strong bisimulation. The largest strong bisimulation relation, written $\sim$, is the union of all the relations $\mathcal{R}$ [13].

activity are bisimulation equivalent. Therefore we can conclude that $DG((\alpha, r).P) \sim RG(\mathcal{N}[\![(\alpha, r).P]\!])$.

Similarly we can prove the other four equivalences (the complete proof can be found in [22]).

3) Markovian Equivalence: The Markov processes underlying a PEPA model and the corresponding SPN model are lumpable equivalent.

A Markov process is lumpable if it is possible to find a partition of its states such that for any two states within a partition their aggregated transition rates to any other partition are the same [11].

Two Markov processes are lumpable equivalent if there exists a one-to-one correspondence between the lumpable partitions of their states that preserves the aggregated transition rates.

In [8] it has been proved that, in general, strong bisimilarity between PEPA components does not provide sufficient information to deduce any relation between the underlying Markov processes. However, in this translation context, we do not consider any pair of components (i.e. any pair of DGs) but we consider the DG of a PEPA component and the RG of the corresponding SPN model. Indeed, also in this case, we claim that the correspondence between the structure of the DG and that of the RG ensures that the two underlying Markov processes are lumpable equivalent [22].

Unfortunately, this is not enough to satisfy the Markovian Equivalence principle. The translation of the PEPA cooperation (see Sec. 3) works correctly from a functional point of view, but may fail in computing the rates of the transitions that model shared activities; hence, the Markov processes underlying the PEPA specification and the SPN model can be different.

Let us show this fact on a simple example. The term Sys below is composed of three components. Two of them — $Comp_1$ and $Comp_2$ — are independent, but they both must synchronize with $Comp_3$ on actions of type α.

$$
\begin{aligned}
Sys &= (Comp_1 \| Comp_2) \underset{\{\alpha\}}{\bowtie} Comp_3 \\
Comp_1 &= (\beta, \lambda_1).(\alpha, \lambda_2) \\
Comp_2 &= (\alpha, \lambda_3) + (\alpha, \lambda_4) \\
Comp_3 &= (\alpha, \lambda_5).(\alpha, \lambda_6)
\end{aligned}
$$

The translation of the components $Comp_i$ in isolation is shown in Fig. 11 and the SPN model of Sys is shown in Fig. 12. Transitions labeled $\alpha_i\alpha_j$ represent the cooperation between pairs of activities (α, λ_i) and (α, λ_j) and their corresponding rates, denoted with R_{ij}, are

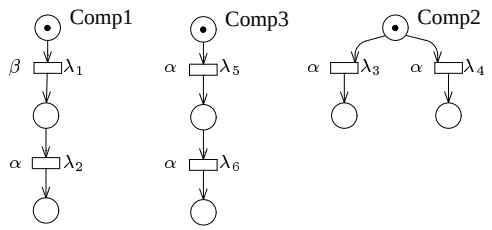

Figure 11: $Comp_1$, $Comp_2$ and $Comp_3$ in isolation.

listed below.

$$
\begin{aligned}
R_{25} &= \tfrac{\lambda_2}{\lambda_2} \cdot \tfrac{\lambda_5}{\lambda_5} \cdot min(\lambda_2, \lambda_5) \\
R_{26} &= \tfrac{\lambda_2}{\lambda_2} \cdot \tfrac{\lambda_6}{\lambda_6} \cdot min(\lambda_2, \lambda_6) \\
R_{35} &= \tfrac{\lambda_3}{\lambda_3+\lambda_4} \cdot \tfrac{\lambda_5}{\lambda_5} \cdot min(\lambda_3 + \lambda_4, \lambda_5) \\
R_{45} &= \tfrac{\lambda_4}{\lambda_3+\lambda_4} \cdot \tfrac{\lambda_5}{\lambda_5} \cdot min(\lambda_3 + \lambda_4, \lambda_5) \\
R_{36} &= \tfrac{\lambda_3}{\lambda_3+\lambda_4} \cdot \tfrac{\lambda_6}{\lambda_6} \cdot min(\lambda_3 + \lambda_4, \lambda_6) \\
R_{46} &= \tfrac{\lambda_4}{\lambda_3+\lambda_4} \cdot \tfrac{\lambda_6}{\lambda_6} \cdot min(\lambda_3 + \lambda_4, \lambda_6)
\end{aligned}
$$

The three transitions $\beta, \alpha_3\alpha_5$ and $\alpha_4\alpha_5$ are enabled in the initial marking and let us suppose that transition β fires first. After this firing the three transitions

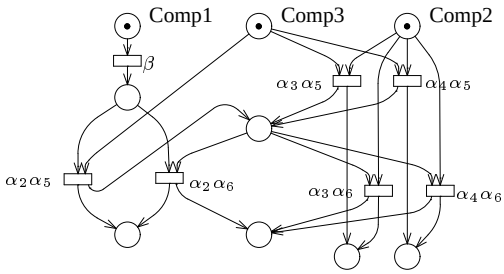

Figure 12: SPN model with a wrong timing behavior.

