Quantitative
Modeling
In
Parallel
Systems

Part 4
Understanding Stochastic Process Algebras via their Stochastic Petri Nets Semantics

Marina Ribaudo
Università di Torino

Approved by: Gianfranco Balbo

Abstract Recently several Stochastic Process Algebras have been proposed for performance modelling. At a superficial level they appear to be very similar but there are subtle differences between them. In this paper three of these languages, PEPA, TIPP and Buchholz's MPA, will be compared on the basis of their SPN semantics. These semantics exhibit both similarities and differences in the three algebras, as well as giving some insight into the relationship between the SPN and SPA approaches to performance modelling.

4.1 Introduction

In recent decades parallel and distributed systems have been extensively used in many application areas and several formal models for their description and analysis have been proposed.

Among these formal models, we are interested in Process Algebras — abstract languages for the description of complex systems following a compositional approach, and Petri nets — a graphical and mathematical modelling tool particularly well suited for the description of concurrent systems.

These formalisms had a similar development. In their original proposals they did not include any notion of time and they were used for the analysis of the functional aspects of the system, i.e. the actions the system can perform.

Later on, to take into account aspects related to temporal requirements, several extensions of the basic modelling formalisms have been proposed. We are particularly interested in two of them called Stochastic Process Algebras (SPA) and Stochastic Petri nets (SPN), respectively. These extensions, in which time is represented by means of random variables, allow to reason about the performance of a system starting from the first phase of the design, probably before the system has been completely specified.

In this paper we will reduce three different SPAs into a common framework (SPN). The comparison among the three languages will be given on the basis of the (small) differences that result in their SPN representations.

Some of the difficulties encountered during the development of the translations will be discussed. In particular the cooperation operator in PEPA proved to be very difficult to represent in a way which preserved the timing behaviour. A resolution of this problem will be described together with the reasons why it is not completely satisfactory.

The paper is organized as follows. In the next two sections we introduce the two formalisms. In Section 4.4 we will define a SPN semantics for each SPA, and in Section 4.5 a comparison among the languages will be discussed. Finally, Section 4.6 concludes this work.

4.2 Petri Nets

Petri nets (PN) are a graphical and mathematical modelling tool for describing concurrent systems. Formally a PN structure is a 4−tuple

$$\mathcal{N} = (P, T, I, O)$$

where

$P = \{p_1, p_2 \ldots, p_n\}$	is the set of places
$T = \{t_1, t_2 \ldots, t_n\}$	is the set of transitions
$I \subseteq P \times T$	is the set of input arcs
$O \subseteq T \times P$	is the set of output arcs

The state of the model is defined by the number of tokens (represented as black dots) contained in each place and it is specified by the function

$$M : P \rightarrow I\!N$$

called *marking*, that for each place $p \in P$ specifies the number of tokens contained in p.

A PN system is given by a PN structure plus an *initial marking* and it is defined as a 5−tuple

$$\mathcal{M} = (P, T, I, O, M_0)$$

where M_0 represents the initial distribution of tokens in the places of the net.

The *dynamic* behaviour of the system is specified by the *enabling* and *firing* rules. When all the input places for a transition t contain one or more tokens, the transition is enabled[1]. An enabled transition t can *fire* consuming tokens from its input places and adding tokens into its output places. PNs incorporate a notion of distributed state which is pointed out by the enabling and firing rules

[1]This is true only in the case of *ordinary* nets, i.e. nets with arcs with multiplicity equal to 1.

because only local information (i.e. input and output places) is sufficient for modifying the state of the system.

Starting from the initial marking M_0 it is possible to compute the set of all the reachable markings, i.e. the *Reachability Set* (RS) of the model. The RS does not contain information about the transition sequences fired to reach each marking. This information is captured by the *Reachability Graph* (RG) whose nodes are labelled with the reachable markings and whose arcs are labelled with the transitions that the system had to fire to move from state to state.

4.2.1 Stochastic Petri Nets

Classical PN models include no notion of time and for this reason they have been traditionally used for the qualitative analysis of *logical* properties of systems. Several authors have proposed augmented PN models which include temporal specifications, so that a quantitative performance analysis of systems is possible. The introduction of temporal specifications in a PN has been done mostly by associating a delay with transitions. This delay represents the time interval for which the transition must remain enabled before it can fire.

Stochastic Petri Nets (SPN) [10] are PN augmented with temporal specifications in which transition firing delays are exponentially distributed random variables: each transition t_i is associated with a random firing delay whose probability density function is a negative exponential with rate λ_i.

The semantics of SPN can be described by means of a *race model*. When a marking simultaneously enables several (conflicting and/or concurrent) transitions, all activities associated with these transitions are assumed to execute in parallel, so that the next marking change is due to the transition whose firing delay in the present marking is minimum, i.e. to the transition that wins the race. The firing of the winning transition implies that the activity associated with it in the model is completed, so that, if the transition is enabled again in a future marking, it starts working anew. The behaviour of the transitions that lose the race can be specified in different ways. In fact, it is possible for these transitions to either remember the time during which they have already been enabled (and thus worked), or not.

The use of exponential distributions for the definition of temporal specifications is particularly attractive for two different reasons. First, SPN can be mapped onto continuous-time Markov chains (CTMC), and, second, the memoryless property of the exponential distribution makes unnecessary the distinction between the distribution of the delay itself, and the distribution of the remaining delay after a change of state, thus avoiding the need for the specification of the behaviour of the transitions that are enabled, but do not fire in a given marking.

Let $E(M)$ represent the set of transitions enabled in marking M. When $E(M)$ contains more than one element, the probability that transition t_k is the one that actually fires (wins the race) can be obtained from the temporal specifications as

$$P\{t_k|M\} = \frac{\lambda_k}{\sum_{j \in E(M)} \lambda_j}$$

The behaviour of SPN models is identical to that of Markovian models in which states are in one-to-one correspondence with the SPN markings, and arcs corresponding to state changes are labelled with the rate associated with the transition that produces the change of marking in the SPN [11].

