Integrating TwoTowers and GreatSPN through a Compact Net Semantics

Marco Bernardo [1]

Università di Urbino, Centro per l'Appl. delle Scienze e Tecnol. dell'Informazione
Piazza della Repubblica 13, 61029 Urbino, Italy

Nadia Busi

Università di Bologna, Dipartimento di Scienze dell'Informazione
Mura Anteo Zamboni 7, 40127 Bologna, Italy

Marina Ribaudo

Università di Genova, Dipartimento di Informatica e Scienze dell'Informazione
Via Dodecaneso 35, 16146 Genova, Italy

Abstract

Stochastic process algebras (SPAs) and stochastic Petri nets (SPNs) are two well known formal methods for the functional and performance modeling and analysis of computer, communication and software systems. Starting from the mappings from process algebras to Petri nets proposed in the literature to provide a truly concurrent semantic framework to concurrent programming languages, in this paper we define a new SPN semantics for SPAs in order to facilitate the integration and the cross fertilization between the two formalisms. We then prove that our net semantics is correct via a retrievability result. Afterwards, we demonstrate that it improves on the previously proposed net semantics with respect to the size of the resulting SPNs and on the standard interleaving semantics because of the detection of system symmetries. Furthermore, we illustrate its usefulness by showing how to reinterpret at the SPA level the results efficiently obtainable at the SPN level. Finally, we describe the implementation of our net semantics that has been realized to integrate the EMPA_{gr} based software tool TwoTowers with the GSPN based software tool GreatSPN.

[1] Corresponding author. E-mail: `bernardo@sti.uniurb.it`

1 Introduction

The complexity of the modern computer, communication and software systems calls for the development of suitable formal description techniques for their modeling and analysis. In the literature, both process algebras and Petri nets have received much attention, as witnessed by their extensive investigation in the last four decades (see, e.g., [24,27] and the references therein). Process algebras and Petri nets have been studied not only in isolation but also jointly, the purpose being that of providing a truly concurrent semantic framework to concurrent programming languages. This has been accomplished by means of semantic mappings from process algebras to Petri nets [19,16,25,9].

More recently, it has been recognized that the net semantics for process algebras have an applicative relevance, as they facilitate the integration of the two formalisms making possible the exchange of modeling capabilities and analysis techniques between them [25,28,4,3]. This is especially important since the two formalisms are characterized by complementary strengths. On the one hand, process algebras offer a compositional linguistic support to system modeling together with congruences that are useful for compositionally reducing the state space underlying the process terms before analyzing them [24]. On the other hand, Petri nets provide a truly concurrent framework equipped with structural analysis techniques that avoid the construction of the underlying state space [27]. Such considerations are valid not only from the functional point of view, but also from the performance point of view [22,20,8,11]. In particular, they have been taken into account in [3], where a formal approach to the modeling and analysis of complex systems has been proposed, which is based on stochastic process algebras (SPAs) and stochastic Petri nets (SPNs). The objective of the approach is to allow both functional and performance aspects of systems to be considered from the early stages of their design, so that malfunctionings and inefficiency can be detected from the very beginning, thereby avoiding project cost increases that would be caused by their late discovery. In order to be effective, this integrated approach must be implemented through a software tool that should assist the performance modeler. Additionally, such a software tool should made transparent to the performance modeler the transformation of SPA models into SPN models and the reinterpretation at the SPA level of the results of the analysis conducted at the SPN level.

Two fundamental techniques are known in the literature for defining the semantics for a language: the denotational one and the operational one [30]. In this paper we concentrate on the latter technique, i.e. we consider only operational SPN semantics for SPAs. The reason is that operational mappings define abstract machines, whose implementation is straightforward in a software tool, that generate SPN models in an incremental way. The operational mappings from SPAs to SPNs can be evaluated on the basis of their com-

pactness. By compactness we mean the size of the resulting SPNs in terms of number of places and transitions. Obtaining a more compact SPN may lead to a more efficient functional and performance analysis of the modeled system, since the underlying linear equation system on which the analysis is based can be smaller.

In the literature, two different approaches to operational Petri net semantics for process algebras have been defined. In the location oriented approach, the Petri net corresponding to a process term contains a place for every position of the sequential subterms with respect to the static operators (such as hiding, relabeling, and parallel composition). As a consequence, the resulting nets turn out to be 1-safe, i.e. every place contains at most one token at any time. All the information about the syntactical structure of the terms is encoded within the places, so that the relationships among the sequential terms are smoothly preserved. This is exploited to define the net transitions by means of inductive rules similar to those for the interleaving semantics for the process algebra, as shown in [16,25].

In the label oriented approach, instead, each net place corresponds to a sequential subterm independently of its position with respect to the static operators. As a consequence, the instances of the same sequential term occurring in different positions with respect to the static operators can be represented by multiple tokens within the same place. This is possible because the syntactical structure of the process terms with respect to the static operators is no longer fully retained within the net places associated with the sequential subterms. Therefore, the correspondence between the inductive rules for generating the net transitions and the inductive rules of the interleaving semantics for the process algebra no longer holds. In this approach the net transitions are defined by axioms. For instance, in [9] five axioms are employed to derive the net transitions, with the resulting nets including places with inhibitor arcs to model scope extrusion and unguarded choice. As another example, in [4,3] only one axiom is employed, the price to be paid being the introduction of places with inhibitor and contextual arcs in which information about the syntactical structure is kept.

In terms of compactness, the label oriented approach is advantageous since the resulting nets are not necessarily 1-safe and therefore they can be smaller than those obtained with the location oriented approach. Furthermore some terms, for which an infinite net is generated in the location oriented approach, give rise to a finite net in the label oriented approach [10]. Given the applicative relevance of the integration of SPAs and SPNs, the label oriented approach seems to be the approach of choice to realize such an integration due to its compactness.

The weakness of this approach is that additional places with inhibitor and

contextual arcs are employed to handle static operators. Actually, in [4,3] it has been proved that these places with the related arcs can be removed a posteriori; this makes the nets simpler but requires an additional computational step. The purpose of this paper [2] is to further elaborate on the label oriented SPN semantics of [4,3] by avoiding the introduction of additional places with inhibitor and contextual arcs altogether. The novel idea to achieve that is to suitably decorate the actions within the sequential terms associated with the net places in order to keep track of the occurrences of the static operators, without falling in the excess of information of the location oriented approach.

The new label oriented net semantics is developed by focusing on EMPA_{gr} (Extended Markovian Process Algebra with generative-reactive synchronizations) [8] and on GSPNs (Generalized Stochastic Petri Nets) [1]. The reason is twofold. First, the structure of EMPA_{gr} actions has been strongly influenced by the structure of GSPN transitions, so the mapping from EMPA_{gr} terms to GSPNs should be natural. Second, analysis tools – TwoTowers [3] and GreatSPN [13] – have been developed for both formalisms, so that the integration of the two formalisms can be realized in practice. It is worth noting that, since the semantics is intended to be implemented in a software tool that should assist the performance modeler, all the details of the transformation are completely transparent.

This paper is organized as follows. In Sect. 2 we recall EMPA_{gr} and GSPNs. In Sect. 3 we define an improved, label oriented GSPN semantics for EMPA_{gr} and we prove its correctness by showing that the interleaving semantics for EMPA_{gr} can be retrieved from it. In Sect. 4 we discuss the compactness of the generated GSPNs both at the net level, with respect to the previously proposed net semantics, and at the state space level, by showing that some system symmetries are captured; we also exhibit a sufficient condition for the finiteness of the generated GSPNs and we investigate the reinterpretability of the analysis results. In Sect. 5 we describe how the novel label oriented GSPN semantics for EMPA_{gr} is implemented to integrate TwoTowers with GreatSPN. In Sect. 6 we present an example in which we exploit the features of the net semantics to measure the performance of a random polling system when varying its parameters. Finally, Sect. 7 concludes the paper with some remarks on related and future work.

2 Background

In this section we recall some concepts about the two specific instances of SPAs and SPNs, respectively, we shall consider in the rest of the paper. In Sect. 2.1

[2] Full and revised version of [5,6].

we introduce EMPA_{gr} while in Sect. 2.2 we present GSPNs. The reader who is already familiar with EMPA_{gr} or GSPNs can safely skip the related section.

2.1 Extended Markovian Process Algebra

EMPA_{gr} [8] is a process algebra that allows both functional and performance aspects of complex systems to be modeled. [3] The main ingredients of EMPA_{gr} are the actions and the algebraic operators. Each action is composed of a type and a rate. Based on rates, an action is classified as exponentially timed if its rate is a positive real number, immediate if its rate is infinite (denoted by $\infty_{l,w}$ with priority level l and weight w), and passive if its rate is left unspecified (denoted by $*_w$ with reactive weight w [4]).

Definition 2.1 Let *AType* be the set of *action types*, including the invisible type τ, and $ARate = \mathbb{R}_+ \cup \{\infty_{l,w} \mid l \in \mathbb{N}_+ \wedge w \in \mathbb{R}_+\} \cup \{*_w \mid w \in \mathbb{R}_+\}$ be the set of *action rates*. We use a to range over *AType*, λ to range over exponentially timed rates, $\bar{\lambda}$ to range over nonpassive rates, and $\tilde{\lambda}$ to range over *ARate*. The set of *actions* is defined by

$$Act = AType \times ARate$$

We define function priority level $PL : ARate \longrightarrow \mathbb{Z}$ by:

$$PL(*_w) = -1$$

$$PL(\lambda) = 0$$

$$PL(\infty_{l,w}) = l$$

The real number summation is extended to rates of the same priority level as follows:

$$*_{w_1} + *_{w_2} = *_{w_1+w_2}$$

$$\infty_{l,w_1} + \infty_{l,w_2} = \infty_{l,w_1+w_2}$$

The multiplication of a rate by a real number is defined as follows:

$$*_w \cdot p = *_{w \cdot p}$$

$$\infty_{l,w} \cdot p = \infty_{l,w \cdot p}$$ ■

Definition 2.2 Let *Const* be a set of *constants* ranged over by A and let $ATRFun = \{\varphi : AType \longrightarrow AType \mid \varphi^{-1}(\tau) = \{\tau\}\}$ be a set of *action type relabeling functions* ranged over by φ. The set $\mathcal{L}$ of *process terms* of EMPA_{gr}

[3] EMPA_{gr} is an extension of EMPA [3] in which passive actions are given priority levels and weights which are interpreted reactively, i.e. only among passive actions of the same type.

[4] We slightly depart from the definition of the syntax of the actions given in [8] as no reactive priority level is associated with passive actions. The reason will be explained in Appendix B.

is generated by the following syntax

$$E ::= \sum_{i \in I} <a_i, \tilde{\lambda}_i>.E_i \mid E/L \mid E[\varphi] \mid E \parallel_S E \mid A$$

where I is a finite set of indices, $\sum_{i \in I} <a_i, \tilde{\lambda}_i>.E_i$ is denoted by $\underline{0}$ whenever $I = \emptyset$, $L, S \subseteq AType - \{\tau\}$, and A is equipped with a constant defining equation of the form $A \triangleq E$. We denote by $sort(E)$ the set of action types occurring in E. Moreover, we denote by $\mathcal{G}$ the set of closed and guarded terms of $\mathcal{L}$. ■

The *guarded alternative composition operator* "$\sum_{i \in I} <a_i, \tilde{\lambda}_i>._$" expresses a choice between several alternative behaviors whose first actions are made explicit. Term $\sum_{i \in I} <a_i, \tilde{\lambda}_i>.E_i$ can execute an action with type a_i and rate $\tilde{\lambda}_i$ and then behaves as term E_i. In the case of exponentially timed actions, the choice is solved according to the *race policy*: the action sampling the least duration succeeds. In the case of immediate actions, they take precedence over exponentially timed actions and the choice is solved according to the *preselection policy*: the immediate actions having the highest priority level are singled out, then each of them is given an execution probability proportional to its weight. In the case of passive actions, the choice is solved according to the *reactive preselection policy*: the choice among passive actions of different types is nondeterministic, while the choice among passive actions of the same type is determined by giving each action an execution probability proportional to its weight.

The *functional abstraction operator* "$\_/L$" abstracts from the type of the actions. Term E/L behaves as term E except that the type a of each executed action is turned into τ whenever $a \in L$.

The *functional relabeling operator* "$\_[\varphi]$" changes the type of the actions. Term $E[\varphi]$ behaves as term E except that the type a of each executed action becomes $\varphi(a)$.

The *parallel composition operator* "$\_ \parallel_S \_$" expresses the concurrent execution of two terms. Term $E_1 \parallel_S E_2$ asynchronously executes actions of E_1 or E_2 not belonging to S and synchronously executes actions of E_1 and E_2 belonging to S according to the two following synchronization disciplines. The synchronization discipline on action types establishes that two actions can synchronize if and only if they have the same visible type in S, which becomes the resulting type. The synchronization discipline on action rates is the *generative master-reactive slaves mechanism*. In the case of synchronization of a nonpassive action of type a having rate $\tilde{\lambda}$ executed by E_1 (E_2) with a passive action of type a having rate $*_w$ executed by E_2 (E_1), the resulting nonpassive action of type a has a rate/weight given by the original rate/weight multiplied by the probability that E_2 (E_1) chooses the passive action at hand among its passive actions of type a. Instead, in case of synchronization of two passive actions of type a having rate $*_{w_1}$ and $*_{w_2}$ executed by E_1 and E_2, respectively, the

resulting passive action of type a has a weight given by the probability that E_1 and E_2 independently choose the two actions, multiplied by a normalization factor equal to the overall weight of the passive actions of type a executable by E_1 and E_2. A synchronization between two nonpassive actions of the same visible type in S is not allowed.

The functional abstraction operator, the functional relabeling operator, and the parallel composition operator are said to be *static* operators. A term is said to be *sequential* if its outermost operator is the guarded alternative composition.

The operational semantics for EMPA_{gr} is the least labeled transition system (LTS for short) satisfying the inference rules of Table 1. We consider the operational rules as generating a multiset of transitions, where a transition has arity m if and only if it can be derived in m possible ways from the operational rules. As far as multisets are concerned, in the following we use "$\{\!|$" and "$|\!\}$" as brackets for multisets, "$\_ \oplus \_$" ("$\_ \ominus \_$") to denote multiset union (difference), and $\mathcal{M}(S)$ ($\mathcal{P}(S)$) to denote the collection of multisets over (subsets of) set S. In the table, $W_a(E) = \sum\{\!| \, w \mid \exists E'.\, E \xrightarrow{a,*_w} E' \, |\!\}$.

