Discrete Time Generative-Reactive Probabilistic Processes with Different Advancing Speeds

M. Bravetti *, A. Aldini

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

Abstract

We present a process algebra expressing probabilistic external/internal choices, multi-way synchronizations, and processes with different advancing speeds in the context of discrete time, i.e. where time is not continuous but is represented by a sequence of discrete steps as in Discrete Time Markov Chains (DTMCs). To this end, we introduce a variant of CSP that employs a probabilistic asynchronous parallel operator whose synchronization mechanism is based on a mixture of the classical generative and reactive models of probability. In particular, differently from existing discrete time process algebras, where parallel processes are executed in synchronous locksteps, the parallel operator that we adopt allows processes with different probabilistic advancing speeds (mean number of actions executed per time unit) to be modeled. Moreover, our generative-reactive synchronization mechanism makes it possible to always derive DTMCs in the case of fully specified systems. We then present a sound and complete axiomatization of probabilistic bisimulation over finite processes of our calculus, that is a smooth extension of the axiom system for a standard process algebra, thus solving the open problem of cleanly axiomatizing action restriction in the generative model. As a further result, we show that, when evaluating steady state based performance measures which are expressible by attaching rewards to actions, our approach provides an exact solution even if the advancing speeds are considered not to be probabilistic, without incurring the state space explosion problem that arises with standard synchronous approaches. We finally present a case study on multi-path routing showing the expressiveness of our calculus and that it makes it particularly easy to produce scalable specifications.

Key words: Process Algebra, Discrete Time Markov Chain, Performance Evaluation, Probabilistic Models

* Corresponding author.
Email address: bravetti@cs.unibo.it (M. Bravetti).

1 Introduction

The modeling experience in the specification of complex concurrent systems (see e.g. [3,4,10]) has revealed the importance of using languages expressing time (the advancing speed of processes), probabilistic internal/external choices and multi-way synchronizations to be able to represent the behavior of such systems.

In this paper we address the open problem of developing a calculus for describing and analyzing probabilistic distributed systems that combines, in a natural way, such mechanisms in a discrete time setting. More precisely, we make the assumption that time does not flow continuously, as e.g. in Continuous Time Markov Chains (CTMCs) [22], but the passage of time is represented by a sequence of discrete steps, as in Discrete Time Markov Chains (DTMCs) [22]. In particular, we recall that DTMCs are labeled transition systems, where the transitions are labeled with their execution probabilities, that represent timed systems performing a transition every discrete time step of the same length 1.

1.1 Generative-Reactive Probabilistic Models for Discrete Time Systems

In the past 10 years several models for representing probabilistic systems (see e.g. [15,28] and the references therein) and several probabilistic extensions of classical process algebras (see e.g. [21,24,31,5,15,14]) based on such models have been proposed. In particular, we briefly recall two different models of probability, which have been presented in [15], by resorting to the terminology of Milner [26], where action type based synchronizations are described in terms of *button pushing experiments*. In this view the environment experiments on a process by pushing one of several buttons, where a button represents an action type. According to the *reactive model* of probability, a process reacts internally to a button push performed by its environment on the basis of a probability distribution that depends on the button that is pushed. According to the *generative model* of probability, instead, the process itself autonomously decides, on the basis of a probability distribution, which button will go down and how to behave after such an event.

When two processes, which behave in a reactive way, synchronize on an action of type a, each of them reacts internally to the synchronization according to the probability distribution associated with the actions of type a it can perform. Whenever the two processes can synchronize on more than one action type, each of them leaves the decision to the environment, hence the choice of the synchronizing action type turns out to be non-deterministic. This kind of synchronization is simple and natural [28], but does not allow a mechanism for the choice of the button to be pushed (external choice) to be expressed, thus leaving the system, in a sense, under-specified.

On the other hand, two processes that behave in a generative way independently decide the action type over which they want to synchronize, hence there may be no agreement on the action type. Almost all the solutions proposed in the literature (see e.g. [15,5]) are based on the fact that the synchronization on different action types is allowed. In the case the model of synchronization adopted is such that processes can synchronize only on the same action type then, as also stated in [28], there is no simple and intuitive solution to the problem. [1] As an example, let us consider a system composed of two sequential processes that must synchronize over actions of type a and b. Suppose that one process may execute an action a with probability p and an action b with probability $1 - p$, while the other process may execute an action a with probability q and an action b with probability $1 - q$. This scenario could be represented by a term like $a +^p b \,\|_{\{a,b\}}\, a +^q b$. The problem is: How do we choose the synchronization to be performed? The choice cannot be made independently by the two processes, because e.g. if the lefthand process locally chooses a and the righthand process locally chooses b, then no synchronization is possible between the two processes. In order to have a clear model of synchronization, it is important that, once chosen the set of processes that must cooperate, each of such processes can *independently* choose one of its local actions. The problem is that we do not know who is in control of the system since both processes want to decide the button to be pushed.

A simple and intuitive solution to the problem (as suggested also in [28]) is to adopt a mixed generative-reactive approach by considering an *asymmetric* form of synchronization, where a process that behaves generatively may synchronize only with processes that behave reactively. The intuition is that the process that behaves generatively decides which button will go down (and how to behave afterwards), while the process that behaves reactively just reacts internally to the button push of the other process. The integration of the generative and reactive approaches is naturally obtained (similarly as done in [32] for a probabilistic version of I/O automata [25]) by designating some actions, called *generative* actions, as predominant over the other ones, called *reactive* actions, and by imposing that generative actions can synchronize with reactive actions only. Syntactically, we distinguish reactive actions from generative actions by using the subscript $*$ stating the reactive nature of the action. As an example, let us consider a system composed of two sequential processes that must synchronize over actions of type a and b whose behavior could be described by a term like:

$$(a +^p b) \,\|_{\{a,b\}} ((a_* +^q a_*) + (b_* +^r b_*))$$

[1] In spite of this, a rather artificial solution has been proposed in [14].

According to the generative-reactive approach: first we probabilistically choose, locally to the lefthand process, between the two generative actions a and b according to probabilities p and $1 - p$ (generative choice); then we probabilistically choose, locally to the righthand process, either between the two reactive actions a according to probabilities q and $1 - q$ or between the two reactive actions b according to probabilities r and $1 - r$, depending on whether generative action a or b wins in the first step (reactive choice).

The discrete time probabilistic transition systems that we consider, called *generative-reactive* transition systems (GRTSs), are a mixture of the generative and reactive transition systems of [15]. In a GRTS (a restricted version of the general model of [28]), each transition represents a discrete time step and is labeled with an action, which can be either a generative action a or a reactive action a_*, and a probability. Transitions leaving a state are grouped in several bundles. We have a single generative bundle composed of all the transitions labeled with a generative action and several reactive bundles, each one referring to a different action type a and composed of all the transitions labeled with a_*. A bundle of transitions expresses a probabilistic choice, hence the sum of the probabilities of the transitions composing a bundle must be 1. On the contrary, the choice among bundles is performed non-deterministically. In a GRTS we see the reactive actions as *incomplete* actions, which must synchronize with generative actions of another system component in order to form a complete system. A system is considered to be *fully specified* only when it gives rise to a probabilistic transition system that is purely generative, in the sense that it does not include reactive bundles. Fully specified systems are therefore fully probabilistic systems (systems not including non-deterministic choices [28]), from which a DTMC can be trivially derived by discarding actions from transition labels. As a consequence, they can be easily analyzed to get performance measures. In other words, the limited form of non-determinism that we consider is sufficient to have a clear notion of control over action synchronizations, but is harmless from the performance analysis viewpoint, in the sense that for complete systems the underlying probabilistic transition system is fully probabilistic. Moreover, the modeling experience [9,3,4,10,8] showed that considering, as in the generative-reactive approach, two kinds of actions where one predominates over the other, leads to a master-slave discipline that is very suitable for describing real systems.

1.2 Probabilistic Asynchronous Parallel Composition

The parallel operator that we consider is similar to the CSP [19] operator $P \,\|_S\, Q$, where processes P and Q are required to synchronize over actions of the same type in the set S and locally execute all the other actions. Such an operator expresses multi-way synchronizations by assuming that the result of the synchronization of two actions a is again a visible action a. In addition, we impose that a synchronization between two actions of type a may occur only if either they are both reactive actions a_* (and the result is a reactive action a_*), or one of them is a generative action a and the other one is a reactive action a_* (and the result is a generative action a). As a consequence, a multi-way synchronization is composed of all reactive actions except at most one generative action: the choice of a generative action a performed by a process determines the button that is pushed (button a) and the other processes internally react by *independently* choosing one of their reactive actions a_*.

Differently from existing discrete time process algebras, where parallel processes are executed in synchronous locksteps (see e.g. [21,31,24]), the parallel composition operator that we adopt is asynchronous and allows processes with different *probabilistic advancing speeds* (mean number of actions executed per time unit) to be modeled.