The translation of a SPN model into a CTMC is thus conceptually very simple. The RG of the SPN is generated, and the firing rates of enabled transitions are used to construct the infinitesimal generator of the CTMC. The steady-state distribution over the states of the CTMC model can then be computed by solving the usual set of linear equations.

Difficulties may arise due to the computational complexity of the algorithm for this solution, when the number of reachable markings grows. In fact, this is the main problem associated with the utilization of SPN which are otherwise very easy to employ, even for inexperienced users.

4.3 Process Algebras

Process Algebras — see for example CCS [9] and CSP [8] — are abstract languages developed for describing parallel and distributed systems. *Processes* are the building blocks of the language and they describe sequential components that run concurrently and cooperate through communication. Complex systems are built starting from the building blocks and by applying the constructors of the algebra: there are operators available for composition as well as mechanisms for abstraction to disregard internal details. Processes execute atomic actions; there is also a special action, called silent or perfect, which is denoted with τ and models an internal activity. A major goal of Process Algebras is the algebraic treatment of system descriptions by means of a simple but powerful calculus. There is, in fact, a set of transformation rules that allows to build and compare systems with the same behaviour.

In the following sections we will presents three different languages, [5, 6, 4, 1], extensions of standard Process Algebras, that have been introduced with the main purpose of integrating performance aspects in the functional analysis of complex systems. Process Algebras allow compositional reasoning and have notions of observational equivalence which determine when systems behave in the same way with respect to the environment. Modelling the timing behaviour by random variables is fundamental for the performance evaluation since in this way it is possible to capture the randomness of the real world. By merging logical and time specifications the authors claim to obtain a constructive methodology for *specifying* and *evaluating* complex systems starting from the analysis of their individual components.

4.3.1 Performance Evaluation Process Algebra

Performance Evaluation Process Algebra (PEPA) [6, 5] is a Stochastic Process Algebra (SPA) derived from CCS and CSP in which time is incorporated into the algebra by associating a duration with each activity; random variables are used to characterize such durations.

Systems are described as interactions of *components*. Each component can perform some activities. An activity a is described as a pair (α, r), where α is the *type* of the activity and r is the parameter of the (negative) exponential distribution determining its duration. Whenever a process P can perform an action, an instance of the 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:

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

Prefix is the basic mechanism by which the behaviours of components are constructed. The component $(\alpha, r).P$ carries out activity (α, r), which has action type α and a duration which is exponentially distributed with parameter r (mean $1/r$) and then it continues as process P. When r is unspecified (i.e. it is equal to $\top$) the component is *passive* with respect to the action type. A passive activity must be executed together with an active one which determines its rate.

The component $P + Q$ represents a system which may behave either as component P or as Q. $P + Q$ enables all the current activities of P and Q; the first activity to complete identifies one of the components which is selected as the component that continues its action; the other component of the choice is instead discarded. The system will subsequently behave as P' or Q' respectively, where P' is the component which results from P completing the activity first, and similarly for Q'.

The component $P \underset{L}{\bowtie} Q$ represents a system in which processes P and Q work together to perform some activities. The set L is named the *cooperation set* and defines the action types on which the components must synchronise or *cooperate*. Both components 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 will have the same action type as the two contributing activities and a rate reflecting the rate of the slower component. The expected duration of a shared activity will be greater than or equal to the expected durations of the corresponding activities in the cooperating components. When the set L is empty, $\underset{\emptyset}{\bowtie}$ has the effect of parallel composition, allowing components to proceed concurrently without any interaction between them.

The component P/L behaves as P except that any activities of types within the set L are *hidden*, meaning that their type is not visible upon completion. Instead they appear as the unknown type τ and can be regarded as internal delays by the component.

Constants are components whose meanings are given by equations such as $A \stackrel{\text{def}}{=} P$ to indicate that constant A behaves like the component P. Constants are also used to describe infinite behaviours.

A *race condition* governs the dynamic behaviour 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 probability that a particular activity completes will be given by the ratio between the activity rate of that activity and the sum of the activity rates of all the enabled activities. Whenever an activity completes, the behaviour of the model may change, as it takes on the behaviour of the resulting component.

The semantics of PEPA, presented in the structural operational semantics style of [12], is shown in Figure 4.1. The operational rules are to be read as follows: if the transition(s) above the inference line can be inferred, then we can infer the transition below the line. The rules outline the activities that a component can witness.

The first rule for the prefixing operator states that in the component $(\alpha, r).P$ an activity of type α is enabled and the component behaves like P after its execution. The delay associated with the activity is exponentially distributed with rate r. The other rules are straightforward and are presented without comment except for the rules concerning cooperation. The first and the second cooperation rules represent the independent evolution of the two components while they are executing activities not in the cooperation set L. The third rule represents instead the cooperation between the two components to achieve a common goal. In this rule the two new concepts of *shared activity* and *apparent rate* are introduced. During the cooperation a new shared activity is created and it is assumed that in general both components of the cooperation will need to complete some work, as reflected by their own version of the activity, for the shared activity to be completed.

The rate R of the shared activity is given by 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 the apparent rate of action type α in the P component and it is the rate at which an action type α appears to occur to an external observer. For any component P, its 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 by P. It is assumed that when P and Q carry out action α in cooperation their total capacity to complete the action is limited to the capacity of the slower one, i.e. the apparent rate of the shared activity is equal to the minimum of the apparent rates of α in the two contributing components. Moreover each component may enable several α activities and it is assumed that each independently chooses which of these activities will take part in the cooperation. The rate R associated with the shared activity is thus adjusted to reflect the probabilities of choosing particular instances in the two components (i.e. $\frac{r_1}{r_\alpha(P)}$ and $\frac{r_2}{r_\alpha(Q)}$).

On the basis of the semantic rules, PEPA may be regarded as a labelled multi-transition system in which the number of instances of a transition between states is recognised. This transition system, called the *Derivation Graph*, describes the possible behaviour of any PEPA component and it provides a useful way to reason about the behaviour of a model. Nodes in the Derivation Graph are labelled by PEPA components and arcs are labelled by action types and activity rates (observe the analogies between the Derivation Graph and the Reachability Graph).