Definition 2.3 The *integrated interleaving semantics* of $E \in \mathcal{G}$ is the LTS

$$\mathcal{I}[\![E]\!] = (\mathcal{G}_E, Act, \longrightarrow_E, E)$$

where $\mathcal{G}_E$ is the least subset of $\mathcal{G}$ such that:

- $E \in \mathcal{G}_E$;
- if $E_1 \in \mathcal{G}_E$ and $E_1 \xrightarrow{a,\tilde{\lambda}} E_2$, then $E_2 \in \mathcal{G}_E$;

and $\longrightarrow_E$ is the restriction of $\longrightarrow$ to the transitions between terms in $\mathcal{G}_E$. We say that $E \in \mathcal{G}$ is *performance closed* if and only if $\mathcal{I}[\![E]\!]$ does not contain passive transitions. ■

We conclude by recalling that from $\mathcal{I}[\![E]\!]$ two projected semantic models can be obtained by essentially dropping action rates or action types, respectively. Before applying such a transformation to $\mathcal{I}[\![E]\!]$, lower priority nonpassive transitions are pruned because E is no longer to be composed with other terms as it describes the whole system we are interested in; in other words, E represents a closed system. The *functional semantics* $\mathcal{F}[\![E]\!]$ is a LTS whose transitions are decorated with action types only. The *Markovian semantics* $\mathcal{M}[\![E]\!]$ is instead a continuous or discrete time Markov chain, which is well defined only if E is performance closed.

$$\textbf{(GAC)}\quad \textstyle\sum_{i\in I} <a_i, \tilde{\lambda}_i>.E_i \xrightarrow{a_i,\tilde{\lambda}_i} E_i \quad i \in I$$

$$\textbf{(FA1)}\ \frac{E \xrightarrow{a,\tilde{\lambda}} E'}{E/L \xrightarrow{a,\tilde{\lambda}} E'/L}\ a \notin L \qquad \textbf{(FA2)}\ \frac{E \xrightarrow{a,\tilde{\lambda}} E'}{E/L \xrightarrow{\tau,\tilde{\lambda}} E'/L}\ a \in L$$

$$\textbf{(FR)}\ \frac{E \xrightarrow{a,\tilde{\lambda}} E'}{E[\varphi] \xrightarrow{\varphi(a),\tilde{\lambda}} E'[\varphi]}$$

$$\textbf{(PC1}_l\textbf{)}\ \frac{E_1 \xrightarrow{a,\tilde{\lambda}} E_1'}{E_1 \parallel_S E_2 \xrightarrow{a,\tilde{\lambda}} E_1' \parallel_S E_2}\ a \notin S$$

$$\textbf{(PC1}_r\textbf{)}\ \frac{E_2 \xrightarrow{a,\tilde{\lambda}} E_2'}{E_1 \parallel_S E_2 \xrightarrow{a,\tilde{\lambda}} E_1 \parallel_S E_2'}\ a \notin S$$

$$\textbf{(PC2}_l\textbf{)}\ \frac{E_1 \xrightarrow{a,\bar{\lambda}} E_1' \qquad E_2 \xrightarrow{a,*_w} E_2'}{E_1 \parallel_S E_2 \xrightarrow{a,\tilde{\lambda}\cdot\frac{w}{W_a(E_2)}} E_1' \parallel_S E_2'}\ a \in S$$

$$\textbf{(PC2}_r\textbf{)}\ \frac{E_1 \xrightarrow{a,*_w} E_1' \qquad E_2 \xrightarrow{a,\bar{\lambda}} E_2'}{E_1 \parallel_S E_2 \xrightarrow{a,\tilde{\lambda}\cdot\frac{w}{W_a(E_1)}} E_1' \parallel_S E_2'}\ a \in S$$

$$\textbf{(PC3)}\ \frac{E_1 \xrightarrow{a,*_{w_1}} E_1' \qquad E_2 \xrightarrow{a,*_{w_2}} E_2'}{E_1 \parallel_S E_2 \xrightarrow{a,*_{p\cdot W}} E_1' \parallel_S E_2'}\ a \in S$$

where: $\quad p = \frac{w_1}{W_a(E_1)} \cdot \frac{w_2}{W_a(E_2)} \qquad W = W_a(E_1) + W_a(E_2)$

$$\textbf{(CI)}\ \frac{E \xrightarrow{a,\tilde{\lambda}} E'}{A \xrightarrow{a,\tilde{\lambda}} E'}\ A \triangleq E$$

Table 1
EMPA$_{\mathrm{gr}}$ operational semantics

GSPNs [1] are an extension of classical Petri nets which is suited to performance evaluation purposes. A Petri net [27] is a bipartite graph whose classes of nodes are called places (representing system conditions and resources) and transitions (representing system activities), respectively. Unlike LTSs, Petri nets support a distributed notion of state given by the marking of places with tokens, which is formalized through a multiset.

GSPNs are Petri nets that comprise inhibitor arcs as well as exponentially timed transitions with marking dependent rates and immediate transitions equipped with priorities and marking dependent weights. We introduce below a slightly different definition of GSPN where inhibitor arcs are removed and passive transitions are added. This is because, when defining the new net semantics for EMPA_{gr}, inhibitor arcs will not be necessary whereas passive actions will be mapped to passive transitions. Passive transitions will not be generated in the case of performance closed terms. We assume an infinite server semantics for transition firing.

Definition 2.4 A *GSPN* is a tuple

$$(P, AType \times ARate^{\mathcal{M}(P)}, T, M_0)$$

where:

- P is a set whose elements are called *places*.
- $T \subseteq \mathcal{M}(P) \times (AType \times ARate^{\mathcal{M}(P)}) \times \mathcal{M}(P)$ is a set whose elements are called *transitions*.
- $M_0 \in \mathcal{M}(P)$ is called the *initial marking*.

The set $\mathcal{M}(P)$ of possible markings will be ranged over by M. ■

In the graphical representation of a GSPN, places are drawn as circles containing black dots called tokens (which describe the current marking of the net) while transitions are drawn as boxes if exponentially timed, bars if immediate, or black boxes if passive, with the appropriate labels. Each transition t can be written as a function of the current marking M_{curr} which returns the triple $({}^\bullet t, <a, \tilde{\lambda}(M_{\text{curr}})>, t^\bullet)$ where ${}^\bullet t$ is the weighted preset of t (places where tokens are consumed) and $t^\bullet$ is the weighted postset of t (places where tokens are produced). Given a transition t, we draw an arrow headed arc from each place in ${}^\bullet t$ to t as well as from t to each place in $t^\bullet$, where each arc is labeled with the multiplicity of the related place (one is the default value for arc labels), and we denote by $PL(t)$ the priority level of the action associated with it.

Definition 2.5 Let $N = (P, AType \times ARate^{\mathcal{M}(P)}, T, M_0)$ be a GSPN.

- Transition $t \in T$ *has concession* at marking $M \in \mathcal{M}(P)$ if and only if ${}^\bullet t \subseteq M$. We denote by $ET(M)$ the set of transitions enabled at marking M.
- Transition $t \in ET(M)$, where $M \in \mathcal{M}(P)$, can *fire* if and only if $PL(t) \in \{-1, \max\{PL(t') \mid t' \in ET(M)\}\}$. The firing of transition $t \in ET(M)$ produces marking $M' = (M \ominus {}^\bullet t) \oplus t^\bullet$. This is written $M\,[a, \tilde{\lambda}(M)\rangle\, M'$.
- The *reachability graph (or interleaving marking graph)* of N is the LTS
$$\mathcal{RG}[\![N]\!] = (RS(M_0), Act, [\rangle, M_0)$$
where $RS(M_0)$ is the least subset of $\mathcal{M}(P)$ such that:
· $M_0 \in RS(M_0)$;
· if $M_1 \in RS(M_0)$ and $M_1\,[a, \tilde{\lambda}(M_1)\rangle\, M_2$, then $M_2 \in RS(M_0)$. ■

3 A New Label Oriented Net Semantics

In this section we present a new label oriented GSPN semantics for EMPA_{gr}. As we have seen in the introduction, the purpose of this semantics is to integrate the two formalisms, so that the performance modeler can profitably exploit the corresponding analysis techniques. This semantics is intended to be implemented in a software tool that should assist the performance modeler, so that all the details of the transformation are made transparent.

The semantics is defined in three steps. In Sect. 3.1 we introduce a suitable term oriented syntax for places and we show how to map terms onto the places themselves. In Sect. 3.2 we provide a technique by means of which all the transitions can be generated. In Sect. 3.3 we exhibit the definition of the GSPN associated with a given EMPA_{gr} term. In Sect. 3.4 we show that the new net semantics satisfies the retrievability principle. For the sake of readability, the definition of the semantics is presented pictorially and is guided by a small running example. The reader interested in the formal definitions is referred to Appendix A and B.

3.1 Net Places

The first step consists of introducing a suitable set of places as well as defining a function which decomposes the terms into their sequential components and maps them onto the places.

Since the sequential terms are represented by guarded alternative compositions, these are used to formalize the places. Here the novel idea to keep track of the static operators is to suitably decorate the actions within the net places, thus avoiding the introduction of inhibitor and contextual arcs as well as the

excess of information of the location oriented approach. Each action type a becomes

$$a^{\theta,R,\sigma}$$

Decoration $\theta \in \{\tau, \varepsilon\}$ is used to remember to hide certain actions. Decoration R is a set of conflicts employed to avoid clashes within the scope of a functional relabeling operator. Its presence is required by the fact that the actions will be directly relabeled within the net places. We define $Conf$ as an infinite set of conflicts (ranged over by r) on which the following operation is defined to introduce the notion of complementary conflicts:

$$^{-} : Conf \longrightarrow Conf, \bar{\bar{r}} = r$$

Finally, decoration σ is a string of combinators used to correctly handle the synchronizations. We define $Comb$ as an infinite set of combinators (ranged over by k) and we consider the set $Comb^*$ of combinator strings (ranged over by σ), on which the two following operations are defined to introduce the notions of complementary combinators and combinator reduction:

$$^{-} : Comb \longrightarrow Comb, \bar{\bar{k}} = k$$

$$\odot : Comb^* \times Comb^* \not\longrightarrow Comb^*, \sigma k \odot \sigma \bar{k} = \sigma$$

The reduction operator is extended to a multiset m over $Comb^*$ as follows:

$$\bigodot m = \begin{cases} \bigodot(\{\!|\, \sigma \,|\!\} \oplus m') & \text{if } \exists \sigma, k, m'.\, m = \{\!|\, \sigma k, \sigma \bar{k} \,|\!\} \oplus m' \\ m & \text{otherwise} \end{cases}$$

Definition 3.1 The set of places is defined by

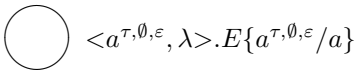
$$\begin{aligned} \mathcal{V} = \{ \textstyle\sum_{i \in I} <a_i^{\theta_i, R_i, \sigma_i}, \tilde{\lambda}_i>.E_i \mid\ & \theta_i \in \{\tau, \varepsilon\} \wedge \\ & R_i \in \mathcal{P}(Conf) \wedge \\ & \sigma_i \in Comb^* \wedge \\ & E_i \in \mathcal{G}' \} \end{aligned}$$

where the summation is associative and commutative, $\mathcal{G}'$ is obtained from $\mathcal{G}$ by replacing $AType$ with $AType' = AType \times \{\tau, \varepsilon\} \times \mathcal{P}(Conf) \times Comb^*$ with (a, θ, R, σ) denoted by $a^{\theta,R,\sigma}$, $a^{\varepsilon,\emptyset,\varepsilon}$ denoted by a, and $AType' \times ARate$ denoted by Act'. $\mathcal{V}$ and $\mathcal{M}(\mathcal{V})$ will be ranged over by V and Q, respectively. ■

Now we pictorially show how terms are decomposed into sequential subterms and then mapped onto places having the syntax above. The reader interested in the formal definition of the decomposition function dec is referred to Appendix A.

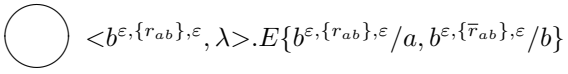
In the case of the guarded alternative composition, $dec(\sum_{i \in I} <a_i, \tilde{\lambda}_i>.E_i)$ is given by

In the case of the functional abstraction, $dec(<a, \lambda>.E)/\{a\})$ is given by



where syntactical substitution $\{a^{\tau,\emptyset,\varepsilon}/a\}$ applied to term E gives rise to a variant of E where every occurrence of a is replaced by $a^{\tau,\emptyset,\varepsilon}$. Note that the first decoration of action type a, i.e. τ, provides the information that a is hidden and this will be subsequently used when generating the net transitions.

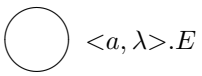
In the case of the functional relabeling, $dec(<a, \lambda>.E)[\varphi])$ where $\varphi(a) = \varphi(b) = b$ is given by



Observe that the second decoration of action type b, i.e. r_{ab}, provides the information that such an action originally had type a. This will be used at transition generation time to avoid the creation of erroneous transitions, such as the one that would result from the synchronization of action $<a, \lambda>$ with action $<b, *_w>$ within the scope of the same functional relabeling as above. This is avoided by rewriting such an action type b as $b^{\varepsilon,\{\bar{r}_{ab}\},\varepsilon}$ with $\bar{r}_{ab}$ complementary to r_{ab}, hence in conflict with it. A synchronization among a set of decorated actions of the same type can occur only if no conflict exists among their second decorations.

In the case of the parallel composition, we distinguish between two limiting scenarios. In the case of full synchronization, $dec(<a, \lambda>.E_1 \parallel_{\{a\}} <a, *_w>.E_2)$ is given by

Note that the two action types are given as third decoration two complementary combinators k and $\bar{k}$, respectively, that reduce to ε. This information will be used later to derive a synchronization transition. In the case of full parallelism of replicas, $dec(<a, \lambda>.E \parallel_\emptyset <a, \lambda>.E)$ is given by



where the place above has multiplicity two. Having just one place captures the symmetry of the term at hand and would have not been possible with the location oriented approach.

Example 3.2 Let us a consider a simple system composed of two identical processes accessing the same memory module in mutual exclusion. This system can be described in EMPA_{gr} as follows:

$$
\begin{aligned}
Sys &\triangleq ((Proc \parallel_{\emptyset} Proc) \parallel_{\{acq,rel\}} Mem)/\{comp\} \\
Proc &\triangleq <comp, \lambda>.<acq, \mu>.<use, \delta>.<rel, \gamma>.Proc \\
Mem &\triangleq <acq, *_w>.<rel, *_w>.Mem
\end{aligned}
$$

where *comp* denotes a local computation (which is therefore hidden) while *acq*, *use*, and *rel* denote the acquisition, use, and release of the memory module, respectively.