Our approach is inspired by that of [5] where ACP [6] is extended with probability by means of a probabilistic parallel composition operator, even if in [5] time is not explicitly considered. In particular, as in [5], our parallel composition operator is parameterized with a probability p, so that in $P \,||_S^p\, Q$ the process performing the next move is determined according to probability p. As an example, let us consider a system composed of two sequential processes whose behavior is described by a term like: [2]

$$(a +^p b) \,||^q (c +^r d)$$

According to the approach inspired by [5], we first choose which of the two processes must make the next move according to probabilities q and $1 - q$. Then, if the lefthand process wins we locally choose between a and b according to probabilities p and $1 - p$, otherwise we locally choose between c and d according to probabilities r and $1 - r$.

As we will shortly show, in our discrete time setting $P \,||_S^p\, Q$ represents a system where the mean action frequency (mean number of actions executed per time unit) of process P is p, while the mean action frequency of process Q is $1-p$. Since P and Q may advance at different action frequencies, with respect to the classical synchronous approach, modeling a concurrent system does not necessarily imply adopting the same duration for the actions of P and Q. For instance, we could model a post office with a priority mail and an ordinary mail service (see Fig. 1) simply by the term $P \,||_\emptyset^{0.2}\, Q$, where: process P, representing the ordinary mail service, repeatedly executes actions a expressing the delivery of a letter via ordinary mail; and process Q, representing the priority mail service, repeatedly executes actions b expressing the delivery of a letter via

[2] We use $||^q$ instead of $||_S^q$ when $S = \emptyset$.

priority mail. Supposed that we take minutes to be the time unit on which the post office specification is based, in $P \,\|_{\emptyset}^{0.2}\, Q$ the mean frequency for the ordinary mail service is 0.2 letters per minute (288 per day) and the mean frequency for the priority mail service is 0.8 letters per minute (1152 per day). Therefore the actions of P take 5 minutes on average to be executed, while the actions of Q take 1 minute and 15 seconds to be executed.

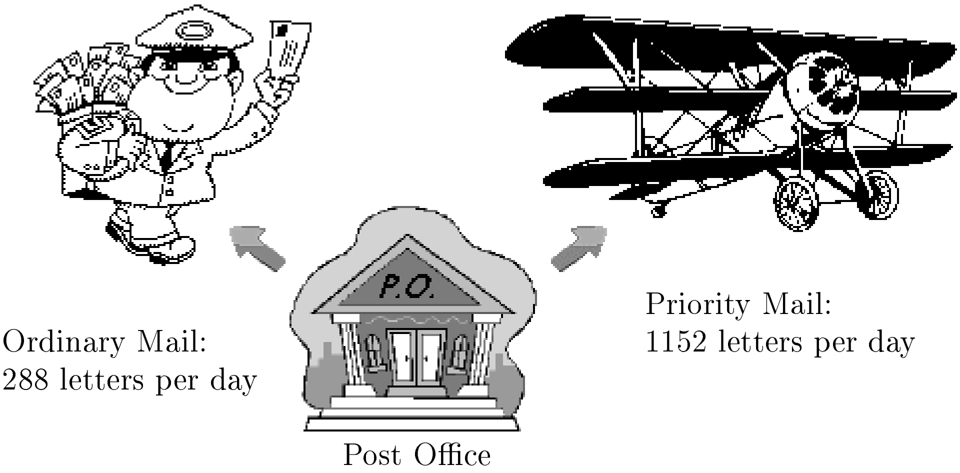


Fig. 1. The Post Office Example

To be more precise, the execution of a system $P \,\|_S^p\, Q$ is determined by assuming a probabilistic scheduler that in each global state decides which process between P and Q will perform the next step. In particular, P and Q advance in discrete steps and the scheduler decides who is going to perform the next move by tossing an unfair coin that gives "head" with probability p and "tail" with probability $1-p$. If the coin gives "head" P moves, if the coin gives "tail" Q moves. After a certain number, let us say n, of coin tosses, i.e. after n time units, it turns out that the mean number of heads that have occurred (steps P has made) is $n \cdot p$ while the mean number of tails that have occurred (steps Q has made) is $n \cdot (1-p)$. Formally, such mean values are derived in the following way: $n \cdot p$ is the mean value of a discrete random variable following a binomial distribution with parameters n (number of experiments) and p (probability of success for each experiment). This means that P performs a mean of p steps per time unit and Q performs a mean of $1-p$ steps per time unit. Hence p is P's probabilistic advancing speed while $1-p$ is Q's probabilistic advancing speed.

Note that in the representation of the behavior of a system $P \,\|_S^p\, Q$ we do not express an actual concurrent execution of the actions of P and Q (so that when time passes for an action of P then it passes also for a concurrent action of Q). The behavior of a system $P \,\|_S^p\, Q$ is, instead, described at a higher level of abstraction by a model with discrete time steps where there is no actual parallel execution, but only an interleaving of the steps of the two processes, where each step takes the same amount of time (a time unit). To make this more clear, we can interpret the behavior of $P \parallel_S^p Q$ as originated by a single-processor machine executing both processes (P and Q) via a probabilistic scheduler. In this view, the choices in the global states of $P \parallel_S^p Q$ do not represent "races" between concurrent time delays (as it is usual for continuous time process algebras) but only probabilistic choices that determine which is the process performing the next discrete step. Therefore we do not assume continuous memoryless distributed sojourn times as, e.g., in the time model of [32], but we simply execute a system transition every discrete time unit. In the execution of $P \parallel_S^p Q$ sometimes we perform a discrete move of P and sometimes we perform a discrete move of Q and what matters, from the viewpoint of performance behavior, is the frequency with which the actions of a given process are executed. Therefore, even if we represent the system behavior at a certain level of abstraction where actions are not concurrently executed but just interleaved, we have that such a representation gives the correct execution frequencies for actions of processes. Representing the system behavior just taking care of execution frequencies (and not actual concurrency) is the level of abstraction necessary in order to evaluate performance properties in discrete time. This because, since the sojourn times in the states of a DTMC before taking a transition are all equal to 1, as opposed to the states of a CTMC, the evaluation of performance measures in a DTMC is entirely based on the execution frequency of transitions.

So far we have explained the behavior of parallel composition in the case of processes performing independent moves. Now we show what happens when we consider also generative-reactive synchronizations. Our approach integrates the generative-reactive approach inspired by [32] with the approach to probabilistic process choice inspired by [5], in that the selection of the action to be executed in a system state is conceptually carried out through two steps (a generative choice determining the action type followed by a reactive choice) each employing probabilistic choice among processes. As an example, let us consider a system composed of four sequential processes whose behavior is described by the term:

$$(a \parallel^p (b +^{p'} c)) \parallel_{\{b\}} (b_* \parallel^q (b_* +^{q'} b_*))$$

In the first step a generative action is chosen as follows. We first choose which of the two lefthand processes must make a move according to probabilities p and $1-p$. Then, if the process executing a wins then a is the winning generative action. Otherwise we choose between b and c according to probabilities p' and $1-p'$. If the generative action chosen in the first step must not synchronize (as for the actions a and c) then we are done, otherwise (as for the action b) a second step must be undertaken, where we choose a reactive action among those that can synchronize with the winning generative action. We first choose which of the two righthand processes must make a move according to probabilities q and $1-q$. Then, if the process executing a single b_* wins then this is the winning reactive action. Otherwise we choose between the two reactive actions b_* of the other process according to probabilities q' and $1-q'$. Note that extending a generative-reactive approach similar to that of [32] with a probabilistic parallel composition allows us to obtain fully probabilistic models, nevertheless remaining in a discrete time probabilistic setting. The approach of [32], instead, strictly relies on the fact that continuous time is considered. This because by endowing each state of a probabilistic I/O automata with an exponentially distributed sojourn time, the relative advancing speeds of the probabilistic I/O automata involved in a parallel composition are determined by a timed race condition. Moreover, expressing advancing speeds of processes via a parameterized parallel operator (instead of, e.g., via weights attached to the actions they can perform [31]) is also adequate from a modeling viewpoint. This because the modeler can first specify the behavior of processes in isolation and then establish, independently on how they are specified, their relative advancing speed when composing them in parallel. Finally, as we will show by means of a case study, this approach is also convenient because it leads to specifications that are easily scalable.

1.3 Modeling Exact Advancing Speeds

We also consider the problem of modeling timed systems where the different advancing speeds at which concurrent processes proceed are not probabilistic. According to this interpretation of the notion of advancing speed, if, e.g., the action frequency of a process P in a parallel composition is $1/n$, with n natural number, then each action of P takes exactly n time units to be executed, instead of n time units on average.