The Markov process underlying any finite PEPA component is obtained starting from the Derivation Graph: a state 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 labelling the arcs connecting the corresponding nodes in the Derivation Graph. 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

Prefix

$$\frac{}{(\alpha, r).P \xrightarrow{(\alpha,r)} P}$$

Choice

$$\frac{P \xrightarrow{(\alpha,r)} P'}{P + Q \xrightarrow{(\alpha,r)} P'} \qquad \frac{Q \xrightarrow{(\alpha,r)} Q'}{P + Q \xrightarrow{(\alpha,r)} Q'}$$

Cooperation

$$\frac{P \xrightarrow{(\alpha,r)} P'}{P \bowtie_L Q \xrightarrow{(\alpha,r)} P' \bowtie_L Q} (\alpha \notin L) \qquad \frac{Q \xrightarrow{(\alpha,r)} Q'}{P \bowtie_L Q \xrightarrow{(\alpha,r)} P \bowtie_L Q'} (\alpha \notin L)$$

$$\frac{P \xrightarrow{(\alpha,r_1)} P' \quad Q \xrightarrow{(\alpha,r_2)} Q'}{P \bowtie_L Q \xrightarrow{(\alpha,R)} P' \bowtie_L Q'} (\alpha \in L) \quad \text{where } R = \frac{r_1}{r_\alpha(P)} \frac{r_2}{r_\alpha(Q)} \min(r_\alpha(P), r_\alpha(Q))$$

Hiding

$$\frac{P \xrightarrow{(\alpha,r)} P'}{P/L \xrightarrow{(\alpha,r)} P'/L} (\alpha \notin L) \qquad \frac{P \xrightarrow{(\alpha,r)} P'}{P/L \xrightarrow{(\tau,r)} P'/L} (\alpha \in L)$$

Constant

$$\frac{P \xrightarrow{(\alpha,r)} P'}{A \xrightarrow{(\alpha,r)} P'} (A \stackrel{\text{def}}{=} P)$$

Figure 4.1: Operational Semantics of PEPA.

arcs between the same states in the Derivation Graph are coalasced into a single one in the Markov chain.

4.3.2 Markovian Timed Performance Processes

Markovian Timed Performance Processes (MTIPP) [4], like PEPA, was developed with the aim of having a support for compositional design and performance evaluation of distributed and parallel systems. The set of MTIPP terms is given by the following grammar:

$$P ::= 0 \mid (\alpha, \lambda).P \mid P + P \mid P \|_S P \mid P \setminus L \mid recX : P$$

In MTIPP both a Nil (0) process and a recursion operator (rec) are explicitly represented. The other operators have the same meaning of PEPA. The MTIPP parallel composition operator is denoted with $\|_S$, where S is the set of joint actions, and it corresponds to the PEPA cooperation $\bowtie_L$.

The semantics of the calculus is given again by a set of deduction rules defined in the style of Plotkin. The rules concerning the parallel composition among processes are shown in Figure 4.2. In MTIPP the synchronization between two or more components leads to the observation of an action

$$\frac{P \xrightarrow{(\alpha,r)} P'}{P\|_S Q \xrightarrow{(\alpha,r)} P'\|_S Q}\ (\alpha \notin S) \qquad \frac{Q \xrightarrow{(\alpha,r)} Q'}{P\|_S Q \xrightarrow{(\alpha,r)} P\|_S Q'}\ (\alpha \notin S)$$

$$\frac{P \xrightarrow{(\alpha,r_1)} P' \quad Q \xrightarrow{(\alpha,r_2)} Q'}{P\|_S Q \xrightarrow{(\alpha,r_1 \cdot r_2)} P'\|_S Q'}\ (\alpha \in S)$$

Figure 4.2: Transition rules for the Parallel composition in MTIPP.

with a rate equal to the product of the rates of the individual components.

4.3.3 Markovian Process Algebra

Markovian Process Algebra (MPA) [1] is a SPA based on PEPA and MTIPP with some extensions introduced to compute the state space and the labelled transition system compositionally. In [1] it is shown how the composition operators on the level of the algebra can be represented by operators combining transition matrices on the level of the underlying CTMC.

The syntax of the language is the following

$$P ::= 0 \ | \ (a,r).P \ | \ P + P \ | \ P\|_S P \ | \ P \setminus L \ | \ recX : P$$

In this language each activity is represented by a pair (a, r) in which a is the action an name and r describes the number of invocations of such action. Usually, an action is invoked only once by an agent, but r can assume also values greater than one. It is assumed that each action a has an associated fixed exponential distribution with rate μ_a.

Also for this language the semantics is given in an operational style. We will describe the semantics of all the operators in more details in Section 4.4.3 when we will define the SPN semantics of MPA.

4.4 SPN semantics for each SPA

In the next sections we will describe a SPN semantics for the three languages previously introduced. Following the same approach taken by Goltz in [3], we will define a compositional SPN semantics for SPA. Assuming that we have already the nets representing the behaviours of two terms P and Q we want to infer the behaviour of the term P *op* Q (where *op* is a syntactic operator of the language) by defining a composition of the corresponding nets. Following the organization of Section 4.3, we first introduce the SPN semantics for PEPA, MTIPP, and finally for MPA. For each operator *op* in the language we will show the operation that has to be performed at the net level to reproduce its effect.

4.4.1 SPN semantics for PEPA

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. The translation of the basic element is shown in Figure 4.3. We call P_1 the input place and P_2 the output place of the basic element and we consider a labelling function[2] $l : T \to Act$ that maps transition names into action names, so that we have $l(T_1) = \alpha$. In PEPA (as well as in MTIPP) both active and passive actions are present. We assume that both kinds of actions are modelled with timed transitions and that it is the associated rate that determine the type of the corresponding action.

[2]In the sequel we will name transitions with the corresponding action name.

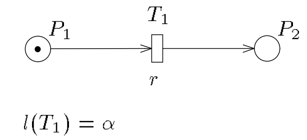


Figure 4.3: Translation of the basic element.