The decomposition of *Sys* is computed as follows:

$$
\begin{aligned}
dec(Sys) &= dec(((Proc \parallel_{\emptyset} Proc) \parallel_{\{acq,rel\}} Mem)/\{comp\}) \\
&= dec(((Proc \parallel_{\emptyset} Proc) \parallel_{\{acq,rel\}} Mem)\{comp^{\tau,\emptyset,\varepsilon}/comp\}) \\
&= dec((Proc \parallel_{\emptyset} Proc)\{comp^{\tau,\emptyset,\varepsilon}/comp\} \parallel_{\{acq,rel\}} \\
&\qquad Mem\{comp^{\tau,\emptyset,\varepsilon}/comp\}) \\
&= dec((Proc\{comp^{\tau,\emptyset,\varepsilon}/comp\} \parallel_{\emptyset} Proc\{comp^{\tau,\emptyset,\varepsilon}/comp\}) \parallel_{\{acq,rel\}} \\
&\qquad <acq, *_w>.<rel, *_w>.Mem) \\
&= dec((<comp^{\tau,\emptyset,\varepsilon}, \lambda>.<acq, \mu>.<use, \delta>.<rel, \gamma>.Proc' \parallel_{\emptyset} \\
&\qquad\quad <comp^{\tau,\emptyset,\varepsilon}, \lambda>.<acq, \mu>.<use, \delta>.<rel, \gamma>.Proc') \parallel_{\{acq,rel\}} \\
&\qquad <acq, *_w>.<rel, *_w>.Mem) \\
&= dec((<comp^{\tau,\emptyset,\varepsilon}, \lambda>.<acq, \mu>.<use, \delta>.<rel, \gamma>.Proc' \parallel_{\emptyset} \\
&\qquad\quad <comp^{\tau,\emptyset,\varepsilon}, \lambda>.<acq, \mu>.<use, \delta>.<rel, \gamma>.Proc') \\
&\qquad\quad \{acq^{\varepsilon,\emptyset,k_1}/acq, rel^{\varepsilon,\emptyset,k_2}/rel\}) \oplus \\
&\quad dec((<acq, *_w>.<rel, *_w>.Mem)\{acq^{\varepsilon,\emptyset,\bar{k}_1}/acq, rel^{\varepsilon,\emptyset,\bar{k}_2}/rel\}) \\
&= dec(<comp^{\tau,\emptyset,\varepsilon}, \lambda>.<acq^{\varepsilon,\emptyset,k_1}, \mu>.<use, \delta>.<rel^{\varepsilon,\emptyset,k_2}, \gamma>.Proc'' \parallel_{\emptyset} \\
&\qquad <comp^{\tau,\emptyset,\varepsilon}, \lambda>.<acq^{\varepsilon,\emptyset,k_1}, \mu>.<use, \delta>.<rel^{\varepsilon,\emptyset,k_2}, \gamma>.Proc'') \oplus \\
&\quad dec(<acq^{\varepsilon,\emptyset,\bar{k}_1}, *_w>.<rel^{\varepsilon,\emptyset,\bar{k}_2}, *_w>.Mem') \\
&= dec(<comp^{\tau,\emptyset,\varepsilon}, \lambda>.<acq^{\varepsilon,\emptyset,k_1}, \mu>.<use, \delta>.<rel^{\varepsilon,\emptyset,k_2}, \gamma>.Proc'') \oplus \\
&\quad dec(<comp^{\tau,\emptyset,\varepsilon}, \lambda>.<acq^{\varepsilon,\emptyset,k_1}, \mu>.<use, \delta>.<rel^{\varepsilon,\emptyset,k_2}, \gamma>.Proc'') \oplus \\
&\quad \{\!| <acq^{\varepsilon,\emptyset,\bar{k}_1}, *_w>.<rel^{\varepsilon,\emptyset,\bar{k}_2}, *_w>.Mem' |\!\} \\
\\
&= \{\!| <comp^{\tau,\emptyset,\varepsilon}, \lambda>.<acq^{\varepsilon,\emptyset,k_1}, \mu>.<use, \delta>.<rel^{\varepsilon,\emptyset,k_2}, \gamma>.Proc'', \\
&\quad <comp^{\tau,\emptyset,\varepsilon}, \lambda>.<acq^{\varepsilon,\emptyset,k_1}, \mu>.<use, \delta>.<rel^{\varepsilon,\emptyset,k_2}, \gamma>.Proc'', \\
&\quad <acq^{\varepsilon,\emptyset,\bar{k}_1}, *_w>.<rel^{\varepsilon,\emptyset,\bar{k}_2}, *_w>.Mem' |\!\}
\end{aligned}
$$

where

$$Proc' \triangleq <comp^{\tau,\emptyset,\varepsilon}, \lambda>.<acq, \mu>.<use, \delta>.<rel, \gamma>.Proc'$$

$$Proc'' \triangleq <comp^{\tau,\emptyset,\varepsilon}, \lambda>.<acq^{\varepsilon,\emptyset,k_1}, \mu>.<use, \delta>.<rel^{\varepsilon,\emptyset,k_2}, \gamma>.Proc''$$

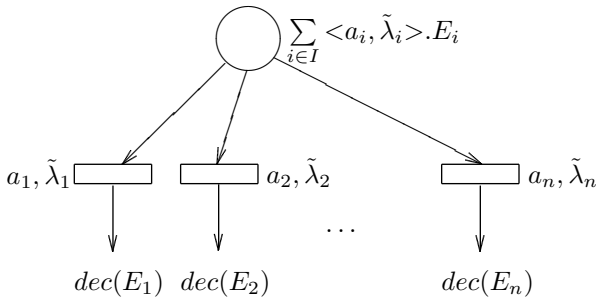
$$Mem' \triangleq <acq^{\varepsilon,\emptyset,\bar{k}_1}, *_w>.<rel^{\varepsilon,\emptyset,\bar{k}_2}, *_w>.Mem'$$

Note that the decomposition comprises two places, the former of which has multiplicity two (it appears twice in the multiset). This correctly captures the symmetry in the system due to the presence of two independent replicas of *Proc*. ■

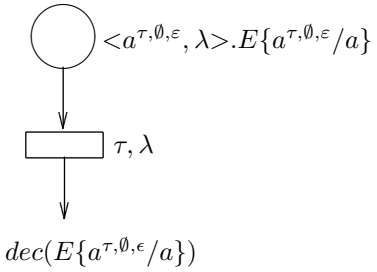
3.2 Net Transitions

The second step consists of connecting by transitions the places arising from the decomposition of the terms. This is achieved by exploiting the action decorations introduced in the first step. Again, we pictorially show how the transitions are generated. The reader interested in the formal definition of the transition generation is referred to Appendix B.

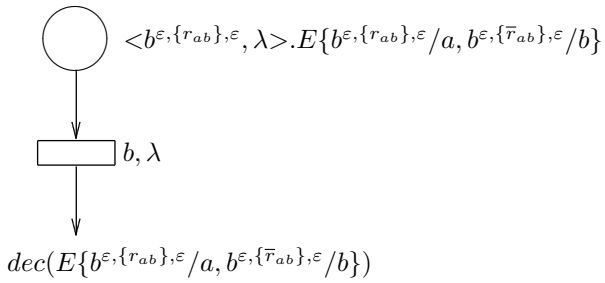
In the case of the guarded alternative composition, $\sum_{i \in I} <a_i, \tilde{\lambda}_i>.E_i$ has the following transitions



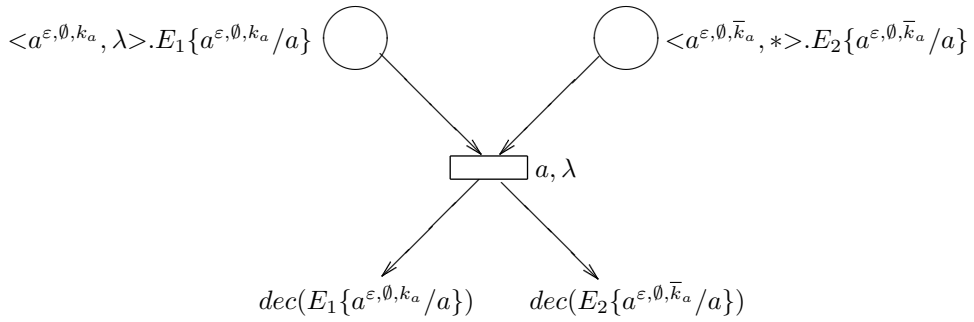
In the case of the functional abstraction, $(<a, \lambda>.E)/\{a\}$ has the following τ transition



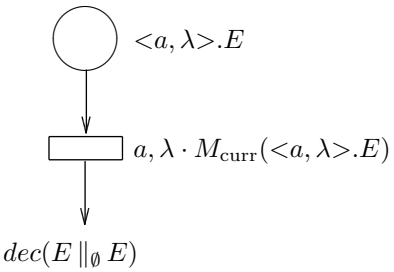
In the case of the functional relabeling, $(<a, \lambda>.E)[\varphi]$ where $\varphi(a) = \varphi(b) = b$ has the following b transition



In the case of full synchronization, $<a, \lambda>.E_1 \parallel_{\{a\}} <a, *_w>.E_2$ has the following a transition



In the case of full parallelism of replicas, $<a, \lambda>.E \parallel_\emptyset <a, \lambda>.E$ has the following a transition



Notice that the transition has a rate which depends on the current marking M_{curr}. Initially, the input place will contain two tokens representing the two identical subterms of the parallel composition, hence the rate will be $2 \cdot \lambda$. After one firing of the transition, the input place will contain only one token, hence the rate will decrease to λ, as expected.

Example 3.3 Let us a consider again the simple system of Ex. 3.2. Starting from the two places

$$V_1 = <comp^{\tau,\emptyset,\varepsilon}, \lambda>.<acq^{\varepsilon,\emptyset,k_1}, \mu>.<use, \delta>.<rel^{\varepsilon,\emptyset,k_2}, \gamma>.Proc''$$

$$V_2 = <acq^{\varepsilon,\emptyset,\bar{k}_1}, *_w>.<rel^{\varepsilon,\emptyset,\bar{k}_2}, *_w>.Mem'$$

forming its decomposition, a single transition labeled with τ, λ can be generated whose preset is V_1 and whose postset is given by

$$V_3 = dec(<acq^{\varepsilon,\emptyset,k_1}, \mu>.<use, \delta>.<rel^{\varepsilon,\emptyset,k_2}, \gamma>.Proc'')$$
$$= \{\!| <acq^{\varepsilon,\emptyset,k_1}, \mu>.<use, \delta>.<rel^{\varepsilon,\emptyset,k_2}, \gamma>.Proc'' |\!\}$$

In contrast, no transition can be generated having as preset the place V_2 because combinator k_1 does not reduce to the empty string. Such a reduction is instead possible when considering V_3 and V_2 together. They constitute the preset of a transition labeled with acq, μ whose postset is composed of places

$$V_4 = dec(<use, \delta>.<rel^{\varepsilon,\emptyset,k_2}, \gamma>.Proc'') = \{\!| <use, \delta>.<rel^{\varepsilon,\emptyset,k_2}, \gamma>.Proc'' |\!\}$$
$$V_5 = dec(<rel^{\varepsilon,\emptyset,\bar{k}_2}, *_w>.Mem') = \{\!| <rel^{\varepsilon,\emptyset,\bar{k}_2}, *_w>.Mem' |\!\}$$

By proceeding in this way, we can generate all the transitions in the net corresponding to *Sys*. Along with the transitions, an additional place will be generated:

$$V_6 = dec(<rel^{\varepsilon,\emptyset,k_2}, \gamma>.Proc'') = \{\!| <rel^{\varepsilon,\emptyset,k_2}, \gamma>.Proc'' |\!\}$$ ■

3.3 Net Construction

Once all the places and all the transitions have been generated in the first two steps, a GSPN can be constructed for the term under consideration.

Definition 3.4 The *integrated net semantics* of $E \in \mathcal{G}$ is the GSPN

$$\mathcal{N}[\![E]\!] = (P_{E,\mathcal{N}}, AType \times ARate^{\mathcal{M}(P_{E,\mathcal{N}})}, \longrightarrow_{E,\mathcal{N}}, dec(E))$$

where, recalling that $\longrightarrow_{\mathcal{N}}$ is formally defined in Appendix B, $P_{E,\mathcal{N}}$ is the least subset of $\mathcal{V}$ such that:

- $dec(E) \subseteq P_{E,\mathcal{N}}$;
- if $Q_1 \subseteq P_{E,\mathcal{N}}$ and $Q_1 \xrightarrow{norm(<a,\tilde{\lambda}>,E,f_1,f_2)}_{\mathcal{N}} Q_2$, then $Q_2 \subseteq P_{E,\mathcal{N}}$;

and $\longrightarrow_{E,\mathcal{N}}$ is the restriction of $\longrightarrow_{\mathcal{N}}$ to the transitions whose preset and postset are entirely in $P_{E,\mathcal{N}}$. ■

Example 3.5 Let us a consider again the simple system of Ex. 3.2 and 3.3. The corresponding GSPN is shown in Fig. 1, where the names of the places are as indicated in Ex. 3.3. ■

3.4 Retrievability Principle

In order to prove the correctness of the net semantics for EMPA_{gr} we have just illustrated, we follow the approach of [25] by establishing a retrievability result. Such a result shows that the interleaving semantics for EMPA_{gr} is retrievable

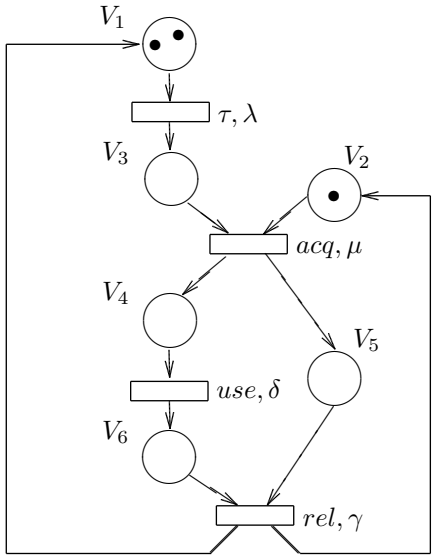


Fig. 1. $\mathcal{N}[\![Sys]\!]$

from the net semantics for EMPA_{gr}, which means that the net semantics and the interleaving semantics of a given term represent the same system both from the functional and the performance point of view. In particular, from the performance viewpoint it turns out that an EMPA_{gr} term and its corresponding GSPN give rise to two Markov chains that are lumping equivalent, i.e. they can be transformed via lumping into the same minimal Markov chain. This entails that, given a performance measure expressed through a reward structure, the value of the measure is the same for the two Markov chains.

The retrievability result is established by proving for all $E \in \mathcal{G}$ that $\mathcal{RG}[\![\mathcal{N}[\![E]\!]]\!]$ is Markovian bisimilar [8] to $\mathcal{I}[\![E]\!]$. We first observe that these two LTSs are not isomorphic in general. For instance, Fig. 2 shows $\mathcal{RG}[\![\mathcal{N}[\![Sys]\!]]\!]$ and $\mathcal{I}[\![Sys]\!]$. As can be noted, there is no 1-1 correspondence between the states in the two LTSs, but a Markovian bisimulation can be recognized in which several states in $\mathcal{I}[\![Sys]\!]$ correspond to a single state in $\mathcal{RG}[\![\mathcal{N}[\![Sys]\!]]\!]$. This correspondence is shown in Table 2, where we have introduced the following shorthands:

$$
\begin{aligned}
Proc_1 &\triangleq <acq, \mu>.<use, \delta>.<rel, \gamma>.Proc \\
Proc_2 &\triangleq <use, \delta>.<rel, \gamma>.Proc \\
Proc_3 &\triangleq <rel, \gamma>.Proc \\
Mem_1 &\triangleq <rel, *_w>.Mem
\end{aligned}
$$

Theorem 3.6 Let $\mathcal{B} = \{(E, Q) \in \mathcal{G} \times \mathcal{M}(\mathcal{V}) \mid Q = dec(E)\}$ and let $\mathcal{B}'$ be the reflexive, symmetric and transitive closure of $\mathcal{B}$. Whenever $(E, Q) \in \mathcal{B}$, then for all $a \in AType$, $l \in \mathbb{N} \cup \{-1\}$, and equivalence classes $C \in (\mathcal{G} \cup \mathcal{M}(\mathcal{V}))/\mathcal{B}'$