With a standard approach based on a synchronous parallel composition, where parallel processes are executed in synchronous locksteps (see, e.g., [21,27,24,31,17]), exact advancing speeds could be modeled as follows. Called n_P the duration of the actions of a process P (number of time units taken to execute each action of P), we compute the greater common divisor div of the set of durations n_P where P is a process composing the system. Then for each process P we split each action in n_P/div time units and we take div to be the duration of the time unit in the specification of the whole system. Such an approach has the problem that the state space of the system greatly increases due to the splitting of the actions of processes, especially if actions with largely different durations are employed: since the step reached by a timed action during its execution needs to be counted in the semantics, each action of P gives rise to

n_P/div states. Other approaches in the literature that avoid action splitting (see, e.g., [12,2,13]) do not give rise to models that can be turned into analyzable stochastic processes. The reason is that they either (as in [12]) produce dense-time models with infinite states, or (as in [2] and [13]) produce models in which the time progress is separately represented within each different process.

Our approach, which is based on probabilistic advancing speeds, constitutes an approximated solution to the problem of modeling timed systems with different exact advancing speeds, in that action frequencies of processes are probabilistic instead of being exact, but it has the advantage that actions are not split. Nevertheless, an important property of our approach is that, in the case of non-blocking processes (i.e. processes enabling at least a generative action in each state), while such an approximation may affect the performance behavior of the system during an initial transient evolution, it gives exact values for the performance measures of interest when the system reaches a stationary behavior. Note that the restriction to non-blocking processes is not a limiting requirement from the modeling viewpoint, because the waiting condition of a process can be expressed by executing generative *idle* actions representing the fact that the process is idle. Therefore, as far as the evaluation of steady state based performance measures of systems is concerned (at least if they are expressible by associating rewards with actions [8]), our approach avoids action splitting, hence the state space explosion problem, while preserving the possibility of exactly analyzing concurrent processes with exact advancing speeds.

1.4 Overview and Main Results

We start by formally presenting the probabilistic model for discrete time, based on a mixture of the generative and reactive models of probability, we explained above. In particular we show that, from the models of fully specified systems, we can easily derive DTMCs (Sect. 2).

We then introduce a calculus based on the generative-reactive model and the probabilistic asynchronous parallel composition operator explained above. In particular we present its syntax, its formal semantics and we show that a simple extension of the probabilistic bisimulation equivalence [23] is a congruence with respect to all the operators of the calculus and recursive definitions (Sect. 3).

Afterwards, we address the problem of finding a sound and complete axiomatization for the probabilistic bisimulation equivalence over the finite processes of our calculus. In order to do this we introduce a new auxiliary operator

$\langle P \rangle^J_K$, defined in such a way that if $\langle P \rangle^J_K$ is derived from P by our axiom system, then K denotes which bundles are executable by P and J denotes the set of types of the generative actions executable by P. Thanks to this operator we obtain an axiomatization that is a smooth probabilistic extension of the classical axiomatization for the parallel composition based on the left and synchronization merge operators [5]. In particular, one of the most important consequences of adopting such an operator is that it leads to a clean axiomatization of action restriction in the generative model, without resorting to axioms with implications as in [5]. Moreover, the axioms for the reactive actions are exactly like the axioms for the standard actions except for the calculations related to probability (Sect. 4).

We subsequently show that when evaluating performance measures of a system that are based on its behavior at a steady state, our approach provides a correct solution even when the advancing speeds are considered to be exact. In particular, with respect to the standard approach based on a synchronous parallel composition, we avoid the state space explosion problem mentioned above (Sect. 5).

Finally, we present a case study that shows all the main features of our calculus: the algebraic model of a router implementing a probabilistic multi-path routing mechanism (Sect. 6). In this case study we show that: (*i*) our approach makes it possible to analyze systems whose components are specified through actions with largely different exact durations (in our model we have actions lasting from 0.5 microseconds to 20 milliseconds); (*ii*) expressing advancing speeds of processes via a parameterized parallel operator (instead of, e.g., via weights attached to the actions they can perform) is convenient from a modeling viewpoint because the modeler can first specify the behavior of processes in isolation and then establish, independently of how they are specified, their advancing speeds when composing them in parallel; (*iii*) thanks to the use of our probabilistic parallel operator and to the generative-reactive mechanism, it is possible to define a specification of the router that is easily scalable to an arbitrary size of the routing table.

To terminate the paper, we report some concluding remarks (Sect. 7).

2 The Generative-Reactive Model

GRTSs are a mixture of the generative and reactive transition systems of [15]. In a GRTS (a restricted version of the general model of [28]), each transition is labeled with an action, which can be either a generative action a or a reactive action a_*, and a probability. Formally, we denote the set of action types by $AType$, ranged over by a, $b, \ldots$. As usual $AType$ includes the special type τ denoting internal actions. We denote the set of reactive actions by $RAct = \{a_* \mid a \in AType\}$ and the set of generative actions by $GAct = AType$. The set of actions is denoted by $Act = RAct \cup GAct$, ranged over by θ, θ', ….

Transitions leaving a state are grouped in several bundles. We have a single generative bundle composed of all the transitions labeled with a generative action and several reactive bundles, each one referring to a different action type a and composed of all the transitions labeled with a_*. A bundle of transitions expresses a probabilistic choice. On the contrary the choice among bundles is performed non-deterministically.

Definition 2.1 A probabilistic transition system PTS is a quadruple $(S, AType, T, s_1)$ with

- S a set of states,
- $AType$ a set of action types,
- $T \in \mathcal{M}(S \times Act \times]0,1] \times S)$ a multiset [3] of probabilistic transitions,
- and $s_1 \in S$ the initial state. ■

Definition 2.2 A $GRTS$ is a PTS $(S, AType, T, s_1)$ such that

(1) $\forall s \in S, \forall a_* \in RAct,\ \sum\{\!|\, p \mid \exists t \in S : (s, a_*, p, t) \in T \,|\!\} \in \{0, 1\}$
(2) $\forall s \in S,\ \sum\{\!|\, p \mid \exists a \in GAct, t \in S : (s, a, p, t) \in T \,|\!\} \in \{0, 1\}$ ■

The first requirement defines the structure of the reactive bundles leaving a state (one for each action type), and the second requirement says that each state must have a unique generative bundle. Both requirements say that for each state the probabilities of the transitions composing a bundle, if there are any, sum up to 1 (otherwise the summation over empty multisets is defined equal to 0).

As an example, we show in Fig. 2 a $GRTS$ composed of a generative bundle which enables three transitions (a, b, and c), a reactive bundle of type b which enables two transitions b_*, and a reactive bundle of type d. As in [28] we denote transitions of the same bundle by grouping them by an arc, and we omit the probability from a transition label when it is equal to 1.

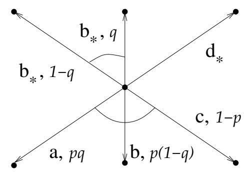


Fig. 2. Example of a Generative-Reactive Transition System

[3] We use "$\{\!|$" and "$|\!\}$" as brackets for multisets and $\mathcal{M}(S)$ to denote the collection of multisets over set S.

We recall that reactive actions are seen as *incomplete* actions which must synchronize with generative actions of another system component in order to form a complete system. Therefore a fully specified system is *performance closed*, in the sense that it gives rise to a fully probabilistic transition system which does not include reactive bundles. From such a transition system a DTMC can be trivially derived by discarding actions from transition labels. Such a DTMC can, then, be easily analyzed to get performance measures of systems. Formally, a (homogeneous) DTMC [22] on a set of states S is defined by a transition matrix $\mathbf{P} = [p_{i,j}]_{i,j \in S}$, where $p_{i,j}$ represents the probability of going from the state i to the state j, and a vector of initial probabilities which associates with each state $i \in S$ the probability of being the starting state. From a *GRTS* $(\{s_1, \ldots, s_n\}, \mathit{AType}, T, s_1)$ not including reactive bundles we derive the corresponding DTMC $\mathbf{P} = [p_{i,j}]_{i,j \in \{1,\ldots,n\}}$ (whose vector of initial probabilities assigns probability 1 to state s_1 and probability 0 to all the other states) as follows:

$$p_{i,j} = \sum \{\!|\, p \mid \exists\, a \in \mathit{GAct} : (s_i, a, p, s_j) \in T \,|\!\}$$

If the system under analysis reaches a steady behavior at a limiting execution time, we can evaluate its performance by studying the (time averaged) stationary state probabilities of the underlying DTMC $\mathbf{P}$ [22,30]. This is done by computing the vector of state probabilities $\boldsymbol{\pi} = [\pi_i]_{i \in \{1,\ldots,n\}}$ (such that $\sum_{i \in \{1,\ldots,n\}} \pi_i = 1$) that solves the global balance equation:

$$\boldsymbol{\pi} \cdot \mathbf{P} = \boldsymbol{\pi}$$

The i-th element π_i of the steady state probability vector $\boldsymbol{\pi}$ represents the probability that $\mathbf{P}$ is in state i when observed at a steady behavior.