$\alpha_2\alpha_5, \alpha_3\alpha_5$ and $\alpha_4\alpha_5$ become concurrently enabled and let us suppose further that transition $\alpha_3\alpha_5$ — modeling the cooperation between (α, λ_3) and (α, λ_5) — wins the race. The rate associated with transition $\alpha_3\alpha_5$ is given by

$$R_{35} = \frac{\lambda_3}{\lambda_3 + \lambda_4} \cdot \frac{\lambda_5}{\lambda_5} \cdot min(\lambda_3 + \lambda_4, \lambda_5)$$

but in the PEPA model is equal to

$$R_{35} = \frac{\lambda_3}{\lambda_2 + \lambda_3 + \lambda_4} \cdot \frac{\lambda_5}{\lambda_5} \cdot min(\lambda_2 + \lambda_3 + \lambda_4, \lambda_5)$$

because, after the completion of the β activity, the apparent rate of α is $\lambda_2 + \lambda_3 + \lambda_4$ in $Comp_1 \| Comp_2$ and it is λ_5 in $Comp_3$.

The difference between these two rates is due to the fact that in the first case (SPN model) the rate

R_{35} has been computed (statically) while setting up the net considering the two transitions that have been superposed. In the second case (PEPA model), the apparent rate depends instead on the global state of the model, i.e. on the number of activities of a given type that are concurrently enabled, and it is computed (dynamically) at each derivation step.

To overcome this problem we should associate the following marking-dependent rate[5] with transition $\alpha_3\alpha_5$

$$R_{35} = \begin{cases} \frac{\lambda_3}{\lambda_3+\lambda_4} \cdot \frac{\lambda_5}{\lambda_5} \cdot min(\lambda_3+\lambda_4, \lambda_5) \\ \qquad \text{if transition } \beta \text{ has not fired yet} \\ \frac{\lambda_3}{\lambda_2+\lambda_3+\lambda_4} \cdot \frac{\lambda_5}{\lambda_5} \cdot min(\lambda_2+\lambda_3+\lambda_4, \lambda_5) \\ \qquad \text{otherwise} \end{cases}$$

With the introduction of marking dependent transition rates the SPN semantics satisfies the Markovian Equivalence criteria.

Another solution could be to derive an SPN model starting from the DG of the term we want to translate: nodes in the DG correspond to the places in the SPN model, arcs in the DG correspond to the transitions in the net. The model we obtain exhibits the same behavior (both functional and temporal) as the PEPA term because it is in one-to-one correspondence with its state space. Obviously this is not our idea of SPN semantics because we would like to derive an SPN model starting from the PEPA specification and not from its state space. Moreover, this solution does not satisfy the Concurrency principle since in such a net there is always only one place marked and we completely lose the idea of separate components that work together. As we discuss in the next section this problem does not occur in the other SPAs.

5 Comparison with other SPAs

Other stochastic extensions of process algebras have been defined in the literature. We recall Markovian Timed Performance Process (MTIPP) [7] and two languages both called Markovian Process Algebra [3, 2]. To avoid confusion we use the acronyms MPA(1) for the language of Buchholz and MPA(2) for the language of Bernardo et al.

The SPN semantics of Sec. 3, with minor changes, is still valid for these three languages. Only two of them, MTIPP and MPA(2), provide the *Nil* operator to describe agents that cannot perform any action. This operator can be modeled with SPNs considering a single marked place and its formal definition is the following.

[5] In a similar way marking dependent rates can be computed for the transitions $\alpha_2\alpha_5$ and $\alpha_3\alpha_5$.

Definition 8 Nil
The net representation of the Nil element is

$$\mathcal{N}_{(nil)} = (\{p\}, \emptyset, \emptyset, \emptyset, \{p\}, \emptyset)$$

In MPA(1) the basic element is denoted (α, r). It is assumed that each action α has a fixed rate, equal to μ_α, and the second parameter[6] r represents the number of instances of α concurrently enabled. The duration until an action α is completed is obtained by considering the minimum among a set of random variables, all with exponential distribution, which is still exponentially distributed with a rate that is equal to the sum of all the involved rates. In this case we have a rate equal to $\mu_\alpha + \ldots + \mu_\alpha = r \cdot \mu_\alpha$.