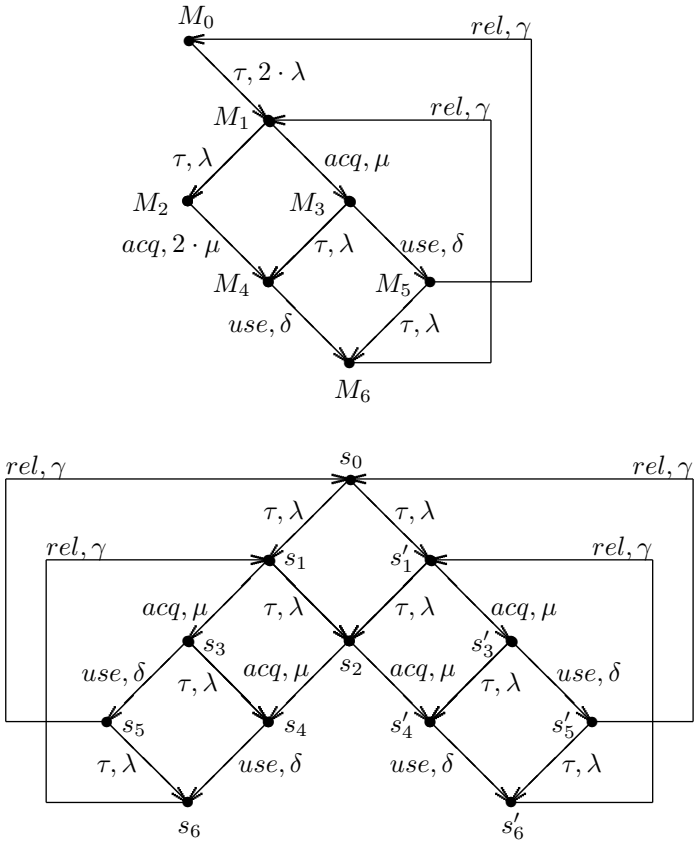


Fig. 2. $\mathcal{RG}[\![\mathcal{N}[\![Sys]\!]]\!]$ and $\mathcal{I}[\![Sys]\!]$

$$\sum\{\!|\, \tilde{\lambda} \mid \exists E' \in C.\, E \xrightarrow{a,\tilde{\lambda}} E' \wedge PL(\tilde{\lambda}) = l \,|\!\}\ =$$
$$\sum\{\!|\, \tilde{\lambda} \mid \exists Q' \in C.\, Q \xrightarrow{a,\tilde{\lambda}} Q' \wedge PL(\tilde{\lambda}) = l \,|\!\}$$

Proof See Appendix C. ∎

4 Properties of the Net Semantics

In this section we investigate the properties of the net semantics just introduced. We recall from the introduction that we aim at improving on the previously proposed net semantics in terms of the compactness of the resulting GSPNs (number of places and transitions) and of their reachability graphs (number of states and transitions). This is especially important for the analysis since the linear equation systems underlying the generated GSPNs can be smaller. Likewise, it is important that the net semantics allows for an easy reinterpretation at the process algebraic level of the results of the analysis conducted at the net level. This is indispensable for the net semantics to be

$M_0 = \{\!\mid V_1, V_1, V_2 \mid\!\}$	$s_0 = (Proc \parallel_\emptyset Proc) \parallel_{\{acq,rel\}} Mem$
$M_1 = \{\!\mid V_1, V_2, V_3 \mid\!\}$	$s_1 = (Proc_1 \parallel_\emptyset Proc) \parallel_{\{acq,rel\}} Mem$
	$s'_1 = (Proc \parallel_\emptyset Proc_1) \parallel_{\{acq,rel\}} Mem$
$M_2 = \{\!\mid V_2, V_3, V_3 \mid\!\}$	$s_2 = (Proc_1 \parallel_\emptyset Proc_1) \parallel_{\{acq,rel\}} Mem$
$M_3 = \{\!\mid V_1, V_4, V_5 \mid\!\}$	$s_3 = (Proc_2 \parallel_\emptyset Proc) \parallel_{\{acq,rel\}} Mem_1$
	$s'_3 = (Proc \parallel_\emptyset Proc_2) \parallel_{\{acq,rel\}} Mem_1$
$M_4 = \{\!\mid V_3, V_4, V_5 \mid\!\}$	$s_4 = (Proc_2 \parallel_\emptyset Proc_1) \parallel_{\{acq,rel\}} Mem_1$
	$s'_4 = (Proc_1 \parallel_\emptyset Proc_2) \parallel_{\{acq,rel\}} Mem_1$
$M_5 = \{\!\mid V_1, V_5, V_6 \mid\!\}$	$s_5 = (Proc_3 \parallel_\emptyset Proc) \parallel_{\{acq,rel\}} Mem_1$
	$s'_5 = (Proc \parallel_\emptyset Proc_3) \parallel_{\{acq,rel\}} Mem_1$
$M_6 = \{\!\mid V_3, V_5, V_6 \mid\!\}$	$s_6 = (Proc_3 \parallel_\emptyset Proc_1) \parallel_{\{acq,rel\}} Mem_1$
	$s'_6 = (Proc_1 \parallel_\emptyset Proc_3) \parallel_{\{acq,rel\}} Mem_1$

Table 2
Markovian bisimilar states in $\mathcal{RG}[\![\mathcal{N}[\![Sys]\!]]\!]$ and $\mathcal{I}[\![Sys]\!]$

used in practice.

In Sect. 4.1 we discuss the compactness of the generated GSPNs with respect to both those generated by the location oriented approach of [16,25] and those generated by the label oriented approach of [4,3]. In Sect. 4.2 we study the compactness of the reachability graphs underlying the generated GSPNs with respect to the LTSs of the corresponding process terms. In Sect. 4.3 we provide some conditions under which the generated GSPNs are finite and we compare them with the corresponding conditions for the location oriented approach. Finally, in Sect. 4.4 we illustrate how to reinterpret at the process algebraic level the results obtained at the GSPN level.

4.1 Comparison at the Net Level

In the location oriented approach of [16,25] the resulting nets are 1-safe, i.e. every place contains at most one token at any time. As formally explained in Appendix A, this is a consequence of the decomposition clause $dec_{\rm loc}(E_1 \parallel_S E_2) = dec_{\rm loc}(E_1) \parallel_S id \oplus id \parallel_S dec_{\rm loc}(E_2)$ for the parallel composition operator, which always keeps distinct the two subnets corresponding to the two operand terms of the parallel composition operator. In our approach, instead, the two subnets can collapse into a single one depending on E_1, E_2,

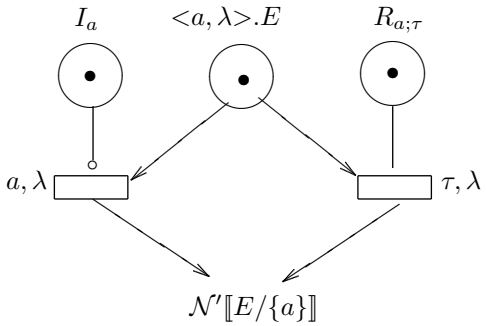
and S.

Theorem 4.1 Let $E_1, E_2 \in \mathcal{G}$ be two sequential terms. Then $|dec(E_1 \|_S E_2)| \leq 2 = |dec_{\mathrm{loc}}(E_1 \|_S E_2)|$. Moreover, if $E_1 \equiv E_2$ and $(sort(E_1) \cup sort(E_2)) \cap S = \emptyset$, then $|dec(E_1 \|_S E_2)| = 1$. ■

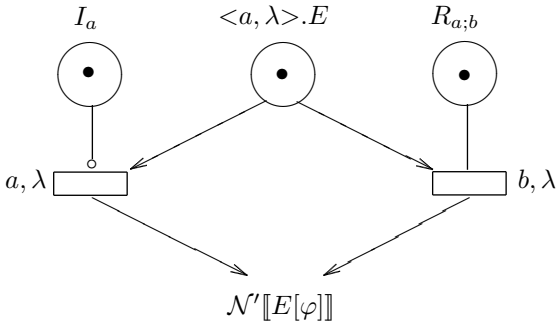
The theorem above straightforwardly generalizes to an arbitrary number of sequential terms and gives an idea of the compactness that can be gained using the proposed label oriented net semantics. This justifies why the label oriented approach should be preferred to the location oriented approach from an applicative point of view, especially when dealing with large systems in which some parts are replicated.

We conclude by observing that the new label oriented net semantics improves on that of [4,3]. In fact, the latter requires the introduction of additional places with inhibitor and contextual arcs to store some information about the syntactical structure of terms. Let us recall that in the graphical representation of a net an inhibitor (contextual) arc is drawn as a circle headed line (simple line), and that a transition having an incoming inhibitor (contextual) arc can be fired only if the place from which the arc departs is unmarked (marked). The three static operators are treated as follows according to the net semantics $\mathcal{N}'$ of [4,3]:

- The net semantics of $(<a, \lambda>.E)/\{a\}$ is

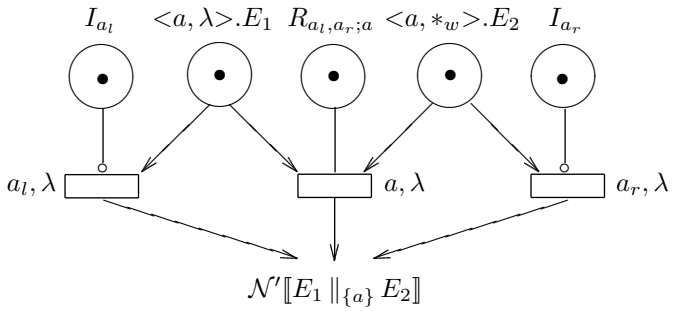


  where I_a is an inhibitor place for all the transitions of type a and $R_{a;\tau}$ is a contextual place for all the transitions of type a to be hidden.
- The net semantics of $(<a, \lambda>.E)[\varphi]$ with $\varphi(a) = \varphi(b) = b$ is



where $R_{a;b}$ is a contextual place for transitions of type a to be relabeled b.

- The net semantics of $<a,\lambda>.E_1 \,\|_{\{a\}}\, <a,*_w>.E_2$ is



where $R_{a_l,a_r;a}$ is a contextual place for transitions of type a_l or a_r to be relabeled a.

By a direct comparison with the corresponding figures in Sect. 3.2, we immediately see that the new label oriented net semantics produces GSPNs that are more compact than those produced by the net semantics of [4,3] as the former does not need any inhibitor or contextual place. Actually, in [4,3] it has been proved that the introduced inhibitor and contextual places can be removed a posteriori, but still this requires an additional computational step with respect to the new net semantics.

4.2 Comparison at the State Level

As shown in Fig. 2, $\mathcal{RG}[\![\mathcal{N}[\![Sys]\!]]\!]$ is smaller than $\mathcal{I}[\![Sys]\!]$. Actually, with an easy adaptation of the proof of the retrievability result, it can be shown that the reverse never happens.

Theorem 4.2 Let $E \in \mathcal{G}$ be a finite state term and let $state(Z)$ be the set of states of LTS Z. Then $|state(\mathcal{RG}[\![\mathcal{N}[\![E]\!]]\!])| \leq |state(\mathcal{I}[\![E]\!])|$. In particular, if $E \equiv E_1 \,\|_S\, E_2$, then $|state(\mathcal{RG}[\![\mathcal{N}[\![E]\!]]\!])| < |state(\mathcal{I}[\![E]\!])|$ whenever $E_1 \equiv E_2$ and $(sort(E_1) \cup sort(E_2)) \cap S = \emptyset$. ■

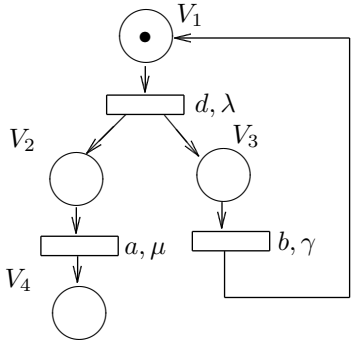
The second part of the theorem above straightforwardly generalizes to an arbitrary number of terms. Again, this is particularly important when dealing with large systems in which some parts are replicated, since aggregated (with respect to the Markovian bisimulation equivalence [8]) state spaces are obtained without applying minimization algorithms such as [26].

4.3 Finiteness Result

Another nice property of our net semantics is that, in certain cases, it yields finite GSPNs for infinite state terms. As an example, consider

$$A \triangleq <d, \lambda>.(<a, \mu>.\underline{0} \parallel_{\emptyset} <b, \gamma>.A)$$

Then $\mathcal{N}[\![A]\!]$ is the finite net depicted in Fig. 3, while $\mathcal{N}_{\mathrm{loc}}[\![A]\!]$ has infinitely many places and $\mathcal{I}[\![A]\!]$ has infinitely many states.



$$V_1 \equiv <d, \lambda>.(<a, \mu>.\underline{0} \parallel_{\emptyset} <b, \gamma>.A)$$
$$V_2 \equiv <a, \mu>.\underline{0}$$
$$V_3 \equiv <b, \gamma>.A$$
$$V_4 \equiv \underline{0}$$

Fig. 3. Finite net semantics of $A \triangleq <d, \lambda>.(<a, \mu>.\underline{0} \parallel_{\emptyset} <b, \gamma>.A)$

Theorem 4.3 Let $E \in \mathcal{G}$ be an infinite state term. If the three following conditions hold:

- For each functional abstraction operator $\_/L$ occurring within a recursive constant definition, $sort(E) \cap L = \emptyset$.
- For each functional relabeling operator $\_[\varphi]$ occurring within a recursive constant definition, $sort(E) \cap \{a \mid \varphi(a) \neq a\} = \emptyset$.
- For each parallel composition operator $\_ \parallel_S \_$ occurring within a recursive constant definition, $sort(E) \cap S = \emptyset$.

then $\mathcal{N}[\![E]\!]$ is finite while $\mathcal{N}_{\mathrm{loc}}[\![E]\!]$ has infinitely many places and $\mathcal{I}[\![E]\!]$ has infinitely many states. ■

The theorem above shows that, for the considered class of infinite state terms, a finite hence analyzable representation can be obtained only with our label oriented net semantics. We conclude by recalling that, given a term E, we are guaranteed that $\mathcal{N}_{\mathrm{loc}}[\![E]\!]$ has finitely many places and $\mathcal{I}[\![E]\!]$ has finitely many states only if no recursive constant invocation occurs within the scope of a functional abstraction, functional relabeling or parallel composition operator.

4.4 Analysis Result Reinterpretation

A desirable property of the net semantics is that of allowing functional and performance properties derived from a GSPN to be reinterpreted at the level

of the process algebraic specification mapped onto that net. In this way one can take advantage of the efficiency of the structural techniques developed for GSPNs [27,11], thus importing such techniques in a formalism that does not directly support them.

For GSPNs there are several structural analysis techniques which can be used to derive functional properties without generating the underlying state space. The most important structural technique is the one based on the computation of P-invariants and T-invariants [27].

Definition 4.4 Let $N = (P, AType \times ARate^{\mathcal{M}(P)}, T, M_0)$ be a GSPN such that $|P| = m$ and $|T| = n$. The *incidence matrix* of N is the matrix

$$\mathbf{A} = [a_{i,j}] \in \mathbb{Z}^{n \times m},\ a_{i,j} = i^{\bullet}(j) - {}^{\bullet}i(j)$$

A *P-invariant* is an integer solution of

$$\mathbf{A} \cdot \mathbf{y} = \mathbf{0}$$

A *T-invariant* is an integer solution of

$$\mathbf{A}^{\mathrm{T}} \cdot \mathbf{x} = \mathbf{0}$$

■

P-invariants single out places that do not change their token count during transition firings, whereas T-invariants indicate how often each transition has to fire in order to reproduce a given marking. P-invariants and T-invariants are computed starting from the incidence matrix and are independent from any initial marking, which is only instrumental for their interpretation. As an example of functional properties that can be proved by exploiting invariants, it can be shown that a GSPN is bounded if for each of its places there exists a P-invariant associating a positive value with that place, while a necessary condition in order for a GSPN to be live and bounded is that for each of its transitions there exists a T-invariant associating a positive value with that transition.

As far as invariants are concerned, our net semantics allows for their reinterpretation at the process algebraic level because net places are in correspondence with (decorated) sequential subterms of the process algebraic specification and net transitions are labeled with actions occurring in the process algebraic specification.