Analyzing the system for a particular performance measure can be done by associating (bonus) *rewards* with the generative actions occurring in the system algebraic specification [20]. In particular we express the performance measure that we want to evaluate in terms of the reward r_a earned by the execution of an action of type a according to such a performance measure. In order to compute the value m of the performance measure we have to evaluate the reward structure $\mathbf{B} = [b_{i,j}]_{i,j \in \{1,\ldots,n\}}$ associated to the DTMC $\mathbf{P}$ underlying the system. This is done as follows:

$$b_{i,j} = \sum \{\!|\, r_a \cdot \frac{p}{p_{i,j}} \mid \exists\, a \in \mathit{GAct} : (s_i, a, p, s_j) \in T \,|\!\}$$

where $p/p_{i,j}$ is the probability that the transition with reward r_a is performed given that we perform a transition from state i to state j. We use such probabilities to aggregate the rewards of several transitions going from state i to state j into a single reward.

Finally, the performance measure m is computed as follows:

$$m = \sum_{i \in \{1,\ldots,n\}} \sum_{j \in \{1,\ldots,n\}} \pi_i \cdot b_{i,j} \cdot p_{i,j}$$

3 The Calculus: Syntax and Semantics

In this section we introduce the generative-reactive calculus. We begin by presenting the syntax of the calculus and a detailed explanation of the probabilistic parallel composition. We then introduce the calculus operational semantics which generates GRTSs from process terms. Finally, we present a notion of strong probabilistic bisimulation, which is shown to be a congruence for the calculus.

3.1 Syntax

The syntax of the generative-reactive process algebra is defined as follows.

Let *Const* be a set of constants, ranged over by $A, B, \ldots$.

Definition 3.1 The set $\mathcal{L}$ of process terms is generated by the syntax:

$$P ::= \underline{0} \mid \theta.P \mid P +^p P \mid P \parallel^p_S P \mid P[a \to b]^p \mid A$$

where $a \in \mathit{AType} - \{\tau\}, b \in \mathit{AType}, S \subseteq \mathit{AType} - \{\tau\}$, $p \in]0,1[$. The set $\mathcal{L}$ is ranged over by P, $Q, \ldots$. We denote by $\mathcal{G}$ the set of guarded and closed terms of $\mathcal{L}$. ■

$\underline{0}$ represents a terminated or deadlocked term having no transitions, while the prefix operator $\theta.P$ performs the action θ with probability 1 and then behaves like P.

The alternative composition operator $P +^p Q$ represents a probabilistic choice between the generative actions of P and Q and between the reactive actions of P and Q of the same type. As far as generative actions are concerned, $P +^p Q$ executes a generative action of P with probability p and a generative action of Q with probability $1 - p$. In the case one process P or Q cannot execute generative actions, $P +^p Q$ chooses a generative action of the other process with probability 1 (similarly as in [5]). As far as reactive actions of a given type a are concerned, $P +^p Q$ chooses between the reactive actions a_* of P and Q according to probability p, by following the same mechanism.

Example 3.2 As an example, [4] $a +^p b$ and $b_* +^p b_*$ represent purely proba-

[4] We abbreviate terms $\theta.\underline{0}$ by omitting the final $\underline{0}$.

bilistic choices made according to parameter p (see the corresponding *GRTS*s in Fig. 3(a)). On the other hand, $a +^p b_*$ and $a_* +^p b_*$ represent purely nondeterministic choices, where the parameter p is not considered (see Fig. 3(b)). Finally, $(a +^{p'} b_*) +^p (b +^{p''} b_*)$ represents a mixed probabilistic and nondeterministic choice where parameters p' and p'' are not considered and therefore we have: one generative bundle made up of two transitions a and b, with probabilities p and $1-p$ respectively, and one reactive bundle of type b made up of two transitions b_*, with probabilities p and $1-p$, respectively (see Fig. 3(c)). ■

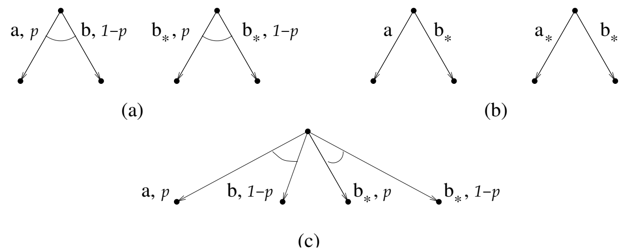


Fig. 3. Some Examples of *GRTS*s Derived from Alternative Composition

The parallel composition operator $P \parallel_S^p Q$ is based on a CSP like synchronization policy, where processes P and Q are required to synchronize over actions of type in the set S, and locally execute all the other actions. Moreover, as already explained in the introduction, we impose that a synchronization between two actions of type a may occur only if either they are both reactive actions a_* (and the result is a reactive action a_*), or one of them is a generative action a and the other one is a reactive action a_* (and the result is a generative action a). As far as generative actions are concerned, the generative actions of P (Q) executable by $P \parallel_S^p Q$ are such that either their type a is not in S, or a is in S and Q (P) can perform some reactive action a_*. In particular, as standard when restricting actions in the generative model [15], the probabilities of executing such actions are proportionally redistributed so that their overall probability sums up to 1. The choice among the generative actions of P and Q executable by $P \parallel_S^p Q$ is made according to probability p, by following the same probabilistic mechanism seen for alternative composition. In the case of synchronizing generative actions a of P (Q), their probability is further redistributed among the reactive actions a_* executable by Q (P), according to the probability they are chosen in Q (P). As far as reactive actions of a given type $a \notin S$ are concerned, $P \parallel_S^p Q$ may perform all the reactive actions a_* executable by P or Q and the choice among them is made according to probability p, by following the same probabilistic mechanism seen for alternative composition. As far as reactive actions of a given type $a \in S$ are concerned, if both P and Q may execute some reactive action a_*, the choice of the two actions a_* of P and Q forming the actions a_* executable by $P \parallel_S^p Q$ is made according to the probability they are *independently* chosen by P and Q.

Example 3.3 As a first example, consider the term $P_1 \equiv ((a +^q b) +^{q'} c) \parallel_{\{b\}}^p \underline{0}$.

The term P_1 may execute the generative actions a and c only, because action b must synchronize but does not have a reactive counterpart in the righthand process. Since term $((a +^q b) +^{q'} c)$ in isolation executes the generative action a with probability $q' \cdot q$, the generative action b with probability $q' \cdot (1-q)$ and the generative action c with probability $1-q'$, we redistribute the probabilities of a and c by dividing them by the sum of their probabilities $q' \cdot q + 1 - q'$. Therefore term P_1 executes action a with probability $(q' \cdot q)/(q' \cdot q + 1 - q')$ and action c with probability $(1-q')/(q' \cdot q + 1 - q')$ (see the *GRTS* of Fig. 4(a)).

As a second example, consider the term $P_2 \equiv ((a +^q b) +^{q'} c_*) \,||^p_{\{c\}}\, c$ which may execute the two non-synchronizing generative actions a and b of the lefthand process and the synchronizing generative action c of the righthand process (note that the execution of c is allowed by the reactive action c_* of the lefthand term). As in the case of alternative composition, since both processes may execute some generative actions, we perform a generative action of the lefthand process with probability p and a generative action of the righthand process with probability $1-p$. Since the generative actions executable by $(a +^q b) +^{q'} c_*$ are a with probability q and b with probability $1-q$ (according to the probabilistic mechanism of alternative composition q' is not considered), term P_2 executes action a with probability $p \cdot q$, action b with probability $p \cdot (1-q)$ and action c with probability $1-p$ (see the *GRTS* of Fig. 4(b)).

As a third example, consider the term $P_3 \equiv b \,||^p_{\{b\}} (b_*.c +^q b_*.d)$ where only the lefthand process may execute some generative action. Therefore, as in the case of alternative composition, we execute the unique generative action b with probability 1 and parameter p is not considered. Such an action synchronizes with one of the two reactive actions b_* of the righthand process chosen according to probability q. Therefore we execute the action b leading to $\underline{0} \,||^p_{\{b\}}\, c$ with probability q and the action b leading to $\underline{0} \,||^p_{\{b\}}\, d$ with probability $1-q$ (see the *GRTS* of Fig. 4(c)).