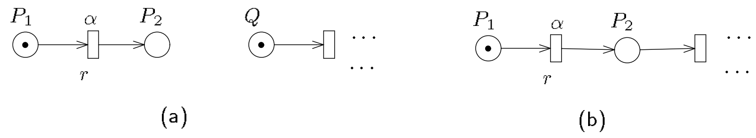


Figure 4.4: Translation of the Prefixing operator.

The execution of a component associated with a net terminates when, after the firing of a certain number of transitions, no more transitions remain enabled. In the simplest case of a component executing a single activity, its completion is represented by the firing of the corresponding transition.

Starting from the representation of the basic element, we can now define the rules to map the language operators into SPN models.

Prefixing Prefixing is the basic mechanism to build expressions in the language. The translation of the term $P = (\alpha, r).Q$ can be formalized as

$$Transl(\cdot)[Transl(\alpha, r), Transl(Q)]$$

and it is shown in Figure 4.4. The expression $Transl(\cdot)[Transl(\alpha, r), Transl(Q)]$ has to be read as follows: first we translate the components inside square brackets, i.e. the action (α, r) and the term Q (see Figure 4.4(a)), and then we translate the "$\cdot$" operator. The translation of the "$\cdot$" is obtained superposing the output place of the net associated to the left part of the expression (place P_2) with the input place of the net associated to the right part of the expression (place Q) as shown in Figure 4.4(b).

Choice The translation of the choice operator is achieved with the superposition of places. The translation of $P + Q$ can be expressed as

$$Transl(+)[Transl(P), Transl(Q)]$$

In this case, first we build the nets representing the components P and Q and then we translate the "+" operator superposing their input places. The translation of the term $P + Q$ where $P = (\alpha, r_1).P_1$ and $Q = (\beta, r_2).Q_1$ is shown in Figure 4.5. The two components are initially translated in isolation as shown in Figure 4.5(a) and then composed into a single net as shown in the right part of the figure. When place $P+Q$ is marked both transitions α and β become enabled and the choice of the transition to fire first is performed according to the *race* policy.

When the starting nets have more that one input place the superposition of places is obtained considering the Cartesian product of the input places. As an example, consider the following term

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

whose translation is depicted in Figure 4.6. The two components in isolation[3] are shown in Figure 4.6(a). The net representing the composed term R is shown if Figure 4.6(b); in this context we want to point out that the initial places of this net are obtained considering the Cartesian product of

[3] Notice that we have not explained yet how to translate components like P.

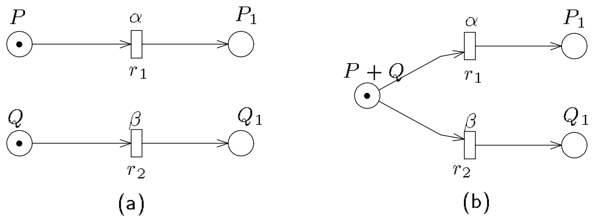


Figure 4.5: Translation of the Choice operator.

the input places in the starting nets (see places labelled with A, B and C in Figure 4.6(a) and places labelled with $A \times C$ and $B \times C$ in Figure 4.6(b)).

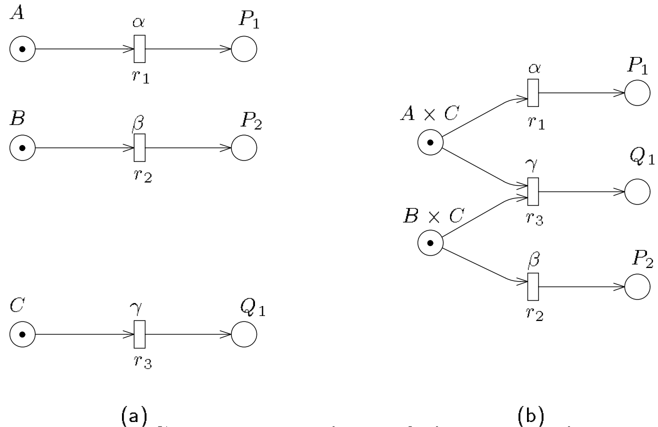


Figure 4.6: Cartesian product of the input places.

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

$$Transl(\underset{L}{\bowtie})[Transl(P), Transl(Q)]$$

Again, 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 have to 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 their rates applying the definition given in the semantic rule of Figure 4.1.

The net depicted in Figure 4.7(a) shows the translation of the term $P = (\alpha, r_1).P_1 \underset{\emptyset}{\bowtie} (\alpha, r_2).Q_1$. The set of joint actions is empty and thus both processes execute their actions independently. Parallelism is modelled by transitions that are enabled in the same marking such that when one transition fires the other is still enabled: the transitions labelled with α in Figure 4.7(a) are of this kind and model two completely independent activities.

Figure 4.7: Translation for the Cooperation operator.

A different situation arises in the case of the term $P = (\alpha, r_1).P_1 \underset{\{\alpha\}}{\bowtie} (\alpha, r_2).Q_1$ 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 both 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 Figure 4.7(b). The rate R to be associated with the transition is equal to $\frac{r_1}{r_\alpha(P)}\frac{r_2}{r_\alpha(Q)}min(r_\alpha(P), r_\alpha(Q)) = \frac{r_1}{r_1}\frac{r_2}{r_2}min(r_1, r_2) = min(r_1, r_2)$.

Hiding The translation of the term $P \setminus L$ is given by

$$Transl(\setminus L)[Transl(P)]$$

and its net representation is shown Figure 4.8. Also for hiding, we have two different possibilities depending on the fact that action type α belongs to the set or not.

Figure 4.8(a) shows the translation of the term $P = (\alpha, r) \setminus \{\beta\}$ and Figure 4.8(b) shows the translation of the term $P = (\alpha, r) \setminus \{\alpha\}$, in which the action α belongs to the cooperation set. This second net is obtained by simply renaming the transition α with the label τ.

(a) (b)

Figure 4.8: Translation for the Hiding operator.