Example 4.5 Let us consider again the system introduced in Ex. 3.2, whose net semantics is depicted in Fig. 1. Its incidence matrix $\mathbf{A}$ is

	V_1	V_2	V_3	V_4	V_5	V_6
τ	-1	0	1	0	0	0
acq	0	-1	-1	1	1	0
use	0	0	0	-1	0	1
rel	1	1	0	0	-1	-1

The net has four P-invariants

$$\mathbf{y}_1^{\mathrm{T}} = [\ 1\ 0\ 1\ 1\ 0\ 1\ ]$$
$$\mathbf{y}_2^{\mathrm{T}} = [\ 1\ 0\ 1\ 0\ 1\ 0\ ]$$
$$\mathbf{y}_3^{\mathrm{T}} = [\ 0\ 1\ 0\ 0\ 1\ 0\ ]$$
$$\mathbf{y}_4^{\mathrm{T}} = [\ 0\ 1\ 0\ 1\ 0\ 1\ ]$$

and one T-invariant

$$\mathbf{x}^{\mathrm{T}} = [\ 1\ 1\ 1\ 1\ ]$$

By exploiting the P-invariants above, we are able to prove that the memory is used in a mutually exclusive way. Since each P-invariant identifies a subset of the places of the net whose token count is invariant under transition firing, we know that

$$M(V_1) + M(V_3) + M(V_4) + M(V_6) = 2$$
$$M(V_1) + M(V_3) + M(V_5) = 2$$
$$M(V_2) + M(V_5) = 1$$
$$M(V_2) + M(V_4) + M(V_6) = 1$$

where M denotes an arbitrary marking while the constant occurring in each equation is determined by the initial marking. To prove mutual exclusion, we have to demonstrate that there is no state in which both processes can execute an action of type *use*. At the net level, V_4 is the place representing a process ready to perform action $<use, \delta>$, so we have to demonstrate that there is no marking in which place V_4 contains more than one token. This can obviously be done by constructing the reachability graph of the GSPN, but it can be also done more efficiently by exploiting the information provided by the fourth P-invariant, as it establishes that, for each reachable marking, the sum of the number of tokens in places V_2, V_4, and V_6 is one.

By exploiting the only T-invariant, instead, we are able to prove that the initial state of the process algebraic specification is a home state, because the initial marking of the GSPN enables the transition sequence $<\tau, \lambda>, <acq, \mu>, <use, \delta>, <rel, \gamma>$ whose transition count vector coincides with the T-invariant. ■

On the performance side, we mention the existence of structural analysis techniques for the computation of the maximum and the average numbers of tokens in the places (which provide a measure of the level of parallelism of the modeled system) as well as upper bounds on the throughput of the transitions (hence of the corresponding actions), without generating the underlying state space. This is accomplished at the net level by solving suitable linear programming problems [11]. Also in this case our net semantics turns out to be useful, because it allows for the reinterpretation of the bounds at the process algebraic level. Again, this is made possible by the correspondence between net places and sequential subterms as well as net transitions and actions. For an example, the reader is referred to Sect. 6.

5 Integrating TwoTowers and GreatSPN

In this section we present an implementation of the net semantics proposed in this paper that is realized to integrate the EMPA_{gr} based software tool TwoTowers and the GSPN based software tool GreatSPN. In Sect. 5.1 and 5.2 we recall some features of the two software tools, then in Sect. 5.3 we discuss their integration.

5.1 TwoTowers

TwoTowers [7] is a software tool for the functional verification and performance evaluation of computer, communication and software systems described in EMPA_{gr}. TwoTowers is composed of a graphical user interface, a compiler, an integrated analyzer, a functional analyzer, and a performance analyzer.

The graphical user interface allows the user to edit EMPA_{gr} specifications, compile them into the semantic models, and run the various analysis routines. Additionally, it permits the editing of the specifications of functional requirements and performance metrics.

The compiler is in charge of parsing EMPA_{gr} specifications and signaling possible errors occurring in them. If a specification is correct, the compiler can produce the semantic model on which further analyses are based.

The integrated analyzer checks whether some given integrated (i.e. functional and performance) requirements are satisfied by a correct EMPA_{gr} specification. More precisely, the integrated analyzer contains a routine to check two correct, finite state EMPA_{gr} specifications for Markovian bisimulation equivalence [3].

The functional analyzer takes care of verifying that certain functional require-

ments are satisfied by a correct EMPA_{gr} specification. This is achieved by interfacing TwoTowers with a version of CWB-NC [15] obtained via PAC-NC [14], thereby providing support for model checking, equivalence checking, and preorder checking.

Finally, the performance analyzer computes certain performance metrics for a correct EMPA_{gr} specification. This is carried out via Markovian analysis, by interfacing TwoTowers with a suitably modified version of MarCA [29], or via simulation.

5.2 GreatSPN

GreatSPN [13] is a software package for the description, verification, and performance evaluation of computer, communication and software systems using GSPNs (and their colored extension). It provides a graphical user interface for model editing and for structural properties and performance results visualization. Once a GSPN model has been completely specified, its structural analysis can start allowing the investigation of interesting model properties without building its state space. Among the structural analysis modules, we recall the one responsible for the computation of invariants for checking state space boundedness, the one responsible for the computation of implicit places, i.e. places which are redundant in the net, and the one responsible for computing performance bounds.

If the GSPN is bounded, the modules responsible for the generation and the analysis of the underlying state space can be invoked. After the derivation of the reachability graph, functional properties like the presence of home states, deadlocks, and livelocks can be computed. If the GSPN has a finite state space, the underlying Markov chain is derived and transient or stationary marking probability distributions can be computed as well as a number of default and user defined performance measures. The GreatSPN architecture contains also several simulation modules that cooperate with each other in order to obtain the interactive simulation (supported by animation of the graphic representation) and performance estimations.

5.3 Implementation of the Net Semantics

Given an EMPA_{gr} specification, the newly implemented module of TwoTowers for interfacing with GreatSPN generates the corresponding GSPN according to the new net semantics by initially producing all the places. This is accomplished by computing the decomposition of the initial constant defining equation of the specification (which corresponds to the initial marking), and

by repeating this procedure for each sequential term associated with a place after removing its outermost action prefixes. As an example, if the sequential term associated with a given place is $\sum_{i \in I} <a_i^{\theta_i, R_i, \sigma_i}, \tilde{\lambda}_i>.E_i$, we compute $dec(E_i)$ for all $i \in I$. The places, as well as the sequential terms, are stored in a hash table for efficient retrieval. The bucket of a term points to the buckets of the related places resulting from its decomposition; in this way, it is straightforward to see whether the decomposition of a given term has already been computed or not. Similarly, the bucket of a place points to the bucket of the related sequential term; this allows an efficient reinterpretation at the process algebraic level of analysis results computed at the net level.

Once all the places have been generated, the transitions can be produced. To do this, we examine the parallel composition related decorations of the actions within the places in order to build transition presets according to Condition (3) of Appendix B, which establishes that the combinator strings of the selected actions in each place of the preset pairwise reduce until the empty string is obtained. More precisely, in the first round we consider those places having some actions with the empty string as combinator string. Such places cannot synchronize with any others and hence give rise to transitions with a singleton preset. In the next rounds, we consider the remaining places two by two and we try to reduce the combinator strings of their selected actions. Three cases arise:

- If two places combine and the resulting combinator string is not empty, a new dummy place is generated which is considered in the subsequent rounds instead of the two original places. The dummy place contains the information about the two places that have generated it and is given the resulting combinator string.
- If two places combine and the resulting combinator string is empty, a new synchronization transition is generated and the two places will no longer be considered in the subsequent rounds. The preset of the newly generated transition is composed of these two places. When one or both of them are dummy places, their expansion is included in the preset instead.
- If two places do not combine, nothing happens.

The number of rounds is bounded by the length of the longest combinator string occurring in the generated places.

Given an EMPA_{gr} specification, TwoTowers produces the corresponding GSPN in the format expected by GreatSPN using the algorithm above. This GSPN can be analyzed by invoking via TwoTowers the routines implemented in GreatSPN for the computation of P-invariants, T-invariants, maximum and average numbers of tokens in the places, and upper bounds on the throughput of the transitions. Finally, some user defined performance measures can be specified at the net level and then evaluated on the underlying state space.

6 Computing Performance Bounds for Random Polling Systems

In this section we apply the new GSPN semantics to the EMPA_{gr} specification of a simple multi-server multi-queue system (MSMQS) to show that the semantics is able to detect and exploit symmetries and allows performance bounds to be efficiently computed.

A MSMQS is composed of a set of waiting lines that receive arrivals from the external world, and a set of servers that attend the queues (stations) and provide service when required (i.e., when some customers are waiting in the queues). Different MSMQSs can be obtained depending on the features which are specified, such as the number of queues, their sizes, the customer arrival process at each queue, the number of servers, the service time distribution, the walk time distribution, and the service policy. Many variations of these types of systems have been investigated in the literature since they are quite easy to understand but their analysis is not trivial. Performance analysis of MSMQSs through GSPNs can be found in [2], while SPA models of MSMQSs are considered in [22].

Our example is a MSMQS composed of $n \in \mathbb{N}_+$ queues and $m \in \mathbb{N}_+$ servers, with $m \leq n$. For simplicity, each queue has capacity one including the customer in service when appropriate. Each server randomly polls a queue and checks if there is a customer to be served. If so, it removes the customer from the buffer, serves it, and then walks to the next randomly chosen queue, even the one it has just visited. We assume that the servers and the queues have the same timing characteristics: arrival times, service times, and walk times have rates λ, μ, and ω, respectively. Note that the system is symmetric with respect to both the servers and the queues.

The EMPA_{gr} specification of the architecture of the MSMQS at hand is the following:

$$RPS_{m,n} \triangleq (S \parallel_\emptyset S \parallel_\emptyset \ldots \parallel_\emptyset S) \parallel_T (Q \parallel_\emptyset Q \parallel_\emptyset \ldots \parallel_\emptyset Q)$$
$$T = \{is\_full, serve\}$$

Each server is specified as follows:

$$S \triangleq <walk, \omega>.(<is\_full, \infty_{2,1}>.<serve, \mu>.S + <is\_empty, \infty_{1,1}>.S)$$

After the execution of action $<walk, \omega>$, the server reaches a queue and checks if there is a waiting customer. Immediate action $<is\_full, \infty_{2,1}>$ models this situation and, if enabled, it is executed in zero time; then a service is provided (action $<serve, \mu>$) and the server repeats its behavior. If the queue is empty, the server leaves it by executing immediate action $<is\_empty, \infty_{1,1}>$ and re-

peats its behavior. Note that action with type *is_full* has priority level 2 while action with type *is_empty* has priority level 1, meaning that if a customer is waiting in the randomly chosen queue, the server does serve it, as expected.

Each queue is specified as follows:

$$Q \triangleq <arrive, \lambda>.<is\_full, *_w>.<serve, *_w>.Q$$

Customer arrivals are modeled through action $<arrive, \lambda>$. The other two actions are passive within this component: their rates will be determined upon synchronization with the corresponding nonpassive actions performed by a server when polling the queue.

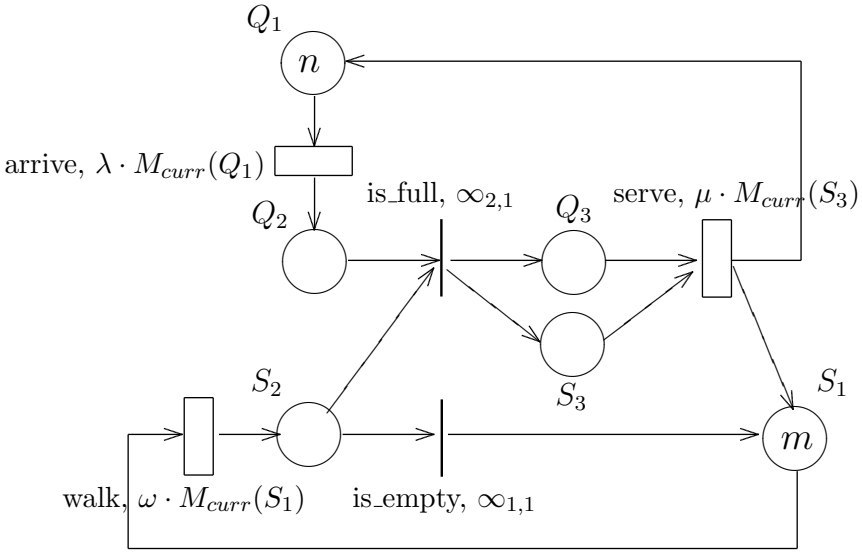


Fig. 4. $\mathcal{N}[\![RPS_{m,n}]\!]$

The GSPN corresponding to $RPS_{m,n}$ is shown in Fig. 4. As explained in Sect. 5.3, it is obtained by computing the initial marking

$$\begin{aligned}
dec(RPS_{m,n}) &= dec((S \parallel_\emptyset S \parallel_\emptyset \ldots \parallel_\emptyset S) \parallel_{\{is_full, serve\}} (Q \parallel_\emptyset Q \parallel_\emptyset \ldots \parallel_\emptyset Q)) \\
&= dec(S \parallel_\emptyset S \parallel_\emptyset \ldots \parallel_\emptyset S)\{is_full^{\varepsilon,\emptyset,k_1}/is_full, serve^{\varepsilon,\emptyset,k_2}/serve\} \oplus \\
&\quad dec(Q \parallel_\emptyset Q \parallel_\emptyset \ldots \parallel_\emptyset Q)\{is_full^{\varepsilon,\emptyset,\bar{k}_1}/is_full, serve^{\varepsilon,\emptyset,\bar{k}_2}/serve\} \\
&= dec(S_1 \parallel_\emptyset S_1 \parallel_\emptyset \ldots \parallel_\emptyset S_1) \oplus \\
&\quad dec(Q_1 \parallel_\emptyset Q_1 \parallel_\emptyset \ldots \parallel_\emptyset Q_1) \\
&= \{\!| S_1, S_1, \ldots, S_1, Q_1, Q_1, \ldots, Q_1 |\!\}
\end{aligned}$$

and then generating all the places

$$
\begin{aligned}
S_1 &\equiv <walk, \omega>.S_2 \\
S_2 &\equiv <is\_full^{\varepsilon,\emptyset,k_1}, \infty_{2,1}>.S_3 + \\
&\quad\ <is\_empty, \infty_{1,1}>.S_1 \\
S_3 &\equiv <serve^{\varepsilon,\emptyset,k_2}, \mu>.S_1 \\
Q_1 &\equiv <arrive, \lambda>.Q_2 \\
Q_2 &\equiv <is\_full^{\varepsilon,\emptyset,\bar{k}_1}, *_w>.Q_3 \\
Q_3 &\equiv <serve^{\varepsilon,\emptyset,\bar{k}_2}, *_w>.Q_1
\end{aligned}
$$

Place S_1 models the servers before polling a queue. When this place is marked, the transition of type *walk* is enabled and its firing represents one server reaching a queue. The rate of this transition varies according to the marking of place S_1. Place S_2 represents the servers in front of the queues. The arrival of a new customer is modeled by the transition of type *arrive* whose rate varies according to the number of tokens in Q_1. If there are waiting customers, i.e. place Q_2 is marked, the immediate transition of type *is_full* can fire modeling a synchronization between one server and one queue. Now the transition of type *serve* can fire modeling the provision of service. If there are no waiting customers, the immediate transition of type *is_empty* can instead fire.