As a final example, consider the term $P_4 \equiv (a +^{q'} b_*) \,||^p_{\{b\}} (b_*.c +^q b_*.d)$. As far as the generative actions executable by P_4 are concerned, only the lefthand process may execute some generative action, hence, as in the case of alternative composition, we execute the unique generative action a with probability 1 and parameter p is not considered. As far as the reactive actions are concerned, P_4 may execute reactive actions of type b because b belongs to the sychronization set and both lefthand and righthand processes may execute some action b_*. The probability associated with an action b_* obtained by synchronizing an action b_* of the lefthand process with an action b_* of the righthand process is given by the probability that such actions are independently chosen by the two processes among their reactive actions of type b. Therefore, since in our example the lefhand process may perform only one reactive action b_*, P_4 executes the action b_* leading to $\underline{0} \,||^p_{\{b\}}\, c$ with probability q and the action b_* leading to $\underline{0} \,||^p_{\{b\}}\, d$ with probability $1-q$ (see the *GRTS* of Fig. 4(d)). ■

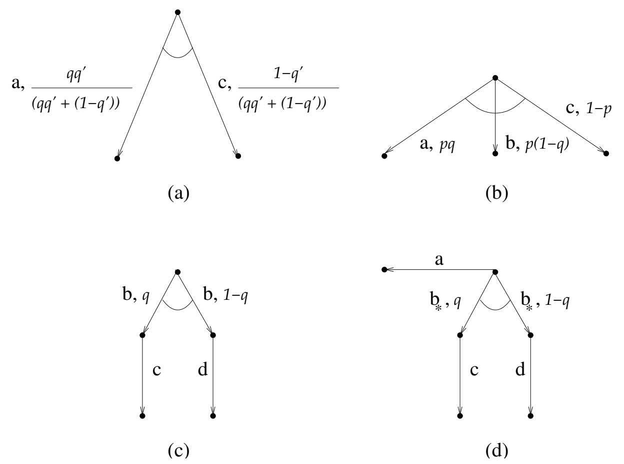


Fig. 4. Some Examples of *GRTS*s Derived from Parallel Composition

The relabeling operator $P[a \rightarrow b]^p$ turns actions of type a into actions of type b. The parameter p expresses the probability that reactive actions b_* obtained by relabeling actions a_* of P are executed with respect to the actions b_* previously performable by term P. As an example, consider the second *GRTS* of Fig. 3(b), corresponding to the process $P \triangleq a_* +^q b_*$, where the choice is purely nondeterministic. If we apply the relabeling operator $P[a \rightarrow b]^p$ we obtain the process represented by the second *GRTS* of Fig. 3(a), where the semantics of $P[a \rightarrow b]^p$ is a probabilistic choice between the action b_* obtained by relabeling the action a_* and the other action b_*, performed according to probabilities p and $1-p$, respectively. In this way the probabilistic information p provided in the operator $P[a \rightarrow b]^p$ guarantees that the relabeling operator does not introduce non-determinism between reactive actions of the same type. Parameter p is, instead, not used when relabeling generative actions because the choice between generative actions of type a and b in P is already probabilistic. Note that, since b may be the internal action τ, $P[a \rightarrow b]^p$ may behave also as an hiding operator.

Finally, constants A are used to specify recursive systems. In general, when defining an algebraic specification, we assume a set of constant defining equations of the form $A \triangleq P$ to be given.

In order to avoid ambiguities, we introduce the following operator precedence relation: prefix > relabeling > parallel composition > alternative composition. Moreover we use the following abbreviations to denote the hiding and relabeling of several action types. Let "P/L", where L is a finite sequence $\langle a_1^{p_1}, \ldots, a_n^{p_n} \rangle$ of actions $a_i \neq \tau$ with an associated probability p_i, stand for the expression $P[a_1 \rightarrow \tau]^{p_1} \ldots [a_n \rightarrow \tau]^{p_n}$, hiding the actions with types $a_1, \ldots, a_n$. In a similar way, let "$P[\varphi]$", where φ is a finite sequence $\langle (a_1, b_1)^{p_1}, \ldots, (a_n, b_n)^{p_n} \rangle$ of pairs of actions (a_i, b_i) such that $\tau \notin \{a_i, b_i\}$, with an associated probability p_i, stand for the expression $P[a_1 \rightarrow b_1]^{p_1} \ldots [a_n \rightarrow b_n]^{p_n}$, relabeling the actions of type $a_1, \ldots, a_n$ with the visible actions $b_1, \ldots, b_n$, respectively. For the sake of simplicity, we assume the parameter p to be equal to $\frac{1}{2}$ whenever it is omitted from any operator of our calculus.

Note that we employ a unique alternative composition operator, instead of a purely probabilistic one and a purely non-deterministic one for two main reasons. First, it permits to circumscribe all the non-determinism to bundles of different kinds, so that fully specified systems are performance closed. Second, a unique probabilistic choice operator is the natural counterpart of the unique parallel composition operator $P \parallel_S^p Q$ (having two parallel composition operators is not convenient because it makes it impossible to express mixed non-deterministic and probabilistic choices between the same two processes). An important consequence is that, as we will see in Sect. 5, we can simply characterize parallel composition by means of alternative composition.

Now, in order to make the reader more familiar with the probabilistic operators of the language, we present a small example that employs the main features of the mixed generative-reactive approach on which our calculus is based.

Example 3.4 Let us consider a system composed of a producer and a buffer. The overall system can be described as the interaction of two processes:

$$Producer \parallel_{\{produce\}}^p Buffer$$

The probabilistic parallel composition operator "$\_ \parallel_S^p \_$" is used to express a set of concurrent system components, their communication interface, and their probabilistic advancing speed in the way we will describe.

The communication interface $\{produce\}$ says that the two processes can interact by synchronously executing actions of type *produce*. Each other local action is asynchronously executed by the two processes. Probability p is the parameter of a probabilistic scheduler which, at each discrete time step (whose duration is the *time unit* on which the system specification is based), decides which of the two processes must be scheduled: in each system state the process between *Producer* and *Buffer* that will execute the next move is probabilistically chosen according to probability p and $1 - p$, respectively. As already explained in Sect. 1.2 this means that process *Producer* executes a mean of p actions per time unit, while process *Buffer* executes a mean of $1 - p$ actions per time unit.

Now let us detail each component. Process *Producer* repeatedly produces new items:

$$Producer \triangleq produce.Producer +^q \tau.Producer$$

The probabilistic alternative choice operator "$\_ +^q \_$" says that the producer can either produce a message (action *produce*) with probability q, or stay idle (action τ) with probability $1-q$, and afterwards behaving as the same process *Producer*. The actions *produce* and τ are *generative*, hence according to the *generative model* of probability [15], the process itself autonomously decides, on the basis of a probability distribution, which action will be executed and how to behave after such an event.

Process *Buffer*, instead, is ready to accept new incoming items or it stays idle:

$$Buffer \triangleq (produce_*.discard.Buffer +^r produce_*.store.Buffer) +^{r'} \tau.Buffer$$

The two actions $produce_*$ allow a synchronous interaction to be activated with an external process through a synchronization with an action of type *produce*. The potential interaction is guided by the external process (the producer) and, whenever this is the case, the buffer reacts by choosing either the first action $produce_*$ with probability r and then discarding the message (generative action *discard*), or the second action $produce_*$ with probability $1-r$ and then storing the message (generative action *store*). The two actions $produce_*$ are *reactive*, hence according to the *reactive model* of probability [15], the process reacts internally to an external action of type *produce* (performed by its environment) on the basis of a probability distribution associated with the reactive actions of type *produce* it can perform. In practice, the buffer reacts to stimuli presented by its environment in the form of synchronizing actions of type *produce*. This reflects a *master-slave* discipline, where a master (the producer) decides the action to execute and the slave (the buffer) reacts to its decision (reactive actions are incomplete actions which must synchronize with generative actions of another system component in order to be executed). In our example, process *Buffer* stays idle by executing internal actions τ while it is not getting items from the *Producer*. The choice between the reactive actions $produce_*$ and such an internal event is just nondeterministic (parameter r' is not considered), because the execution of a reactive action $produce_*$ by means of process *Buffer* is entirely guided by the process *Producer*. Whenever the process *Producer* performs its generative action *produce* the process *Buffer* will execute one of its $produce_*$ actions. For instance, in the initial state $Producer \parallel^p_{\{produce\}} Buffer$ of our example, the system executes a move of process *Producer* with probability p: it executes either the internal move τ with probability $p \cdot (1-q)$, or the move *produce* with probability $p \cdot q$ (with probability $p \cdot q \cdot r$ it executes a *produce* action synchronized with the first reactive action of *Buffer* and with probability $p \cdot q \cdot (1-r)$ a *produce* action synchronized with the second reactive action of *Buffer*). On the other hand, the system may schedule with probability $1-p$ the process *Buffer* by executing its internal action τ (which gets the *entire* probability $1-p$ associated to process *Buffer*). ■

3.2 Parallel Composition of Processes with Different Action Durations

As already explained in Sect. 1.2 our probabilistic parallel composition operator $P \parallel^p_S Q$ can be used to express the concurrent execution of processes P and Q specified with respect to different action durations. Before showing how this can be done in the general case we recall some basic concepts concerning the semantics of our parallel operator.