Recursion In PEPA there is no explicit recursion operator and infinite behaviours can be obtained considering constants. For example, the expressions

$$P \stackrel{\text{def}}{=} (\alpha, r).Q \qquad Q \stackrel{\text{def}}{=} P$$

describe a component that can perform an infinite number of α actions. The translation of recursive terms can be performed in two steps. First we translate component P as shown in Figure 4.9(a) and then we recognize that $Q \stackrel{\text{def}}{=} P$ means that we have to repeat the same net structure. Instead of repeating the same net structure, we can close the net as shown in Figure 4.9(b), obtaining a SPN model in which transition labelled α can fire infinitely many times. As pointed out in [3], this scheme

(a) (b)

Figure 4.9: Translation of the Recursion operator.

does not work when combined with choice. Figure 4.11 shows an example of this kind. The term

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

can witness an infinite number of α as well as an infinite number of β. The nets representing the two components in isolation are shown in Figure 4.10. The net of the term $P + Q$ obtained superposing the input places, as required by the translation rule of the choice operator, is shown in Figure 4.11(a). 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 Figure 4.11(b) and it is obtained by "unrolling" the loop of 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.

A simple translation example Let us now consider the term

$$P \underset{\{\alpha\}}{\bowtie} Q \quad \text{where} \quad P = (\alpha, r_1).P_1 + (\alpha, r_2).P_2 \qquad Q = (\alpha, r_3).Q_1 + (\alpha, r_4).Q_2$$

Figure 4.10: Components P and Q translated in isolation.

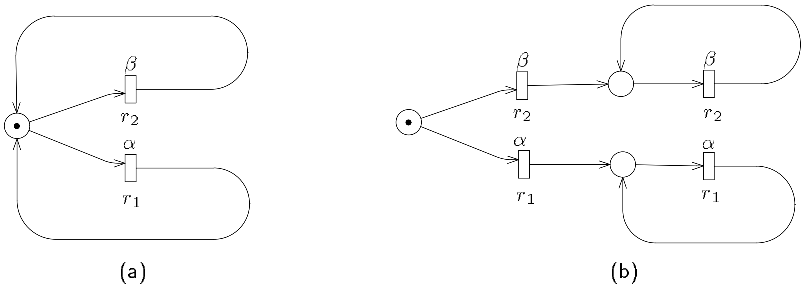


Figure 4.11: Translation of the term $P + Q$.

Figure 4.12(a) shows the translation of processes P and Q in isolation: the net in the upper part is relative to process P, that in the lower part represents process Q. The net resulting from the translation of $P \underset{\{\alpha\}}{\bowtie} Q$ (see Figure 4.12(b)) is obtained by merging pairs of transitions with the same label into single ones to model all the potential cooperations between the two components. The

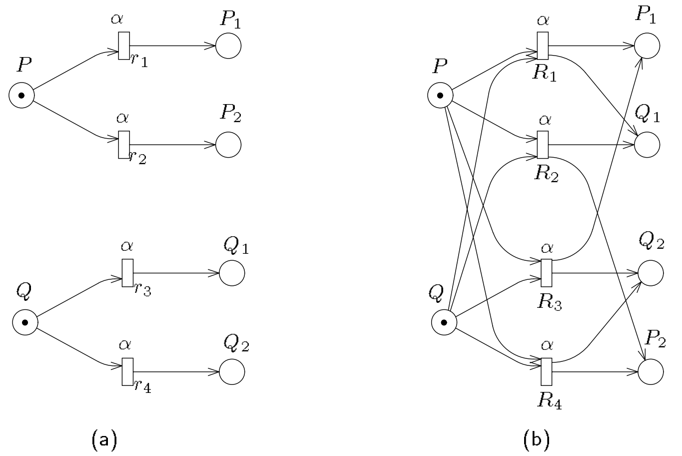


Figure 4.12: Example of SPN semantics.

apparent rate of α in the P component 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 the Q component 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 transitions rates R_i in the whole net are then equal to:

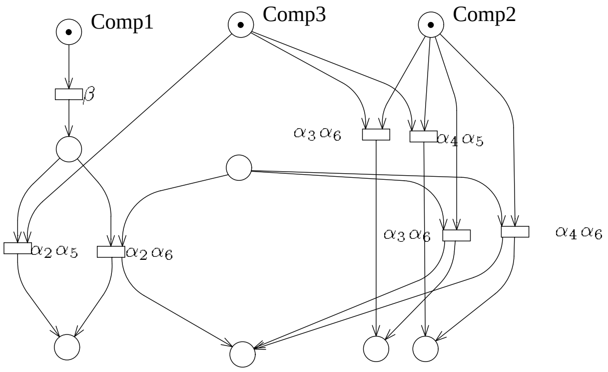
$$R_1 = \frac{r_1}{r_1+r_2}\frac{r_3}{r_3+r_4}min(r_1+r_2, r_3+r_4) \qquad R_2 = \frac{r_2}{r_1+r_2}\frac{r_3}{r_3+r_4}min(r_1+r_2, r_3+r_4)$$
$$R_3 = \frac{r_1}{r_1+r_2}\frac{r_4}{r_3+r_4}min(r_1+r_2, r_3+r_4) \qquad R_4 = \frac{r_2}{r_1+r_2}\frac{r_4}{r_3+r_4}min(r_1+r_2, r_3+r_4)$$

Problem: the apparent rate The translation schema we have given for the cooperation works correctly from a functional point of view, but may fail in capturing the temporal behaviour of the term (i.e. the underlying CTMCs are different). Let us consider for example the following expression

$$Sys = (Comp_1 \underset{\emptyset}{\bowtie} Comp_2) \underset{\{\alpha\}}{\bowtie} Comp_3$$
$$Comp_1 = (\beta, \lambda_1).(\alpha, \lambda_2) \quad Comp_2 = (\alpha, \lambda_3) + (\alpha, \lambda_4) \quad Comp_3 = (\alpha, \lambda_5).(\alpha, \lambda_6)$$

Figure 4.13 shows the SPN model obtained translating the term Sys. Transitions labelled $\alpha_i\alpha_j$ model the cooperation between pairs of activities (α, λ_i) and (α, λ_j) and their corresponding rates are denoted with R_{ij} (see the right part of Figure 4.13).