Note that the symmetry in the roles of the servers and the queues is reflected at the net level by the presence of two subnets. The former is composed of places S_1, S_2, and S_3, while the latter is composed of places Q_1, Q_2, and Q_3. Changes in the number of servers and/or queues do not alter the structure of the net but only its initial marking.

Given the GSPN model of Fig. 4 automatically produced by TwoTowers, we can run both TwoTowers and GreatSPN to compute the system throughput for $\lambda = 1$, $\mu = 2$, and $\omega = 3$, when varying the numbers m of servers and n of queues from 1 to 5. The results are shown in Table 3. The third and fourth columns refer to TwoTowers and describe the number of states of the integrated interleaving semantics and the value of the service throughput, respectively. The last three columns, instead, refer to GreatSPN and describe the number of states of the reachability graph of the integrated net semantics, the system throughput, and an upper bound on the system throughput, respectively. As can be observed, for systems exhibiting a high degree of symmetry, working with TwoTowers at the integrated interleaving semantic model level requires an amount of time which grows exponentially with the number of servers and queues. Instead, working with GreatSPN on the GSPN model derived from TwoTowers by applying our net semantics requires an amount of time which grows linearly with the number of servers and queues. Moreover, with the GSPN model it is possible to avoid the state space construction if we restrict ourselves to compute bounds on the system throughput.

m	n	$\#\mathcal{I}[\![RPS_{m,n}]\!]$	throughput	$\#\mathcal{RG}[\![\mathcal{N}[\![RPS_{m,n}]\!]]\!]$	throughput	upper bound
1	1	5	0.545455	5	0.545455	0.666778
1	2	12	0.923077	8	0.923077	1.200000
2	2	29	1.159270	11	1.159270	1.333333
1	3	28	1.116280	11	1.116280	1.200000
2	3	78	1.636170	16	1.636170	2.000000
3	3	177	1.796080	19	1.796080	2.000000
1	4	64	1.182550	14	1.182550	1.200000
2	4	200	1.997520	21	1.997530	2.400000
3	4	504	2.318680	26	2.318680	2.666700
4	4	1089	2.443010	29	2.443010	2.666700
1	5	144	1.197440	17	1.197440	1.200000
2	5	496	2.224840	26	2.224850	2.400000
3	5	1368	2.766000	33	2.766010	3.333333
4	5	3210	2.991610	38	2.991610	3.333333
5	5	6693	3.095440	41	3.095430	3.333333

Table 3
Results of the analysis of $RPS_{m,n}$

7 Conclusion

In this paper we have presented and proved correct a new label oriented GSPN semantics for EMPA_{gr}, which improves all the net semantics previously published in the literature with respect to the size of the generated nets and allows functional and performance properties efficiently derived at the GSPN level via structural techniques to be reinterpreted at the EMPA_{gr} level. Because of its practical advantages, such a label oriented GSPN semantics has been implemented to integrate the EMPA_{gr} based software tool TwoTowers with the GSPN based software tool GreatSPN, in order to create a tool for the formal modeling and analysis of complex systems that benefits from the complementary strengths of both formalisms.

Related Work. A comparison with the net semantics for process algebras previously proposed in the literature has been made throughout the paper. As we have seen, one of the advantages of our new semantics – the compactness of the resulting GSPNs – allows system symmetries to be captured and exploited at analysis time.

The detection of symmetries for state space reduction is not new in the field of SPAs. In [18] a reduced state space underlying a PEPA [22] model is obtained by building equivalence classes of states and by considering one element for each class only. Equivalence classes of states are computed syntactically by looking at the structure of the terms and by considering as equivalent those states which differ in the position of the components within a parallel composition. For example, if we consider the term $E \parallel_S E$, its possible derivatives $E \parallel_S E'$ and $E' \parallel_S E$ are considered equivalent and therefore belong to the same equivalence class. The reduction in the size of the state space is not optimal – it is still possible to find states which are Markovian bisimulation equivalent but do not belong to the same equivalence class – but it is stronger than ours since it works for synchronized replicas whose corresponding GSPNs are kept distinct in our framework. In this respect, it is worth observing that the advantage of having a GSPN representation of an EMPA_{gr} term is not limited to state space reduction only. In fact, structural techniques for functional verification and performance evaluation like those mentioned in Sect. 4.4 can be applied to the GSPN representation of an EMPA_{gr} specification, thereby avoiding the construction of the underlying state space.
Another approach to state space reduction is discussed in [21]. Here a new operator is introduced for representing sets of identical processes, all synchronizing on the same sets of actions (possibly the empty set). New semantic rules have been defined, which are consistent with those for the traditional parallel composition operator but avoid the representation of all possible interleavings. An informal net semantics which generates k-safe nets is also discussed.
Finally, we mention the work in [17] where an attempt is made to adapt for PEPA the structural analysis techniques typical of Petri nets. This is achieved thanks to a matrix representation of PEPA components that is reminiscent of incidence matrices of Petri nets.

Future Work. The GSPN model of the random polling system we have shown in Sect. 6 is very compact since the net semantics captures all the intrinsic symmetries of the system. In order to be able to capture a higher number of symmetries for more complex systems, we may think of changing the underlying model of our semantics in favor of more appropriate models for which reduced state space construction methods have been already proposed. For instance, we may think of mapping EMPA_{gr} terms onto stochastic well formed nets (SWNs) [12] or stochastic activity networks (SANs) [23]. As an example, in a random polling system there might be a customer that is faster than the others. In such a case the customers are still equivalent from a purely functional point of view, but the GSPN generated by applying our net semantics to the EMPA_{gr} specification of the system contains a distinct subnet for the fast customer because of the different rate values. The same would not happen e.g. in a SWN model since there it is possible to define transition rates depending on the identity of the tokens, i.e. on the identity of the customers.

References

[1] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, G. Franceschinis, *"Modelling with Generalized Stochastic Petri Nets"*, Wiley & Sons, 1995

[2] M. Ajmone Marsan, S. Donatelli, F. Neri, *"GSPN Models of Markovian Multiserver Multiqueue Systems"*, in Performance Evaluation 11:227-240, 1990

[3] M. Bernardo, *"Theory and Application of Extended Markovian Process Algebra"*, Ph.D. Thesis, University of Bologna (Italy), 1999

[4] M. Bernardo, N. Busi, R. Gorrieri, *"A Distributed Semantics for EMPA Based on Stochastic Contextual Nets"*, in Computer Journal 38:492-509, 1995

[5] M. Bernardo, N. Busi, M. Ribaudo, *"Integrating TwoTowers and GreatSPN"*, in Proc. of the *8th Int. Workshop on Process Algebra and Performance Modelling (PAPM '00)*, Carleton Scientific, pp. 551-563, Geneva (Switzerland), 2000

[6] M. Bernardo, N. Busi, M. Ribaudo, *"Compact Net Semantics for Process Algebras"*, in Proc. of the *IFIP Joint Int. Conf. on Formal Description Techniques for Distributed Systems and Communication Protocols and Protocol Specification, Testing and Verification (FORTE/PSTV '00)*, Kluwer, pp. 319-334, Pisa (Italy), 2000

[7] M. Bernardo, W.R. Cleaveland, S.T. Sims, W.J. Stewart, *"TwoTowers: A Tool Integrating Functional and Performance Analysis of Concurrent Systems"*, in Proc. of the *IFIP Joint Int. Conf. on Formal Description Techniques for Distributed Systems and Communication Protocols and Protocol Specification, Testing and Verification (FORTE/PSTV '98)*, Kluwer, pp. 457-467, Paris (France), 1998

[8] M. Bravetti, M. Bernardo, *"Compositional Asymmetric Cooperations for Process Algebras with Probabilities, Priorities, and Time"*, Tech. Rep. UBLCS-2000-01, University of Bologna (Italy), 2000 (extended abstract in Proc. of the *1st Int. Workshop on Models for Time Critical Systems (MTCS '00)*, Electronic Notes in Theoretical Computer Science 39(3), State College (PA), 2000)

[9] N. Busi, R. Gorrieri, *"A Petri Net Semantics for π-Calculus"*, in Proc. of the *6th Int. Conf. on Concurrency Theory (CONCUR '95)*, LNCS 962:145-159, Philadelphia (PA), 1995

[10] N. Busi, R. Gorrieri, G. Zavattaro, *"On the Expressiveness of Linda Coordination Primitives"*, in Information and Computation 156:90-121, 2000

[11] G. Chiola, C. Anglano, J. Campos, J.M. Colom, M. Silva, *"Operational Analysis of Timed Petri Nets and Applications to the Computation of Performance Bounds"*, in Proc. of the *5th Int. Workshop on Petri Nets and*

Performance Models (PNPM '93), IEEE-CS Press, pp. 128-137, Toulouse (France), 1993

[12] G. Chiola, C. Dutheillet, G. Franceschinis, S. Haddad, *"On Well-Formed Coloured Nets and their Symbolic Reachability Graph"*, in Proc. of the *11th Int. Conf. on Application and Theory of Petri Nets (ATPN '90)*, Paris (France), 1990

[13] G. Chiola, G. Franceschinis, R. Gaeta, M. Ribaudo, *"GreatSPN 1.7: Graphical Editor and Analyzer for Timed and Stochastic Petri Nets"*, in Performance Evaluation 24:47-68, 1995

[14] W.R. Cleaveland, E. Madelaine, S.T. Sims, *"A Front-End Generator for Verification Tools"*, in Proc. of the *1st Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '95)*, LNCS 1019:153-173, Aarhus (Denmark), 1995

[15] W.R. Cleaveland, S.T. Sims, *"The NCSU Concurrency Workbench"*, in Proc. of the *8th Int. Conf. on Computer Aided Verification (CAV '96)*, LNCS 1102:394-397, New Brunswick (NJ), 1996

[16] P. Degano, R. De Nicola, U. Montanari, *"A Distributed Operational Semantics for CCS Based on Condition/Event Systems"*, in Acta Informatica 26:59-91, 1988

[17] S. Gilmore, J. Hillston, L. Recalde, *"Elementary Structural Analysis for PEPA"*, Tech. Rep. ECS-LFCS-97-377, University of Edinburgh (UK), 1997

[18] S. Gilmore, J. Hillston, M. Ribaudo, *"An Efficient Algorithm for Aggregating PEPA Models"*, in IEEE Trans. on Software Engineering 27:449-464, 2001

[19] U. Goltz, *"On Representing CCS Programs by Finite Petri Nets"*, in Proc. of the *13th Int. Symp. on Mathematical Foundations of Computer Science (MFCS '88)*, LNCS 324:339-350, Carlsbad (Czechoslovakia), 1988

[20] H. Hermanns, *"Interactive Markov Chains"*, Ph.D. Thesis, University of Erlangen (Germany), 1998

[21] H. Hermanns, M. Ribaudo, *"Exploiting Symmetries in Stochastic Process Algebras"*, in Proc. of the *12th European Simulation Multiconference (ESM '98)*, SCS Europe, pp. 763-770, 1998

[22] J. Hillston, *"A Compositional Approach to Performance Modelling"*, Cambridge University Press, 1996

[23] J.F. Meyer, W.H. Sanders, *"Reduced Base Model Construction Methods for Stochastic Activity Networks"*, in IEEE Journal on Selected Areas in Communications 9:25-36, 1991

[24] R. Milner, *"Communication and Concurrency"*, Prentice Hall, 1989

[25] E.-R. Olderog, *"Nets, Terms and Formulas"*, Cambridge University Press, 1991

[26] R. Paige, R.E. Tarjan, *"Three Partition Refinement Algorithms"*, in SIAM Journal of Computing 16:973-989, 1987

[27] W. Reisig, *"Petri Nets: An Introduction"*, Springer-Verlag, 1985

[28] M. Ribaudo, *"On the Relationship between Stochastic Process Algebras and Stochastic Petri Nets"*, Ph.D. Thesis, University of Torino (Italy), 1995

[29] W.J. Stewart, *"Introduction to the Numerical Solution of Markov Chains"*, Princeton University Press, 1994

[30] G. Winskel, *"The Formal Semantics of Programming Languages"*, MIT Press, 1993

A Definition of the Decomposition Function

The decomposition function introduces suitable decorations into the action types through syntactical substitutions. In order to avoid undesired syntactical substitutions, we identify binders arising from the static operators. In E/L, all the action types in L are bound in E. In $E[\varphi]$, all the action types in $Dom(\varphi) = \{c \in AType \mid \varphi(c) \neq c\}$ are bound in E. For operational convenience, we rewrite the synchronization set S of $E_1 \|_S E_2$ as the function $s = \{(a,a) \mid a \in S\}$. In $E_1 \|_s E_2$, all the action types in $\pi_1(s) = \{a \mid \exists b.\, (a,b) \in s\}$ are bound in E_1 and E_2. When dealing with syntactical substitutions, we make explicit the action types occurring in the body E of each constant definition $A \triangleq E$ by rewriting the definition itself as $A(a_1, \ldots, a_n) \triangleq E$, where $\{a_1, \ldots, a_n\} = sort(E)$. The semantic rule **(CI)** in Table 1 is modified as follows

$$\frac{E\{b_i/a_i \mid 1 \leq i \leq n\} \xrightarrow{a,\tilde{\lambda}} E'}{A(b_1, \ldots, b_n) \xrightarrow{a,\tilde{\lambda}} E'} \quad A(a_1, \ldots, a_n) \triangleq E$$

where $E\{b_i/a_i \mid 1 \leq i \leq n\}$ is the term obtained from E by replacing each free occurrence of a_i with b_i, as formalized below.