In the semantic model of $P \,\|^p_\emptyset\, Q$ the parallel execution of processes P and Q is represented as being originated by a single-processor machine executing both processes via a probabilistic scheduler. In each global state the scheduler probabilistically decides if P or Q is going to perform the next move according to probabilities p and $1-p$, respectively. In this way, as we explained in Sect. 1.2, P performs a mean of p actions per time unit (P is executed with action frequency p) and Q performs a mean of $1-p$ actions per time unit (Q is executed with action frequency $1-p$).

If we strictly follow this single-processor interpretation of the semantics of $P \,\|^p_S\, Q$, we assume that the specifications of processes P and Q are based on the same time unit u representing action duration, and consequently that actions of $P \,\|^p_S\, Q$ also take time u to be executed, i.e. u is also the time unit of $P \,\|^p_S\, Q$. Since P and Q must share a single resource (the processor), the effect of putting P in parallel with Q is that both P and Q get slowed down. In particular, when P (Q) is considered in isolation it executes one action per time unit u; when, instead, it is assumed to be in parallel with Q (P) by means of $P \,\|^p_\emptyset\, Q$, it executes p $(1-p)$ actions per time unit u.

On the other hand such an action interleaving based representation of $P \,\|^p_S\, Q$ can be interpreted as being an abstract description of the actual concurrent execution of two processes P and Q specified with respect to (possibly) different action durations, as in the case of the post office example of Sect. 1.2. In general, if f_P is the mean action frequency assumed in the specification of P (an action of P is assumed to take time $1/f_P$ on average to be executed) and f_Q is the mean action frequency assumed for the specification of Q, it is easy to derive a time unit u and a probability p such that, if we assume actions to take time u to be executed, $P \,\|^p_\emptyset\, Q$ represents the actual concurrent execution of processes P and Q with mean action frequencies f_P and f_Q, respectively. Since the mean action frequency of process P is f_P and the mean action frequency of process Q is f_Q, the mean action frequency of the parallel composition of P and Q must be $f = f_P + f_Q$. Therefore the time unit representing the duration of the actions of $P \,\|^p_\emptyset\, Q$ that we have to consider is $u = 1/f = 1/(f_P + f_Q)$, and the action frequency p of P with respect to the new time unit u is given by $f_P = p/u = p \cdot f$, hence $p = f_P/f = f_P/(f_P + f_Q)$. Similarly the action frequency $1-p$ of Q with respect to u turns out to be $1-p = f_Q/f = f_Q/(f_P + f_Q)$. It is worth noting that by adopting a suitable time unit in this way, the speed at which P and Q are executed when they are considered in isolation is not reduced when they are executed in parallel.

Example 3.5 Let us consider a system composed of a server managing a video-conference and a mail server. The first server must broadcast the flow of video images, hence requires a high speed channel to deliver data, while the second server mainly deals with textual messages, hence has smaller requirements in terms of channel bandwidth.

Let us first consider the scenario where the two servers share a single high speed channel which can transmit 500 data packets per second. Since both servers must send data over the same channel we assume a probabilistic scheduling policy for accessing the channel. In particular, since the video-conference server requires much larger bandwidth than the mail server we assume that every 2 milliseconds (the time it takes to send a packet through the channel) we schedule: a data packet coming from the video-conference server with probability 0.8, a data packet coming from the mail server with probability 0.2. In the system algebraic specification we represent the transmission over the channel of a packet coming from the video-conference server with actions a and the transmission over the channel of a packet coming from the mail server with actions b. The system is then specified by the term $A \,\|_{\emptyset}^{0.8}\, B$, where $A \triangleq a.A$ (which repeatedly executes actions a) describes the behavior of the video-conference server and $B \triangleq b.B$ describes the behavior of the mail server. The time unit that we adopt is 2 milliseconds. If we consider the specification A in isolation, the video-conference server has at its disposal the full bandwidth of the channel, hence it transmits a packet every time unit, i.e. 500 data packets per second. The same happens if we consider the specification B of the mail server in isolation. When instead we consider the system $A \,\|_{\emptyset}^{0.8}\, B$, since the two servers must share a unique channel, the effect that we obtain is that both servers get slowed down. In particular the video-conference server transmits 0.8 packets per time unit, i.e. 400 packets per second, while the mail server transmits 0.2 packets per time unit, i.e. 100 packets per second.

We can give a different interpretation to the same specification $A \,\|_{\emptyset}^{0.8}\, B$ by considering it as the description of the two servers in a different scenario where each of them has a dedicated channel: the video-conference server has an outgoing channel which can transmit a mean of 400 packets per second, while the mail server has an outgoing channel which can transmit a mean of 100 packets per second. According to this interpretation the actions a executed by process A (considered in isolation) represent the transmission of a packet of the video-conference server over its dedicated channel and take a mean of 2.5 milliseconds to be executed, while the actions b executed by process B (considered in isolation) represent the transmission of a packet of the mail server over its dedicated channel and take a mean of 10 milliseconds to be executed. In order to specify the actual concurrent execution of processes A and B we have to consider the global action frequency of the two processes, i.e. the total number $400 + 100$ of packets per second they transmit, and take as time unit the mean time to send a packet in the overall system, i.e. $1/(400 + 100)$ seconds or equally 2 milliseconds. Then we have to compute the action frequency p of A with respect to such a time unit. 400 packets per second correspond to $0.8 = 400/500$ packets per 2 milliseconds. Similarly the action frequency $1 - p$ of B with respect to the adopted time unit turns out to be $0.2 = 100/500$ packets per 2 milliseconds. Therefore the specification $A \,\|_{\emptyset}^{0.8}\, B$ with respect to the time unit 2 milliseconds represents the actual parallel execution of the two servers. In particular, differently from the case of the shared channel their advancing speed is not reduced when they are executed in parallel. ■

3.3 Semantics

The formal semantics of our calculus maps terms onto GRTSs, where each transition label is composed of an action and a probability.

We assume the following abbreviations that will make the definition of the semantic rules easier. We use $P \xrightarrow{\theta}$ to stand for $\exists\, p, P' : P \xrightarrow{\theta, p} P'$, meaning that P can execute action θ and $P \xrightarrow{G}$ to stand for $\exists\, a \in G : P \xrightarrow{a}$, $G \subseteq$ *AType*, meaning that P can execute a generative action of type belonging to set G.

The GRTSs deriving from a term $\mathcal{G}$ is defined by the operational rules in Tables 1 and 2, where in addition to the rules (r2_l) – (r5_l) and (g2_l) – (g7_l) referring to a local move of the lefthand process P, we consider also the symmetrical rules (r2_r) – (r5_r) and (g2_r) – (g7_r) taking into account the local moves of the righthand process Q, obtained by exchanging the roles of terms P and Q in the premises and, for rules (r2_r), (r4_r), (g2_r), (g4_r) and (g6_r), by replacing p with $1 - p$ in the label of the derived transitions. Similarly as in [18], we consider the operational rules as generating a multiset of transitions (consistently with the definition of a *GRTS*), where a transition has arity n if and only if it can be derived in n possible ways from the operational rules. Note that even if the operational rules in Tables 1 and 2 include negative premises, this does not cause inconsistencies because when applying such rules for deriving the moves of a term P, the negative premises always refer to the moves of a subterm of P (and not of P itself), hence the operational semantics is stratifiable [16].

Rule (r1) states that a term $a_*.P$ may execute a single reactive bundle of type a composed of a single transition leading to state P.

Rules (r2) state that, for any type a, the reactive bundle a executable by $P{+^p}Q$ is obtained by redistributing the probabilities of the transitions composing the reactive bundle a of P and Q, according to p and $1{-}p$, respectively. In the case the reactive bundle a of P or Q is empty, according to (r3), $P +^p Q$ simply

$$(\mathbf{r1})\quad a_*.P \xrightarrow{a_*,1} P$$

$$(\mathbf{r2_l})\quad \frac{P \xrightarrow{a_*,q} P' \quad Q \xrightarrow{a_*}}{P +^p Q \xrightarrow{a_*,p\cdot q} P'} \qquad (\mathbf{r3_l})\quad \frac{P \xrightarrow{a_*,q} P' \quad Q \not\xrightarrow{a_*}}{P +^p Q \xrightarrow{a_*,q} P'}$$

$$(\mathbf{r4_l})\quad \frac{P \xrightarrow{a_*,q} P' \quad Q \xrightarrow{a_*}}{P \parallel_S^p Q \xrightarrow{a_*,p\cdot q} P' \parallel_S^p Q}\ a \notin S \qquad (\mathbf{r5_l})\quad \frac{P \xrightarrow{a_*,q} P' \quad Q \not\xrightarrow{a_*}}{P \parallel_S^p Q \xrightarrow{a_*,q} P' \parallel_S^p Q}\ a \notin S$$

$$(\mathbf{r6})\quad \frac{P \xrightarrow{a_*,q} P' \ Q \xrightarrow{a_*,q'} Q'}{P \parallel_S^p Q \xrightarrow{a_*,q\cdot q'} P' \parallel_S^p Q'}\ a \in S$$