In the initial marking the three transitions $\beta, \alpha_3\alpha_5$ and $\alpha_4\alpha_5$ are enabled together. Let us suppose that transition β fires first. At this point the three transitions $\alpha_2\alpha_5, \alpha_3\alpha_5$ and $\alpha_4\alpha_5$ are concurrently



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

Figure 4.13: SPN model with a wrong timing behaviour.

enabled. Let us suppose further that transition $\alpha_3\alpha_5$ — modelling the cooperation between (α, λ_3) in $Comp_2$ and (α, λ_5) in $Comp_3$ — wins the race.

The rate associated to transition $\alpha_3\alpha_5$ is equal to $R_{35} = \frac{\lambda_3}{\lambda_3+\lambda_4} \cdot \frac{\lambda_5}{\lambda_5} \cdot min(\lambda_3 + \lambda_4, \lambda_5)$ in the SPN model but it is equal to $\frac{\lambda_3}{\lambda_2+\lambda_3+\lambda_4} \cdot \frac{\lambda_5}{\lambda_5} \cdot min(\lambda_2 + \lambda_3 + \lambda_4, \lambda_5)$ in the PEPA model because, after the completion of the β activity, the apparent rate of α is equal to $\lambda_3 + \lambda_4, \lambda_5$ in $Comp_1 \underset{\emptyset}{\bowtie} Comp_2$ and it is equal to 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 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 to transition $\alpha_3\alpha_5$ the following marking dependent rate:

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

Unfortunately, the definition of marking dependent transition rates can be too difficult when modelling complex systems because it requires the analyst to have in mind the dynamic behaviour of the whole system while setting up the model.

An alternative approach might be that of restricting the language we want to map into SPNs by imposing, for example, that activities with the same action type must have the same rate. With this restriction we no longer have the problem of computing the minimum of the apparent rates in a synchronization but we still have problems in the computation of the probability of choosing a given instance.

Another solution could be to derive a SPN model starting from the LTS of the term we want to translate: nodes in the LTS correspond to the places in the SPN model, arcs in the LTS correspond to the transitions of the net. The model we obtain exhibits the same behaviour (both functional and temporal) as the PEPA term because it is in one to one correspondence with its state space. In fact we have simply found a different way of describing the state space of the PEPA component. Obviously this is not our idea of SPN semantics because we would like to derive a SPN model starting from the PEPA specification and not from its state space. Moreover in such a net we have always only one place marked and we loose completely the idea of separate components that work together: we are in fact modelling the behaviour of the system instead of deriving it from the model of the structure of the system.

4.4.2 SPN semantics for MTIPP

The SPN semantics for MTIPP is shown in Figure 4.14. In this language there are two basic ele-

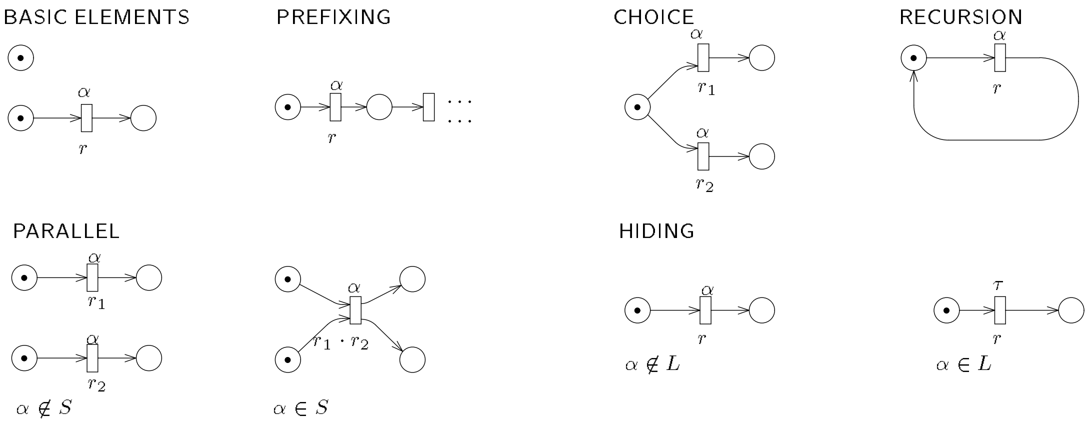


Figure 4.14: SPN semantics for MTIPP.

ments: the Nil (0) operator and the single activity. The Nil operator is translated with a single place that represents a situation in which nothing can happen. The single activity, represented again by two pieces of information, is translated into a net consisting of an input place, a transition and an output place. The semantics of the other operators of MTIPP is similar to that of the corresponding constructs of PEPA, thus they are translated following the same schema introduced in the previous section. The most relevant difference between PEPA and MTIPP concerns the rates they associate with transitions that model synchronization between components. In this case in fact MTIPP assumes that the rate at which the synchronization takes place is equal to the product of the corresponding rates of the individual components as shown in the semantic rules (Figure 4.2) and in the SPN semantics (Figure 4.14). The main advantage of this choice is that the two CTMCs underlying the MTIPP term and the corresponding SPN model are exactly the same.

In MTIPP there is also an explicit recursion operator (*rec*). The presence of this operator does not increase the expressive power of MTIPP with respect to PEPA but in a certain sense it simplifies the translation phase. Each time the word *rec* is encountered we know that we have to "close" the net representation as shown by the backward arc of Figure 4.14.