Definition A.1 Let us assume a total ordering $\preceq$ be defined over $AType$. The decomposition function $dec : \mathcal{G}' \longrightarrow \mathcal{M}(\mathcal{V})$ is defined by structural induction as follows (assuming $A(a_1, \ldots, a_n) \triangleq E$):

$$
\begin{aligned}
dec(\textstyle\sum_{i\in I} <a_i^{\theta_i,R_i,\sigma_i}, \tilde{\lambda}_i>.E_i) &= \{\!|\ \textstyle\sum_{i\in I} <a_i^{\theta_i,R_i,\sigma_i}, \tilde{\lambda}_i>.E_i\ |\!\} \\
dec(E/L) &= dec(E\{a^{\tau,\emptyset,\varepsilon}/a \mid a \in L\}) \\
dec(E[\varphi]) &= dec(E\{d^{\theta,R\cup R_a,\sigma}/a \mid a \in sort(E) \wedge \\
&\qquad (a, d^{\theta,R,\sigma}) \in \varphi \wedge \\
&\qquad R_a = \{r_{ca} \mid \varphi(c) = \varphi(a) \wedge \\
&\qquad\quad c \prec a \wedge \\
&\qquad\quad r_{ca} \text{ fresh}\} \cup \\
&\qquad \{\bar{r}_{ac} \mid \varphi(c) = \varphi(a) \wedge \\
&\qquad\quad a \prec c \wedge \\
&\qquad\quad r_{ac} \text{ fresh}\}\}) \\
dec(E_1 \parallel_s E_2) &= dec(E_1\{d^{\theta,R,\sigma k_a}/a \mid (a, d^{\theta,R,\sigma}) \in s \wedge \\
&\qquad k_a \text{ fresh}\}) \oplus \\
&\quad dec(E_2\{d^{\theta,R,\sigma \bar{k}_a}/a \mid (a, d^{\theta,R,\sigma}) \in s \wedge \\
&\qquad k_a \text{ fresh}\}) \\
dec(A(b_1^{\theta_1,R_1,\sigma_1}, \ldots, b_n^{\theta_n,R_n,\sigma_n})) &= dec(E\{b_i^{\theta_i,R_i,\sigma_i}/a_i \mid 1 \le i \le n\})
\end{aligned}
$$

with the syntactical substitutions on $\mathcal{G}'$ defined by structural induction as follows:

$$
\begin{aligned}
(\textstyle\sum_{i\in I} <a_i^{\theta_i,R_i,\sigma_i}, \tilde{\lambda}_i>.E_i)\{b^{\theta,R,\sigma}/a\} &= \textstyle\sum_{i\in I} <c_i^{\theta'_i,R'_i,\sigma'_i}, \tilde{\lambda}_i>.E_i\{b^{\theta,R,\sigma}/a\} \\
E/L\{b^{\theta,R,\sigma}/a\} &= \begin{cases} E\{b^{\theta,R,\sigma}/a\}/L & \text{if } a \notin L \\ E/L & \text{if } a \in L \end{cases} \\
E[\varphi]\{b^{\theta,R,\sigma}/a\} &= \begin{cases} E\{b^{\theta,R,\sigma}/a\}[\varphi'] & \text{if } a \notin Dom(\varphi) \\ E[\varphi'] & \text{if } a \in Dom(\varphi) \end{cases} \\
(E_1 \parallel_s E_2)\{b^{\theta,R,\sigma}/a\} &= \begin{cases} E_1\{b^{\theta,R,\sigma}/a\} \parallel_s E_2\{b^{\theta,R,\sigma}/a\} & \\ \qquad\qquad \text{if } a \notin \pi_1(s) & \\ E_1 \parallel_{s'} E_2 \qquad \text{if } a \in \pi_1(s) & \end{cases} \\
A(a_1^{\theta_1,R_1,\sigma_1}, \ldots, a_n^{\theta_n,R_n,\sigma_n})\{b^{\theta,R,\sigma}/a\} &= A(c_1^{\theta'_1,R'_1,\sigma'_1}, \ldots, c_n^{\theta'_n,R'_n,\sigma'_n})
\end{aligned}
$$

where:

$$
\begin{aligned}
\varphi' &= \{(d_i, c_i^{\theta'_i,R'_i,\sigma'_i}) \mid \exists a_i^{\theta_i,R_i,\sigma_i}.\, (d_i, a_i^{\theta_i,R_i,\sigma_i}) \in \varphi\} \\
s' &= \{(d_i, c_i^{\theta'_i,R'_i,\sigma'_i}) \mid \exists a_i^{\theta_i,R_i,\sigma_i}.\, (d_i, a_i^{\theta_i,R_i,\sigma_i}) \in s\}
\end{aligned}
$$

and:

$$c_i^{\theta'_i,R'_i,\sigma'_i} = \begin{cases} b^{\theta_i\theta,R_i\cup R,\sigma_i\sigma} & \text{if } a_i = a \\ a_i^{\theta_i,R_i,\sigma_i} & \text{if } a_i \neq a \end{cases}$$ ■

It is worth recalling that in the location oriented approach an explicit track of the parallel composition operator would be kept by decomposing $E_1 \parallel_S E_2$ into

$$dec_{\rm loc}(E_1) \parallel_S id \oplus id \parallel_S dec_{\rm loc}(E_2)$$

Here, instead, according to the label oriented approach, we do not explicitly keep track of the parallel composition operator. As a consequence, instances of the same sequential term occurring within the scope of a parallel composition operator can be mapped onto the same place; e.g., term $<a,\lambda>.\underline{0} \parallel_\emptyset <a,\lambda>.\underline{0}$ is mapped onto a single place $<a,\lambda>.\underline{0}$ with multiplicity two.

Let us see some critical examples showing how the decomposition function and the decorations work. As far as the interplay of the parallel composition and the functional abstraction is concerned, the binders are exploited to avoid the synchronization of hidden actions. As an example, in term

$$(<a,\lambda>.\underline{0})/\{a\} \parallel_{\{a\}} <a,*_w>.\underline{0}$$

the synchronization of the two a actions is not possible because the left hand one is hidden. This is reflected at the net level by the fact that the generated places $<a^{\tau,\emptyset,\varepsilon},\lambda>.\underline{0}$ and $<a^{\varepsilon,\emptyset,\bar{k}},*_w>.\underline{0}$ cannot synchronize because their combinators do not reduce. In particular, the combinator of the left hand a action is ε instead of k because that a action occurs within the scope of binder $/\{a\}$.

As far as the interplay of the parallel composition and the functional relabeling is concerned, the related decorations are used in a combined way in order to avoid the synchronization between different actions whose types collapse after relabeling them. As an example, in term

$$(<a,\lambda>.\underline{0} \parallel_{\{b\}} <b,*_w>.\underline{0})[\varphi],\ \varphi(a) = \varphi(b) = b$$

no synchronization on b is possible because the parallel composition is within the scope of the functional relabeling and not the other way around. Again, this is reflected at the net level by the fact that the generated places $<b^{\varepsilon,\{r\},k},\lambda>.\underline{0}$ and $<b^{\varepsilon,\{\bar{r}\},\bar{k}},*_w>.\underline{0}$ cannot synchronize, despite the fact that they have complementary combinators k and $\bar{k}$ reducing to ε, because of the presence of complementary conflicts r and $\bar{r}$.

We observe that the identification of binders to avoid undesired syntactical substitutions is necessary mainly because, in the case of the functional relabeling operator, the actions are directly relabeled within the net places. As an example, consider term

$$(<a,\lambda>.\underline{0} + <b,\mu>.\underline{0})/\{a\}[\varphi],\ \varphi(a) = \varphi(b) = b$$

which can execute action $<\tau,\lambda>$ or action $<b,\mu>$. When decomposing this term, there are two possible translations causing errors if $/\{a\}$ is not correctly taken into account. In the former case, one may think of relabeling the actions

occurring in $/\{a\}$ as well, thus erroneously obtaining term

$$(<b^{\varepsilon,\{r\},\varepsilon}, \lambda>.\underline{0} + <b^{\varepsilon,\{\bar{r}\},\varepsilon}, \mu>.\underline{0})/\{b\}$$

which gives rise to place $<b^{\tau,\{r\},\varepsilon}, \lambda>.\underline{0} + <b^{\tau,\{\bar{r}\},\varepsilon}, \mu>.\underline{0}$ from which only a transition labeled with $\tau, \lambda + \mu$ can be generated. In the latter case, one may think of not considering $/\{a\}$ at all, thereby erroneously obtaining term

$$(<b^{\varepsilon,\{r\},\varepsilon}, \lambda>.\underline{0} + <b^{\varepsilon,\{\bar{r}\},\varepsilon}, \mu>.\underline{0})/\{a\}$$

which gives rise to place $<b^{\varepsilon,\{r\},\varepsilon}, \lambda>.\underline{0} + <b^{\varepsilon,\{\bar{r}\},\varepsilon}, \mu>.\underline{0}$ from which only a transition labeled with $b, \lambda + \mu$ can be generated. In conclusion, the actions occurring in the binders can be neither relabeled nor ignored. For this reason, we decompose the term at hand into place $<a^{\tau,\emptyset,\varepsilon}, \lambda>.\underline{0} + <b^{\varepsilon,\{\bar{r}\},\varepsilon}, \mu>.\underline{0}$, where we note that the a action is not relabeled to b because it occurs in $/\{a\}$, from which two transitions labeled with τ, λ resp. b, μ can be generated, as expected.

As another example, consider term

$$((<a, \lambda>.\underline{0} + <b, \mu>.\underline{0}) \parallel_{\{a\}} (<a, *_{w_1}>.\underline{0} + <b, *_{w_2}>.\underline{0}))[\varphi],\ \varphi(a) = \varphi(b) = b$$

which can execute synchronization action $<a, \lambda>$, which is turned into $<b, \lambda>$, or actions $<b, \mu>$ and $<b, *_{w_2}>$. Similarly to the previous example, if we relabel $\parallel_{\{a\}}$ as $\parallel_{\{b\}}$ we erroneously introduce an additional synchronization between the two original b actions, while if we do not consider $\parallel_{\{a\}}$ at all we get rid of the synchronization between the two original a actions. In order to avoid the two cases above, we first rewrite the term as

$$((<a, \lambda>.\underline{0} + <b, \mu>.\underline{0}) \parallel_{\{(a,a)\}} (<a, *_{w_1}>.\underline{0} + <b, *_{w_2}>.\underline{0}))[\varphi]$$

This gives rise to

$$(<a, \lambda>.\underline{0} + <b^{\varepsilon,\{\bar{r}\},\varepsilon}, \mu>.\underline{0}) \parallel_{\{(a,b^{\varepsilon,\{r\},\varepsilon})\}} (<a, *_{w_1}>.\underline{0} + <b^{\varepsilon,\{\bar{r}\},\varepsilon}, *_{w_2}>.\underline{0})$$

which results in set of places $\{<b^{\varepsilon,\{r\},k}, \lambda>.\underline{0} + <b^{\varepsilon,\{\bar{r}\},\varepsilon}, \mu>.\underline{0}, <b^{\varepsilon,\{r\},\bar{k}}, *_{w_1}>.\underline{0} + <b^{\varepsilon,\{\bar{r}\},\varepsilon}, *_{w_2}>.\underline{0}\}$ from which the three expected transitions can be generated. As usual, the application of each syntactical substitution may be preceded by alpha conversion to avoid binding free names. This is necessary e.g. when decomposing term

$$(<a, \lambda>.\underline{0})/\{b\}[\varphi],\ \varphi(a) = \varphi(b) = b$$

B Definition of the Transition Generation

Formally, the set of net transitions $\longrightarrow_{\mathcal{N}}$ for a given EMPA_{gr} term is defined as the least subset of $\mathcal{P}(\mathcal{V}) \times (AType \times ARate^{\mathcal{M}(\mathcal{V})}) \times \mathcal{M}(\mathcal{V})$ generated by the axiom reported in Table B.1, where $n \in \mathbb{N}_+$ and $n_i \in \mathbb{N}_+$ for all $i = 1, \ldots, n$. It is worth observing that we have just one axiom for generating all the net transitions. This is quite different from the location oriented approach, where the net transitions are generated by means of a set of rules whose structure is similar to the structure of the rules of the interleaving semantics.

$$\boxed{\{\sum_{h=1}^{n_i} <a^{\theta,R_{i,h},\sigma_i}, \tilde{\lambda}_{i,h}>.E_{i,h} + F_i \mid 1 \leq i \leq n\} \xrightarrow{b,\tilde{\lambda}}_{\mathcal{N}} \bigoplus_{i=1}^{n} dec(E_{i,1})}$$

Table B.1
Axiom for generating net transitions

For each transition generated by the axiom, the preset is a finite nonempty multiset of places, from each of which a set of summands is factorized. If the preset contains more than one place, then the generated transition results from the synchronization of the actions of the factorized summands. The factorized summands must have the same action type a at the same visibility level θ; within the same place i, the factorized summands must have the same combinator string σ_i. The postset of the transition is the multiset composed of the places representing the decomposition of the derivative terms of the factorized summands in the preset. The axiom is subject to the following conditions:

(1) For all $i = 1, \ldots, n$ and $h, h' = 1, \ldots, n_i$

$$PL(\tilde{\lambda}_{i,h}) = PL(\tilde{\lambda}_{i,h'})$$
$$dec(E_{i,h}) = dec(E_{i,h'})$$

whereas

$$F_i \equiv \sum_{h=n_i+1}^{m_i} <a_{i,h}^{\theta_{i,h},R_{i,h},\sigma_{i,h}}, \tilde{\lambda}_{i,h}>.E_{i,h}$$

is such that for all $h = n_i + 1, \ldots, m_i$ at least one of the following holds

$$a_{i,h} \neq a$$
$$\theta_{i,h} \neq \theta$$
$$\sigma_{i,h} \neq \sigma_i$$
$$PL(\tilde{\lambda}_{i,h}) \neq PL(\tilde{\lambda}_{i,1})$$
$$dec(E_{i,h}) \neq dec(E_{i,1})$$

(2) For all $i, i' = 1, \ldots, n$, $h = 1, \ldots, n_i$, and $h' = 1, \ldots, n_{i'}$

$$i \neq i' \Longrightarrow \nexists r \in Conf.\, \{r, \bar{r}\} \subseteq R_{i,h} \cap R_{i',h'}$$

(3) $\bigodot_{i=1}^{n} \sigma_i = \{\!|\, \varepsilon \,|\!\}$

(4) $b = \begin{cases} a \text{ if } \theta = \varepsilon \\ \tau \text{ if } \theta = \tau \end{cases}$

(5) Let $V_j, 1 \leq j \leq n$ be the i-th place in the preset. If there exists one and only one place in the preset, say V_i, that contributes with nonpassive actions, then $\tilde{\lambda}$ is

$$\sum_{h=1}^{n_i} \tilde{\lambda}_{i,h} \cdot M_{\mathrm{curr}}(V_i) \cdot$$
$$\prod_{j=1, j\neq i}^{n} \frac{\sum_{h=1}^{n_j} \{\!|\, w | \tilde{\lambda}_{j,h} = *_w \,|\!\}}{\sum_{h=1}^{m_j} \{\!|\, w | a_{j,h} = a \wedge \theta_{j,h} = \theta \wedge \sigma_{j,h} = \sigma_j \wedge \tilde{\lambda}_{j,h} = *_w \wedge \nexists r \in Conf. \{r, \bar{r}\} \subseteq R_{j,h} \cap R_{j',h'} \,|\!\}} \cdot$$
$$\prod_{\sigma \in Comb^*} \frac{w(a,\theta,\sigma)}{W(a,\theta,\sigma)}$$

where:

- $w(a, \theta, \sigma)$ is obtained from the sums of the weights of all the passive actions having type a, visibility θ, no complementary conflicts, and the factorized combinator string, that occur in a subset of places of the preset of the transition at hand whose reduced combinator string is σ, by multiplying each such sum by the current marking of the corresponding place;
- $W(a, \theta, \sigma)$ is obtained from the sums of the weights of all the passive actions having type a, visibility θ, no complementary conflicts, and the factorized combinator string, that occur in a subset of places of the preset of an enabled transition containing V_i in its preset whose reduced combinator string is σ, by multiplying each such sum by the current marking of the corresponding place;
- $w(a, \theta, \sigma)/W(a, \theta, \sigma)$ is taken to be 1 whenever $W(a, \theta, \sigma) = 0$;
- the second and the third line have to be ignored if the preset is composed of one place only.

If all the places in the preset contribute with passive actions, then $\tilde{\lambda}$ is

$$\prod_{j=1, j\neq i}^{n} \frac{\sum_{h=1}^{n_j} \{\!|\, w | \tilde{\lambda}_{j,h} = *_w \,|\!\}}{\sum_{h=1}^{m_j} \{\!|\, w | a_{j,h} = a \wedge \theta_{j,h} = \theta \wedge \sigma_{j,h} = \sigma_j \wedge \tilde{\lambda}_{j,h} = *_w \wedge \nexists r \in Conf. \{r, \bar{r}\} \subseteq R_{j,h} \cap R_{j',h'} \,|\!\}} \cdot$$
$$\prod_{\sigma \in Comb^*} \frac{w(a,\theta,\sigma)}{W(a,\theta,\sigma)} \cdot$$
$$\sum_{\sigma \in Comb^*} W(a, \theta, \sigma)$$

where:

- $w(a, \theta, \sigma)$ is obtained from the sums of the weights of all the passive actions having type a, visibility θ, no complementary conflicts, and the factorized combinator string, that occur in a subset of places of the preset of the transition at hand whose reduced combinator string is σ, by multiplying each such sum by the current marking of the corresponding place;
- $W(a, \theta, \sigma)$ is obtained from the sums of the weights of all the passive actions having type a, visibility θ, no complementary conflicts, and the factorized combinator string, that occur in a subset of places of the preset of an enabled transition containing one of the places at hand in its preset whose reduced combinator string is σ, by multiplying each such sum by the current marking of the corresponding place;

- $w(a,\theta,\sigma)/W(a,\theta,\sigma)$ is taken to be 1 whenever $W(a,\theta,\sigma) = 0$;
- in the last sum, every involved place is counted once;
- if the preset is composed of one place only, then $\tilde{\lambda}$ is just passive with a weight equal to the sum of the weight of its factorized passive actions.