The translation of the basic element is obtained considering r conflicting transitions with label α and rate μ_α, as shown in Fig. 13(a). This net correctly models the MPA(1) basic operator, since r transitions with rate μ_α are enabled in parallel and the duration until one transition is fired is exponentially distributed with rate $r \cdot \mu_\alpha$ (because we assume a race policy between concurrently enabled transitions).

The other operators of the language are translated following the scheme proposed for PEPA, but in this case each basic element has a more complex structure due to the need of repeating the transitions. Fig. 13(b) shows, for example, the translation of the term $P \stackrel{\text{def}}{=} (\alpha, 3).Q$ where $Q \stackrel{\text{def}}{=} (\beta, 3).Q_1 \| (\gamma, 2).Q_2$.

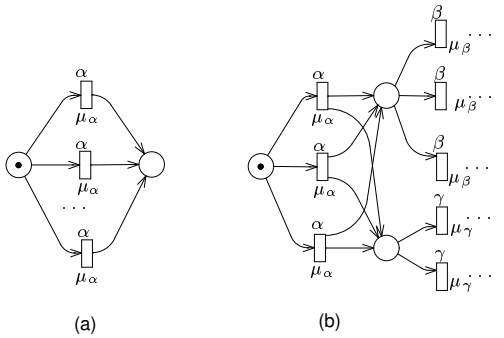

Figure 13: SPN models for MPA(1).

Another difference between PEPA and the other languages concerns the rates they associate with actions that model synchronizations between components. In MTIPP the rate at which a synchronization takes place is equal to the product of the corresponding rates of the individual components. In MPA(1) the rate is equal to the product of the invocations times the rate associated with the action type. Finally, in

[6] In the definition of the language r may be any value; here we restrict to values of $r \in I\!N$.

MPA(2) the maximum of the rates of the starting components is considered. In this last language only one of the interacting components can carry out an active action with a given rate r. All the other components must execute passive[7] actions whose associated rates are equal to zero.

All the choices for the synchronization rates satisfy the algebraic conditions necessary for any equivalence relation to be a congruence relation and therefore they are sound from the algebraic point of view. However, the performance modeler must take into account the differences in the computed rates because the performance measures extracted from the model can be completely different depending on the chosen language. An interesting discussion about the implications due to the choice of a particular kind of synchronization rate can be found in [9].

If we consider, for instance, the example $P \stackrel{\text{def}}{=} (\alpha, r).P \underset{\{\alpha\}}{\bowtie} (\alpha, r).P$ already discussed (see Fig. 8) we can compute the following rates for successive recursive steps.

	PEPA	MTIPP
1) R	$min(r, r) = r$	$r \cdot r$
2) R	$min(min(r, r), min(r, r))$	$r \cdot r \cdot r \cdot r$
	$\cdots$	$\cdots$
i) R	r	r^{2^i}
	MPA(1)	MPA(2)
1) R	$(r \cdot r)r_\alpha$	$max(r, 0) = r$
2) R	$(r \cdot r \cdot r \cdot r)r_\alpha$	$max(max(r, 0), max(0, 0))$
	$\cdots$	$\cdots$
i) R	$r^{2^i} \cdot r_\alpha$	r

In MTIPP we have the product of the rates and in MPA(1) the rate at which action α is completed remains constant (equal to the fixed value r_α), but at each unfolding of the recursion we have the product of the instances available to form the interaction. We can distinguish three different cases. For values of $r > 1$, the agent performs an infinite sequence of α at progressively greater and greater rates R as more copies of the P component are introduced into the interaction; this represents a system that becomes faster at each recursive step. If $r < 1$ the rate R at which the interaction takes place progressively decreases, so that the system becomes slower. Finally, if $r = 1$, the rate remains constant: $R = 1$ in MTIPP and $R = \mu_\alpha$ in MPA(1).

In MPA(2) the rate R at which successive α actions take place is determined by the only active agent and it is always equal to r, like in PEPA, but in this case P requires a different specification since all the components, except one, must carry out passive actions.

[7] A passive action can be executed only in cooperation with an active one which determines its rate. PEPA, MTIPP and MPA(2) provide passive actions, MPA(1) does not.

It is important to note that these three languages satisfy the Markovian Equivalence principle for the definition of a good SPN semantics. The Markov processes underlying the algebraic specifications and the corresponding SPN models are lumpable equivalent and it is not necessary to introduce marking dependent rates as was the case for PEPA.

6 Conclusion

We have defined an SPN semantics for PEPA and we have also briefly described how this semantics can be extended to the other recently proposed SPA languages. This semantics provides a way to understand SPAs using a well established formalism, and it also provides an abstract framework for the comparison of the different languages [22].

The mapping of SPA models into SPNs offers a way to derive different representations (global versus distributed models [1]) of the same system. The problem of the apparent rate computation we encountered during the translation of the PEPA cooperation points out the differences that exist between the global state, typical of the algebras, and the local state typical of the nets.