4.4.3 SPN semantics for MPA

The semantics of MPA is shown in Figure 4.15. Also in this language we have two basic elements, the Nil operator and the single activity. In MPA an activity (a, r) represents r different instances of action a all concurrently enabled. The translation of such activity is obtained by considering r conflicting transitions with name a and rate μ_a and it is shown in Figure 4.15. The net denotes a situation in which action a is performed r times in parallel and the duration until its completion is exponentially distributed with rate $r \cdot \mu_a$. The value $r \cdot \mu_a$ is obtained considering the fact that the minimum among a set of random variables all with exponential distribution is is still exponentially distributed with a rate which is equal to the sum of all the involved rates. The other operators of the language are

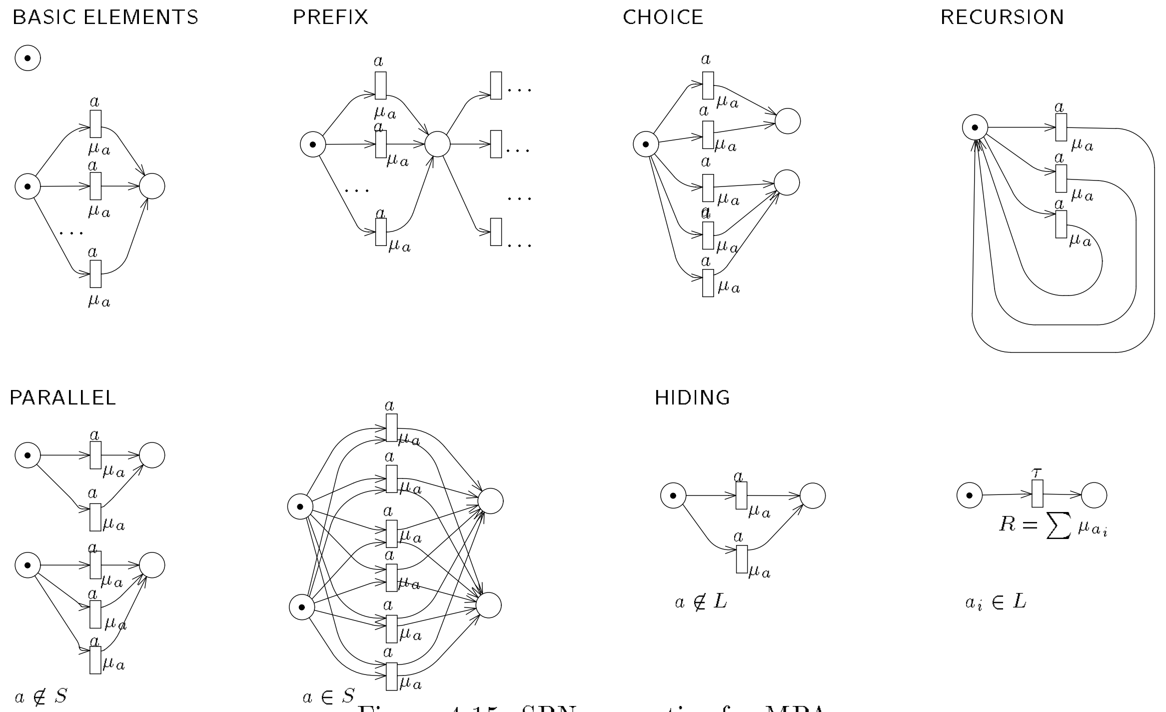


Figure 4.15: SPN semantics for MPA.

translated following the schema already proposed for the other two languages, but in this case each basic element has a more complex structure due to the need to repeat the transitions (see Figure 4.15). Let us consider, for instance, the term

$$P + Q \quad \text{where} \quad P = (a, 2).Nil \quad Q = (a, 3).Nil$$

Action a is invoked 2 times in component P and 3 times in component Q and then it is invoked 2+3 times in $P + Q$ as shown in the translation of the choice in Figure 4.15.

In the parallel composition the components proceed independently on actions $a \notin S$ while they have to synchronize on actions $a \in S$ and the number of invocations of a joint activity equals the product of the number of invocations of both involved activities. Figure 4.15 shows the SPN representation of the term

$$P \|_a Q \quad \text{where} \quad P = (a, 2).Nil \quad Q = (a, 3).Nil$$

The net is obtained superposing the transitions of the nets representing P and Q. Notice that the superposition of transitions leads to a number of resulting transitions equal to Cartesian product of the number of transitions in the starting components, as required by the semantic rule in the language. The total rate of action a in this case is equal to $2 \cdot 3 \cdot \mu_a$.

4.5 Comparison of the languages

The three languages we have considered have very subtle differences that can be precisely outlined analysing their SPN semantics (see the table of Figure 4.16 for some differences between the languages).

One of the main differences is the rate at which cooperation takes place. In PEPA there is the notion of apparent rate, in MTIPP the synchronization is performed with a rate equal to the product of the starting rates and finally in MPA the product of the invocations has been introduced. All these three different choices guarantee that the equivalence relations defined in the languages are also congruences and therefore they are sound from the algebraic point of view. The meaning of the different choices is explained in [7].

Another difference between the languages concerns the presence of the Nil operator. In PEPA there is no Nil operator and it is an inherent assumption that the component structure is static. When a

	PEPA	MTIPP	MPA
Synchronization	Minimum of the apparent rates	Product of rates	Product of invocations
Nil	No	Yes	Yes
Choice	Inside Cooperation for cyclic components	Guarded Choice for Normal Form	Guarded Choice
Rewards	Yes	No	Yes

Figure 4.16: Comparison among PEPA, MTIPP and MPA.

model is constructed there are a certain number of components that are combined using the $\bowtie$ operator. It is assumed that these components remain active throughout the life of the model. This is necessarily the case since without the Nil operator there is no way for a component to "die". For this reason it is not possible to write PEPA expressions whose translation gives SPN models similar to that depicted in Figure 4.17.

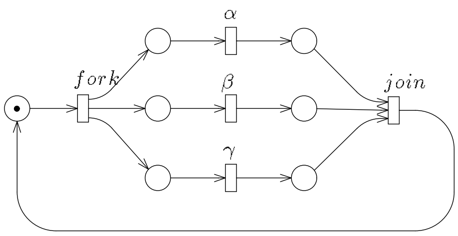


Figure 4.17: SPN model for the Fork & Join structure.