$$(\mathbf{r7})\quad \frac{P \xrightarrow{a_*,q} P' \quad P \xrightarrow{b_*}}{P[a \to b]^p \xrightarrow{b_*,p\cdot q} P'[a \to b]^p} \qquad (\mathbf{r8})\quad \frac{P \xrightarrow{a_*,q} P' \quad P \not\xrightarrow{b_*}}{P[a \to b]^p \xrightarrow{b_*,q} P'[a \to b]^p}$$

$$(\mathbf{r9})\quad \frac{P \xrightarrow{b_*,q} P' \quad P \xrightarrow{a_*}}{P[a \to b]^p \xrightarrow{b_*,(1-p)\cdot q} P'[a \to b]^p} \qquad (\mathbf{r10})\quad \frac{P \xrightarrow{b_*,q} P' \quad P \not\xrightarrow{a_*}}{P[a \to b]^p \xrightarrow{b_*,q} P'[a \to b]^p}$$

$$(\mathbf{r11})\quad \frac{P \xrightarrow{c_*,q} P'}{P[a \to b]^p \xrightarrow{c_*,q} P'[a \to b]^p}\quad c \notin \{a, b\}$$

$$(\mathbf{r12})\quad \frac{P \xrightarrow{a_*,q} P'}{A \xrightarrow{a_*,q} P'}\quad \text{if } A \triangleq P$$

Table 1
Semantic Rules for Reactive Transitions

$$(\mathbf{g1})\quad a.P \xrightarrow{a,1} P$$

$$(\mathbf{g2_l})\quad \frac{P \xrightarrow{a,q} P' \quad Q \xrightarrow{GAct}}{P +^p Q \xrightarrow{a,p\cdot q} P'} \qquad\qquad (\mathbf{g3_l})\quad \frac{P \xrightarrow{a,q} P' \quad Q \not\xrightarrow{GAct}}{P +^p Q \xrightarrow{a,q} P'}$$

$$(\mathbf{g4_l})\quad \frac{P \xrightarrow{a,q} P' \quad Q \xrightarrow{G_{S,P}}}{P \parallel_S^p Q \xrightarrow{a,p\cdot q/\nu_P(G_{S,Q})} P' \parallel_S^p Q}\ a \notin S$$

$$(\mathbf{g5_l})\quad \frac{P \xrightarrow{a,q} P' \quad Q \not\xrightarrow{G_{S,P}}}{P \parallel_S^p Q \xrightarrow{a,q/\nu_P(G_{S,Q})} P' \parallel_S^p Q}\ a \notin S$$

$$(\mathbf{g6_l})\quad \frac{P \xrightarrow{a,q} P' \quad Q \xrightarrow{a_*,q'} Q' \quad Q \xrightarrow{G_{S,P}}}{P \parallel_S^p Q \xrightarrow{a,p\cdot q'\cdot q/\nu_P(G_{S,Q})} P' \parallel_S^p Q'}\ a \in S$$

$$(\mathbf{g7_l})\quad \frac{P \xrightarrow{a,q} P' \quad Q \xrightarrow{a_*,q'} Q' \quad Q \not\xrightarrow{G_{S,P}}}{P \parallel_S^p Q \xrightarrow{a,q'\cdot q/\nu_P(G_{S,Q})} P' \parallel_S^p Q'}\ a \in S$$

$$(\mathbf{g8})\quad \frac{P \xrightarrow{a,q} P'}{P[a \to b]^p \xrightarrow{b,q} P'[a \to b]^p} \qquad (\mathbf{g9})\quad \frac{P \xrightarrow{c,q} P'}{P[a \to b]^p \xrightarrow{c,q} P'[a \to b]^p}\ a \neq c$$

$$(\mathbf{g10})\quad \frac{P \xrightarrow{a,q} P'}{A \xrightarrow{a,q} P'}\ \text{if } A \triangleq P$$

Table 2
Semantic Rules for Generative Transitions

inherits the reactive bundle a of the other process, without considering the parameter p.

Rules (r4) and (r5) state that, for any type $a \notin S$, the reactive bundle a executable by $P \,\|^p_S\, Q$ is obtained as previously explained for the alternative composition. Rule (r6) states that, for any type $a \in S$, if both the reactive bundles a of P and Q are non-empty, then the reactive bundle a of $P \,\|^p_S\, Q$ is non-empty, and is obtained by merging the two reactive bundles a of P and Q. In particular, the probability assigned to each transition a_* composing the reactive bundle a of $P \,\|^p_S\, Q$, is equal to the probability that the two synchronizing transitions a_* of P and Q are independently chosen by P and Q.

Rules (r7) and (r9) state that, the reactive bundle b executable by $P[a \rightarrow b]^p$ is obtained by redistributing the probabilities of the transitions composing the reactive bundles a and b of P, according to p and $1-p$, respectively. As for the alternative composition, if the reactive bundle a of P is empty, according to (r10), $P[a \rightarrow b]^p$ simply inherits the reactive bundle b of P, without considering the parameter p. Similarly, if the reactive bundle b of P is empty, according to (r8), the reactive bundle b of $P[a \rightarrow b]^p$ is obtained from the reactive bundle a of P by simply relabeling actions in transitions, without considering the parameter p. Rule (r11) states that, for any type $c \notin \{a, b\}$, $P[a \rightarrow b]^p$ simply inherits the reactive bundle c of P.

Rule (r12) states that the reactive bundles executable by A such that $A \triangleq P$ are the same reactive bundles executable by P.

Rules (g1), (g2), and (g3) are similar to the corresponding reactive rules. We only point out that the set $GAct$ is used in the premises of rules (g2) and (g3) because all generative actions are grouped in a single bundle.

According to rules (g4) $-$ (g7), the generative transitions of P (Q) executable by $P \,\|^p_S\, Q$ are such that either their type a is not in S, or a is in S and Q (P) can perform some reactive action a_*. In rules (g4) $-$ (g7) we assume the set $G_{S,P} \subseteq AType$, with $S \in AType - \{\tau\}$ and $P \in \mathcal{G}$, to be defined as follows:

$$G_{S,P} = \{a \in AType \mid a \notin S \vee (a \in S \wedge P \xrightarrow{a_*}\,)\}$$

In this way $G_{S,Q}$ ($G_{S,P}$) is the set of types of the generative transitions of P (Q) executable by $P \,\|^p_S\, Q$. Since we consider a restricted set of executable actions, as explained in Sect. 3.1 we redistribute the probabilities of the generative transitions of P (Q) executable by $P \,\|^p_S\, Q$ so that their overall probability sums up to 1 [15]. To this aim in semantic rules we employ the function $\nu_P(G) : \mathcal{P}(AType) \longrightarrow]0,1]$, with $P \in \mathcal{G}$, defined as follows:

$$\nu_P(G) = \sum \{\!|\, p \mid \exists P',\, a \in G : P \xrightarrow{a,p} P' \,|\!\}$$

which computes the sum of the probabilities of the generative transitions executable by P whose type belongs to the set G. In this way $\nu_P(G_{S,Q})$ ($\nu_Q(G_{S,P})$) computes the overall probability of the generative transitions of P (Q) executable by $P \parallel_S^p Q$ and can be used to normalize the probabilities of the generative transitions of P (Q). Finally, the generative bundle of $P \parallel_S^p Q$ is obtained by redistributing the probabilities of the normalized generative transitions of P and Q executable by $P \parallel_S^p Q$ according to p and $1 - p$, as for the alternative composition. The probability of the generative transitions a of P (Q), such that $a \in S$, must be further distributed among the reactive transitions belonging to the reactive bundle a of Q (P).

Rules (g8) and (g9) state that the generative bundle of $P[a \rightarrow b]^p$ is obtained from the generative bundle of P by simply relabeling actions in transitions, without considering the parameter p. Finally, rule (g10) is the exact counterpart of (r12).

Definition 3.6 The operational semantics of $P \in \mathcal{G}$ is the *PTS* $[\![P]\!]$ composed of the terms reachable from P according to the operational rules of Tables 1 and 2. We say that $P \in \mathcal{G}$ is *performance closed* if and only if $[\![P]\!]$ does not include reactive transitions. ■

Lemma 3.7 Let P and Q be two processes of $\mathcal{G}$. If $[\![P]\!]$ and $[\![Q]\!]$ are *GRTSs*, then $[\![\theta.P]\!]$, $[\![P[a \rightarrow b]^p]\!]$, $[\![P +^p Q]\!]$ and $[\![P \parallel_S^p Q]\!]$ are *GRTSs*. Moreover, recursive definitions preserve the property of being a *GRTS*.

Proof The result simply derives from the fact that, as explained above for each operational rule, the application of an operator preserves the bundle structure of *GRTS*s. ■

Theorem 3.8 If P is a process of $\mathcal{G}$, then $[\![P]\!]$ is a *GRTS*.