Our final goal is the integration of the SPN and SPA formalisms so that the properties of the modeled system can be studied by applying different analysis techniques on different system representations. For instance, we can apply simplification techniques based on equivalence notions on the model specified with the algebra or we can apply structural analysis techniques on the corresponding SPN model. The SPN semantics we have defined can be considered as the preliminary step towards this integration.

Acknowledgements

M. Ribaudo is supported in part by the ESPRIT-BRA project No. 7269 "QMIPS" and by a grant from The British Council and MURST-CRUI.

References

[1] M. Bernardo, L. Donatiello, and R. Gorrieri. Describing queueing systems with MPA. Tech. Rep. UBLCS-94-11, Univ. of Bologna, Lab. of Computer Science, 1994.

[2] M. Bernardo, L. Donatiello, and R. Gorrieri. MPA: a stochastic Process Algebra. Tech. Rep. UBLCS-94-10, Univ. of Bologna, Lab. of Computer Science, 1994.

[3] Peter Buchholz. On a Markovian Process Algebra. Tech. Report 500/1994, Univ. of Dortmund, 1994.

[4] P. Degano, R. De Nicola, and U. Montanari. A distributed operational semantics for CCS based on Condition/Events systems. Acta Informatica, 26, 1988.

[5] Ursula Goltz. On representing CCS programs by finite Petri nets. In Mathematical Foundations of Computer Science, volume 324 of LNCS, 1988.

[6] Ursula Goltz. CCS and Petri nets. Technical Report 467, GMD, July 1990.

[7] N. Gotz, U. Herzog, and M. Rettelbach. Multiprocessor and distributed system design: The integration of functional specification and performance analysis using stochastic process algebra. In Tutorial Proc. Performance 1993, Rome, 1994.

[8] Jane Hillston. A Compositional Approach to Performance Modelling. PhD thesis, Computer Science Department, University of Edinburgh, April 1994.

[9] Jane Hillston. The nature of synchronization. In U. Herzog and M. Rettelbach, editors, Proc. 2*nd* Workshop on Process Algebra and Performance Modelling, Erlangen, 1994.

[10] C.A.R. Hoare. Communicating Sequential Processes. Prentice-Hall, 1985.

[11] J.G. Kemeny and J.L. Snell. Finite Markov Chains. Van Nostrand, Princeton, NJ, 1960.

[12] Antoni Mazurkiewicz. Trace theory. In Petri Nets: Applications and Relationship to Other Models of Concurrency, volume 255 of LNCS, 1987.

[13] Robin Milner. Communication and Concurrency. Prentice Hall, 1989.

[14] M. K. Molloy. Performance analysis using stochastic Petri nets. IEEE Transaction on Computers, 31(9):913–917, September 1982.

[15] T. Murata. Petri nets: properties, analysis, and applications. Proceedings of the IEEE, 77(4):541–580, April 1989.

[16] X. Nicollin and J. Sifakis. An overview and synthesis on Timed Process Algebra. In Real-Time: Theory in Practice, volume 600 of LNCS, 1991.

[17] E.R. Olderog. Operational Petri nets semantics for CCSP. In Advances in Petri Nets, volume 266 of LNCS, 1987.

[18] E.R. Olderog. Nets, Terms and Formulas, volume 23. Cambridge Tracts in Theoretical Computer Science, Cambridge University Press, 1991.

[19] D. Park. Concurrency and automata on infinite sequences. In Proc. 5*th* GI Conf. on Theoretical Computer Science, volume 104 of LNCS, 1981.

[20] G.D. Plotkin. An operational semantics for CSP. Technical Report CSR-114-82, The University of Edinburgh, May 1982.

[21] M. Ribaudo. Understanding Stochastic Process Algebras via their Stochastic Petri Nets Semantics. In U. Herzog and M. Rettelbach, editors, Proc. 2*nd* Workshop on Process Algebra and Performance Modelling, Erlangen, 1994.

[22] M. Ribaudo. On the relationship between Stochastic Petri Nets and Stochastic Process Algebras. PhD thesis, Dip. di Informatica, Univ. di Torino, 1995.

[23] Dirk Taubner. Finite Representation of CCS and TCSP Programs by Automata and Petri nets, volume 369 of LNCS, 1990.

[24] Rob van Glabbeek and Frits Vaandrager. Petri nets models for algebraic theory of concurrency. In PARLE Parallel Architectures and languages Europe, volume 259 of LNCS, 1987.

[25] G. Winskel. Event structures. In Petri Nets: Applications and Relationship to Other Models of Concurrency, volume 255 of LNCS, 1987.