Instead of assuming that there is one process which splits into three and then recombines to form a single process again, we need an alternative view of the system assuming that there are three processes within the system which are initially and finally constrained to act together; they are only free to act independently during the middle phase of their execution. The PEPA system will be specified as

$$\begin{aligned}
P_1 &\stackrel{\text{def}}{=} (fork, \lambda).(\alpha, r_\alpha).(join, \mu).P_1 \\
P_2 &\stackrel{\text{def}}{=} (fork, \lambda).(\beta, r_\beta).(join, \mu).P_2 \\
P_3 &\stackrel{\text{def}}{=} (fork, \lambda).(\gamma, r_\gamma).(join, \mu).P_3 \\
Fork &\stackrel{\text{def}}{=} P_1 \underset{\{fork,join\}}{\bowtie} P_2 \underset{\{fork,join\}}{\bowtie} P_3
\end{aligned}$$

and the corresponding SPN model is shown in Figure 4.18. It can be argued that this does not truly capture the nature of the Fork & Join structure shown in Figure 4.17.

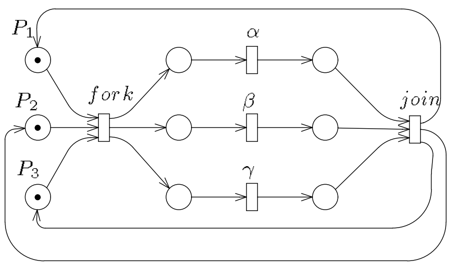


Figure 4.18: SPN model for the Fork & Join expressed with PEPA.

In MTIPP and MPA indeed it is possible to define components that terminate. We can for example specify the Fork & Join system as

$$P_1 \stackrel{\text{def}}{=} (\alpha, r_\alpha).(join, \mu).Nil$$
$$P_2 \stackrel{\text{def}}{=} (\beta, r_\beta).(join, \mu).Fork$$
$$P_3 \stackrel{\text{def}}{=} (\gamma, r_\gamma).(join, \mu).Nil$$
$$Fork \stackrel{\text{def}}{=} (fork, \lambda).(P_1 \underset{\{join\}}{\bowtie} P_2 \underset{\{join\}}{\bowtie} P_3)$$

whose corresponding SPN is shown in Figure 4.19. In this case places Nil_{P_1} and Nil_{P_3}, that model the termination of the two processes P_1 and P_3, are unbounded (i.e. they can contain an infinite number of tokens) and we can think of them as counters that record the number of recursive calls of the Fork component. In this case, of course, the RG of the net (and the underlying CTMC) is infinite, as well as the labelled transition system associated to the language expression.

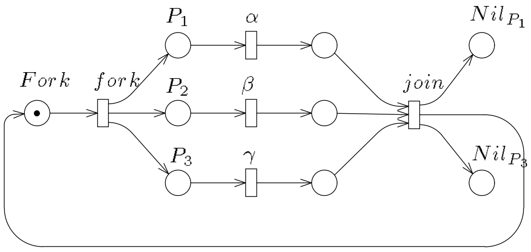


Figure 4.19: SPN model for the Fork & Join expressed with MTIPP and MPA.

Let us now consider the choice operator. In PEPA there is a necessary but not sufficient condition for ergodicity that requires that choices can occur only inside the cooperations. For example the expression $(P \underset{L}{\bowtie} Q)+(R \underset{L}{\bowtie} S)$ does not satisfy the requirement while the expression $(P+Q) \underset{L}{\bowtie} (R+S)$ does. Components of the second kind are called *cyclic* and their underlying CTMCs are ergodic.

The translation of cyclic components is performed starting from the low level components. These low level components contain only the "." and the "+" operators and the resulting nets belong to a particular class of nets, called Stochastic Automata (SA), in which all transitions have one input and one output arc. The composition of the low level components using $\bowtie$ is translated at the net level by superposing transitions and the resulting nets are almost those in the class of Superposed Stochastic Automata (SSA) [2] with different behaviours, the only difference being the peculiar manner of assigning the rate to superposed transitions. The same SSA structure cannot be captured in the other two languages because they do not have similar restrictions imposed on the choice operator.

4.6 Conclusions

The use of SPNs as a common framework to compare several SPAs has been described in Section 4.4. The translation into SPNs elucidates the differences of the considered SPAs allowing therefore a deeper understanding of their semantics.

Moreover, the existence of such a mapping between SPAs and SPNs shows that these two formalisms are very similar and, superficially, we can conclude that they appear the same. The goal of this work is then more ambitious with respect to the translation we have presented. In particular we aim to compare SPAs and SPNs in terms of the facilities they offer to the modeller. For instance, SPNs do not allow a compositional approach for specifying complex systems. Compositionality however has become a strong requirement to deal with the increasing complexity of modern distributed systems and SPAs seem to be a good candidate for the specification and the quantitative analysis of such systems. It is our hope that this and other considerations will help to find a common framework for studying the performance of complex systems.

Acknowledgments I would like to thanks Prof. Herzog and his group for introducing me to the application of process algebra to performance evaluation, and Jane Hillston for the discussion about

this work and for her help in improving its presentation.

Bibliography

[1] Peter Buchholz. On a markovian Process Algebra. Technical Report 500/1994, University of Dortmund, 1994.

[2] S. Donatelli. Superposed stochastic automata: a class of stochastic Petri nets amenable to parallel solution. In *Proc. 4th Intern. Workshop on Petri Nets and Performance Models*, pages 54–63, Melbourne, Australia, December 1991. IEEE-CS Press.

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

[4] H. Hermanns and M. Rettelbach. Markovian Processes go Algebra. Technical Report IMMD VII-10/94, University of Erlangen, March 1994.

[5] Jane Hillston. PEPA: Performance enhanced Process Algebra. Technical Report CSR-24-93, The University of Edinburgh, March 1993.

[6] Jane Hillston. *A Compositional Approach to Performance Modelling.* PhD thesis, 1994.

[7] 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.

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

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

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

[11] M.K. Molloy. *On the Integration of Delay and Throughput Measures in Distributed Processing Models.* PhD thesis, UCLA, Los Angeles, CA, 1981. Ph.D. Thesis.

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