Condition (1) states that the summands factorized for each place in the preset of the transition are all and only those involving the same priority level and derivative terms having the same decomposition. We require terms to have the same decomposition instead of being syntactically identical because *dec* is not injective. Such a factorization, together with Condition (5), prevents identical transitions from collapsing by summing up their rates (GSPNs do not assign a multiplicity to the transitions). Condition (2) and (3) ensure that the places in the preset correctly synchronize by considering the decorations of the factorized actions. More precisely, Condition (2) avoids the generation of synchronization transitions that would erroneously stem from the application of a functional relabeling operator. Condition (3), instead, checks for the correct reduction of combinator strings to the empty string. Condition (4) determines the action type labeling the transition according to the functional abstraction related decoration θ of the selected summands.

As far as Condition (5) is concerned, we observe first of all that it allows only master-slave synchronizations, thus complying with the interleaving semantics for EMPA_{gr}. There are two cases, depending on whether there is a place in the preset that contributes with nonpassive actions or not. If one of the places in the preset contributes with nonpassive actions, then the rate of the transition is the product of three factors:

- The first factor is the sum of the rates of the actions that are factorized in that place, multiplied by the number of tokens currently in the place itself. As an example, if we consider term
$$((<a,\lambda>.\underline{0} + <a,\lambda>.\underline{0}) \parallel_{\emptyset} (<a,\lambda>.\underline{0} + <a,\lambda>.\underline{0})) \parallel_{\{a\}} <a,*_w>.\underline{0}$$
then its decomposition
$\{\!|\, <a^{\varepsilon,\emptyset,k},\lambda>.\underline{0} + <a^{\varepsilon,\emptyset,k},\lambda>.\underline{0}, <a^{\varepsilon,\emptyset,k},\lambda>.\underline{0} + <a^{\varepsilon,\emptyset,k},\lambda>.\underline{0}, <a^{\varepsilon,\emptyset,\bar{k}},*_w>.\underline{0} \,|\!\}$
is the preset of a transition of type a whose rate is given by
$$(\lambda + \lambda) \cdot M_{\text{curr}}(<a^{\varepsilon,\emptyset,k},\lambda>.\underline{0})$$
When the transition is fired for the first time, the actual rate is $4 \cdot \lambda$.
- The second factor is the product, computed over all the places in the preset contributing with passive actions, of the probabilities of selecting within each place those passive actions that are factorized. As an example, if we consider term
$$<a,\lambda>.\underline{0} \parallel_{\{a\}} (<a,*_{w_1}>.E_1 + <a,*_{w_2}>.E_2)$$
where $dec(E_1) \neq dec(E_2)$, then its decomposition
$$\{\!|\, <a^{\varepsilon,\emptyset,k},\lambda>.\underline{0}, <a^{\varepsilon,\emptyset,\bar{k}},*_{w_1}>.E_1 + <a^{\varepsilon,\emptyset,\bar{k}},*_{w_2}>.E_2 \,|\!\}$$

is the preset of two transitions of type a whose rates are given by

$$\lambda \cdot M_{\rm curr}(<a^{\varepsilon,\emptyset,k}, \lambda>.\underline{0}) \cdot \tfrac{w_1}{w_1+w_2}$$
$$\lambda \cdot M_{\rm curr}(<a^{\varepsilon,\emptyset,k}, \lambda>.\underline{0}) \cdot \tfrac{w_2}{w_1+w_2}$$

- The third factor takes into account the fact that the same place in the preset of the transition at hand contributing with nonpassive actions can be in the preset of several transitions of the same type. The places contributing with passive actions that belong to different presets of such transitions are independent of each other. All these places contribute to the computation of the rate of the transition at hand, so we need to consider them. In order to detect them, we look at the combinator string of their factorized actions, because the factorized passive actions of the same type occurring in different places and sharing the same combinator string come from the decomposition of independent terms composed in parallel. As an example, if we consider term

$$<a, \lambda>.\underline{0} \,\|_{\{a\}} (<a, *_{w_1}>.E_1 \,\|_\emptyset <a, *_{w_1}>.E_1 \,\|_\emptyset <a, *_{w_2}>.E_2)$$

where $w_1 \neq w_2$ or $E_1 \neq E_2$, then its decomposition

$$\{\!| <a^{\varepsilon,\emptyset,k}, \lambda>.\underline{0}, <a^{\varepsilon,\emptyset,\bar{k}}, *_{w_1}>.E_1, <a^{\varepsilon,\emptyset,\bar{k}}, *_{w_1}>.E_1, <a^{\varepsilon,\emptyset,\bar{k}}, *_{w_2}>.E_2 |\!\}$$

is the preset of two transitions of type a whose rates are given by

$$\lambda \cdot M_{\rm curr}(<a^{\varepsilon,\emptyset,k}, \lambda>.\underline{0}) \cdot \tfrac{w_1}{w_1} \cdot \frac{w_1 \cdot M_{\rm curr}(<a^{\varepsilon,\emptyset,\bar{k}}, *_{w_1}>.E_1)}{w_1 \cdot M_{\rm curr}(<a^{\varepsilon,\emptyset,\bar{k}}, *_{w_1}>.E_1) + w_2 \cdot M_{\rm curr}(<a^{\varepsilon,\emptyset,\bar{k}}, *_{w_2}>.E_2)}$$
$$\lambda \cdot M_{\rm curr}(<a^{\varepsilon,\emptyset,k}, \lambda>.\underline{0}) \cdot \tfrac{w_2}{w_2} \cdot \frac{w_2 M_{\rm curr}(<a^{\varepsilon,\emptyset,\bar{k}}, *_{w_2}>.E_2)}{w_1 \cdot M_{\rm curr}(<a^{\varepsilon,\emptyset,\bar{k}}, *_{w_1}>.E_1) + w_2 \cdot M_{\rm curr}(<a^{\varepsilon,\emptyset,\bar{k}}, *_{w_2}>.E_2)}$$

When the two transitions are fired for the first time, the actual rates are

$$(\lambda \cdot 1) \cdot \tfrac{w_1}{w_1} \cdot \tfrac{w_1 \cdot 2}{w_1 \cdot 2 + w_2 \cdot 1}$$
$$(\lambda \cdot 1) \cdot \tfrac{w_2}{w_2} \cdot \tfrac{w_2 \cdot 1}{w_1 \cdot 2 + w_2 \cdot 1}$$

If all the places in the preset contribute with passive actions only, then again the rate of the transition is the product of three factors. The first two factors coincide with the second and the third factor of the previous case, respectively, while the third factor is just the sum of the contributions of all the involved places (compare with factor W in rule (***PC3***) of Table 1).

We conclude by observing that the reason why the full EMPA$_{\rm gr}$ language has not been considered in this paper, i.e. reactive priority levels are not attached to passive actions, is due to the transition generation mechanism. If we consider the full EMPA$_{\rm gr}$ and we take term

$$<a, \lambda>.\underline{0} \,\|_{\{a\}} (<a, *_{l_1,w_1}>.E_1 \,\|_\emptyset <a, *_{l_2,w_2}>.E_2)$$

where $l_1 > l_2$, then its decomposition

$$\{\!| <a^{\varepsilon,\emptyset,k}, \lambda>.\underline{0}, <a^{\varepsilon,\emptyset,\bar{k}}, *_{l_1,w_1}>.E_1, <a^{\varepsilon,\emptyset,\bar{k}}, *_{l_2,w_2}>.E_2 |\!\}$$

would the preset of two exponentially timed transitions of type a. Here the problem is that both transitions would be enabled, but only the former could

be actually fired and the GSPN firing rule does not allow us to recognize that. We conjecture that the introduction of suitable inhibitor arcs should make it possible the generation of transitions for the full EMPA_{gr}.

C Proof of the Retrievability Result

Let us first introduce the following notation: given a LTS whose label set is Act and a partition $\mathcal{R}$ of its state space S, we pose for all $s \in S$, $a \in AType$, $l \in \mathbb{N} \cup \{-1\}$, and $C \in S/\mathcal{R}$

$$Rate(s, a, l, C) = \sum \{\!|\, \tilde{\lambda} \mid \exists s' \in C.\, s \xrightarrow{a,\tilde{\lambda}} s' \wedge PL(\tilde{\lambda}) = l \,|\!\}$$

Given such a notation, and considered the closed interleaving semantics for EMPA_{gr} (i.e. lower priority nonpassive transitions are pruned), we have to prove the following: whenever $(E, Q) \in \mathcal{B}$, then for all $a \in AType$, $l \in \mathbb{N} \cup \{-1\}$, and $C \in (\mathcal{G} \cup \mathcal{M}(\mathcal{V}))/\mathcal{B}'$

$$Rate(E, a, l, C) = Rate(Q, a, l, C)$$

Let $(E, Q) \in \mathcal{B}$. We first show that $Rate(E, a, l, C) \leq Rate(Q, a, l, C)$ for all $a \in AType$, $l \in \mathbb{N} \cup \{-1\}$, and $C \in (\mathcal{G} \cup \mathcal{M}(\mathcal{V}))/\mathcal{B}'$. If $Rate(E, a, l, C) = 0$, i.e. the summation is computed on an empty set, then there is nothing to prove, otherwise we proceed by induction on the maximum depth d of the derivations of the transitions for E having type a, priority level l, and derivative term in C.

- If $d = 1$ then only rule (***GAC***) of Table 1 has been used to deduce the existing transitions. Therefore $E \equiv \sum_{i \in I} <a_i, \tilde{\lambda}_i>.E_i$ with $PL(\tilde{\lambda}_j) = l$ and $E'_j \in C$ for some $j \in I$. From the definition of $\mathcal{B}$ it follows that $Q = \{\!|\, \sum_{i \in I} <a_i, \tilde{\lambda}_i>.E_i \,|\!\}$, hence the result trivially follows.
- If $d > 1$ then several subcases arise depending on the syntactical structure of E.
  - If $E \equiv E'/L$ then $Q = dec(E'\{a^{\tau,\emptyset,\varepsilon}/a \mid a \in L\})$. Since d is the maximum depth of the derivations of the transitions for E having type a, priority level l, and derivative term in C, either E' has no transitions having type a, priority level l, and derivative term in C' (where $C = C'/L$), or $d - 1$ is the maximum depth of the derivations of the transitions for E' having type a, priority level l, and derivative term in C'. Since

$$Rate(E, a, l, C) = \begin{cases} Rate(E', a, l, C') \\ \qquad \text{if } a \notin L \cup \{\tau\} \\ Rate(E', \tau, l, C') + \sum \{\!|\, Rate(E', b, l, C') \mid b \in L \,|\!\} \\ \qquad \text{if } a = \tau \end{cases}$$

and similarly for Q and $dec(E')$, the result follows by the induction hypothesis.

- If $E \equiv E'[\varphi]$ then the proof is similar to the one developed in the previous subcase.
- If $E \equiv E_1 \parallel_S E_2$ then $Q = dec(E_1\{d^{\theta,R,\sigma k_a}/a \mid (a, d^{\theta,R,\sigma}) \in s \wedge k_a \text{ fresh}\}) \oplus dec(E_2\{d^{\theta,R,\sigma \bar{k}_a}/a \mid (a, d^{\theta,R,\sigma}) \in s \wedge k_a \text{ fresh}\})$.

  Let $a \notin S$. Since d is the maximum depth of the derivations of the transitions for E having type a, priority level l, and derivative term in C, for each $j \in \{1, 2\}$ either E_j has no transitions having type a, priority level l, and derivative term in C_j (where $C = C_1 \parallel_S C_2$), or $d - 1$ is the maximum depth of the derivations of the transitions for E_j having type a, priority level l, and derivative term in C_j. Since

$$Rate(E, a, l, C) = \begin{cases} Rate(E_1, a, l, C_1) + Rate(E_2, a, l, C_2) \\ \qquad \text{if } E_1 \in C_1 \wedge E_2 \in C_2 \\ Rate(E_1, a, l, C_1) \\ \qquad \text{if } E_1 \notin C_1 \wedge E_2 \in C_2 \\ Rate(E_2, a, l, C_2) \\ \qquad \text{if } E_1 \in C_1 \wedge E_2 \notin C_2 \\ 0 \\ \qquad \text{if } E_1 \notin C_1 \wedge E_2 \notin C_2 \end{cases}$$

  and similarly for Q, $dec(E_1)$, and $dec(E_2)$, the result follows by the induction hypothesis.

  Let $a \in S$. Since d is the maximum depth of the derivations of the transitions for E having type a, priority level l, and derivative term in C, let $C = C_1 \parallel_S C_2$ we have that if $Rate(E_2, a, -1, C_2) \neq 0$ ($Rate(E_1, a, -1, C_1) \neq 0$) then either E_1 (E_2) has no transitions having type a, priority level l, and derivative term in C_1 (C_2), or $d-1$ is the maximum depth of the derivations of the transitions for E_1 (E_2) having type a, priority level l, and derivative term in C_1 (C_2). If the resulting transition is nonpassive, since

$$Rate(E, a, l, C) = \begin{cases} Rate(E_1, a, l, C_1) \cdot p_{E_2} + Rate(E_2, a, l, C_2) \cdot p_{E_1} \\ Rate(E_1, a, l, C_1) \cdot p_{E_2} \\ Rate(E_2, a, l, C_2) \cdot p_{E_1} \\ 0 \end{cases}$$

  depending on whether $Rate(E_2, a, -1, C_2) \neq 0 \wedge Rate(E_1, a, -1, C_1) \neq \perp$ or $Rate(E_2, a, -1, C_2) \neq 0 \wedge Rate(E_1, a, -1, C_1) = 0$ or $Rate(E_2, a, -1, C_2) = 0 \wedge Rate(E_1, a, -1, C_1) \neq 0$ or $Rate(E_2, a, -1, C_2) = 0 \wedge Rate(E_1, a, -1, C_1) = 0$, where p_{E_1} (p_{E_2}) is the ratio of the total

weight of the passive transitions of type a from E_1 (E_2) to C_1 (C_2) to the total weight of the passive transitions of type a from E_1 (E_2), and similarly for Q, $dec(E_1)$, and $dec(E_2)$, the result follows by the induction hypothesis. If the resulting transition is passive, we reason in a similar way.

- Let $E \equiv A$ with $A \stackrel{\Delta}{=} E'$. Since d is the maximum depth of the derivations of the transitions for E having type a, priority level l, and derivative term in C, $d-1$ is the maximum depth of the derivations of the transitions for E' having type a, priority level l, and derivative term in C. From the induction hypothesis it follows that $Rate(E,a,l,C) = Rate(E',a,l,C) \leq Rate(dec(E'),a,l,C) = Rate(Q,a,l,C)$.

To show the reverse inequality, we similarly proceed by induction on the number of applications of function dec to obtain Q from E.