Proof The result derives by structural induction by using lemma 3.7. ■

3.4 Equivalence

We now equip the algebra with a probabilistic bisimulation equivalence (along the lines of [23]), which relates systems having the same functional and probabilistic behavior. We first introduce a function capturing the total probability with which a term reaches a given class of terms by executing a given action of *Act*.

Definition 3.9 We define function $\mathit{Prob} : (\mathcal{G} \times \mathit{Act} \times \mathcal{P}(\mathcal{G})) \longrightarrow [0,1]$ by

$$\mathit{Prob}(P, \theta, C) = \sum \{\!| \, p \mid P \xrightarrow{\theta,p} P' \wedge P' \in C \, |\!\}$$

■

Definition 3.10 An equivalence relation $\mathcal{B} \subseteq \mathcal{G} \times \mathcal{G}$ is a *strong probabilistic bisimulation* if and only if, whenever $(P, Q) \in \mathcal{B}$, then for all $\theta \in Act$ and equivalence classes $C \in \mathcal{G}/\mathcal{B}$

$$Prob(P, \theta, C) = Prob(Q, \theta, C)$$

The *strong probabilistic bisimulation equivalence*, denoted by $\sim_{PB}$, is the union of all the strong probabilistic bisimulations. ■

Theorem 3.11 $\sim_{PB}$ is a congruence w.r.t. all the algebraic operators and recursive definitions.

Proof The most relevant cases are those of parallel composition operator and recursive definitions. In the case of the parallel operator, it suffices to show that $\{(P_1 \parallel^p_S Q, P_2 \parallel^p_S Q) \mid P_1, P_2, Q \in \mathcal{G}. P_1 \sim_{PB} P_2\} \cup Id_{\mathcal{G}}$, where $Id_{\mathcal{G}}$ is the identity relation over $\mathcal{G}$, is a strong probabilistic bisimulation. In the case of recursive definitions, it suffices to apply the technique introduced in [11]. ■

4 Axiomatization

In this section we develop an equational characterization of $\sim_{PB}$ for the set $\mathcal{G}_{fin}$ of nonrecursive terms of $\mathcal{G}$, i.e. guarded and closed terms of our generative-reactive calculus not including constants A.

In order to produce a finite axiom system which is sound and complete over $\mathcal{G}_{fin}$ processes we adopt (as e.g. in [5]) the standard technique of expressing the parallel composition operator by means of the left merge $\lfloor^p_S$ and the synchronization merge $|^p_S$ operators. Moreover, in order to achieve completeness we need to introduce an auxiliary operator $\langle P \rangle^J_K$, so that when a term $\langle P \rangle^J_K$ is derived from P by the axiom system, K denotes the bundles executable by P and J denotes the type of the executable generative actions in the case the generative bundle belongs to K. Formally, we define bundle *kinds* as follows. Given a term P, "$\bullet$" is the kind of the generative bundle executable by P, while a is the kind of the reactive bundle of type a executable by P. We denote with $BKind = \{\bullet\} \cup AType$ the set of bundle kinds, ranged over by $\kappa, \kappa', \ldots$

We denote by $\mathcal{G}_{ext}$ the set of terms obtained by extending the syntax of $\mathcal{G}_{fin}$ terms with the auxiliary operators $\lfloor^p_S$ and $|^p_S$, and the new operator $\langle P \rangle^J_K$, where K is a finite set of bundle kinds, i.e. a subset of $BKind$, and $J \subseteq GAct$ is such that $J \neq \emptyset \iff \bullet \in K$.

Intuitively we have to introduce the new operator $\langle P \rangle^J_K$ in order to understand the role of parameter p in a choice $P +^p Q$, which can be probabilistic or nondeterministic depending on which bundles are executable by P and Q. In particular, we have that the choice $\langle P\rangle^J_K +^p \langle Q\rangle^{J'}_{K'}$ is purely probabilistic when there exists κ such that $K = K' = \{\kappa\}$, while it is purely non-deterministic whenever K and K' are disjoint.

In order to make it possible to produce an axiom system which, besides being complete, is also sound, we define an operational semantics for the new operator $\langle P\rangle^J_K$. As for the other operators of our calculus we will show that such semantics produces GRTSs and that $\langle P\rangle^J_K$ satisfies the congruence property.

The role of the operator $\langle P\rangle^J_K$ in our axiom system is just to denote that, when $\langle P\rangle^J_K$ is derived from P, the bundles executable by P are exactly the bundles stated by K and the types of the generative transitions of P are included in J. Therefore the operator $\langle P\rangle^J_K$ must have no effect on the behavior of processes P satisfying such a requirement. However, even if our axiom system cannot produce a term $\langle P\rangle^J_K$ from a process P not satisfying the requirement above, we have to define the semantics of $\langle P\rangle^J_K$ over arbitrary terms P. The idea here is to define $\langle P\rangle^J_K$ in such a way that, independently on which are the bundles executable by P, the bundles executable by $\langle P\rangle^J_K$ satisfy the requirement above. As we will see, defining $\langle P\rangle^J_K$ in this way will lead to the soundness of our axiom system.

The operational semantics of $\langle P\rangle^J_K$ is defined by the semantic rules defined in the first part of Table 3. In this table we employ the function $\kappa : Act \longrightarrow BKind$ which determines the kind of bundle a transition belongs to. Formally $\kappa(\theta)$ is defined as follows:

$$\forall a \in GAct. \quad \kappa(a) = \bullet$$
$$\forall a_* \in RAct.\ \kappa(a_*) = a$$

Rule (e1) eliminates all the transitions of P not executable according to K and J. Rule (e2) creates a reactive bundle of type a in the case P cannot execute such a bundle and $a \in K$. Rule (e3) creates a complete generative bundle in the case $\bullet \in K$ and the generative transitions of P whose type is in J do not form a complete generative bundle. The action a used in rule (e3) is any *fixed* arbitrarily chosen generative action $a \in J$. Such an action can be considered as a fixed parameter of the operator $\langle P\rangle^J_K$.

In the second and third part of Table 3 we present the obvious operational rules for $\lfloor^p_S$ and $|^p_S$ that derive from those we presented for the parallel operator.

Lemma 4.1 Let P and Q be two processes of $\mathcal{G}_{ext}$. If $[\![P]\!]$ and $[\![Q]\!]$ are *GRTSs* then $[\![\langle P\rangle^J_K]\!]$, $[\![P \lfloor^p_S Q]\!]$, and $[\![P\ |^p_S\ Q]\!]$ are *GRTSs*.

Proof The result simply derives from the fact that, as it can be easily verified, the application of the operators $[\![\langle P\rangle^J_K]\!]$, $[\![P \lfloor^p_S Q]\!]$, and $[\![P\ |^p_S\ Q]\!]$ preserve the bundle structure of *GRTS*s. In particular for the left merge and synchroniza-

$$(\mathbf{e1})\ \frac{P \xrightarrow{\theta,q} P'}{\langle P\rangle^J_K \xrightarrow{\theta,q} P'}\ \text{if } \begin{array}{l}\kappa(\theta) \in K \wedge \\ (\kappa(\theta) = \bullet \ \Rightarrow\ \theta \in J)\end{array}$$

$$(\mathbf{e2})\ \frac{P \stackrel{a_*}{\not\longrightarrow}}{\langle P\rangle^J_K \xrightarrow{a_*,1} \underline{0}}\ \text{if } a \in K \qquad (\mathbf{e3})\ \frac{\nu_P(J) < 1}{\langle P\rangle^J_K \xrightarrow{a,1-\nu_P(J)} \underline{0}}\ \text{if } \begin{array}{l}\bullet \in K \wedge \\ a \in J\end{array}$$

$$(\mathbf{lm1})\ \frac{P \xrightarrow{a_*,q} P' \quad Q \xrightarrow{a_*}}{P \lfloor^p_S Q \xrightarrow{a_*,p\cdot q} P' \|^p_S Q}\ a \notin S \qquad (\mathbf{lm2})\ \frac{P \xrightarrow{a_*,q} P' \quad Q \stackrel{a_*}{\not\longrightarrow}}{P \lfloor^p_S Q \xrightarrow{a_*,q} P' \|^p_S Q}\ a \notin S$$

$$(\mathbf{lm3})\ \frac{P \xrightarrow{a,q} P' \quad Q \xrightarrow{G_{S,P}}}{P \lfloor^p_S Q \xrightarrow{a,p\cdot q/\nu_P(G_{S,Q})} P' \|^p_S Q}\ a \notin S$$

$$(\mathbf{lm4})\ \frac{P \xrightarrow{a,q} P' \quad Q \stackrel{G_{S,P}}{\not\longrightarrow}}{P \lfloor^p_S Q \xrightarrow{a,q/\nu_P(G_{S,Q})} P' \|^p_S Q}\ a \notin S$$

$$(\mathbf{lm5})\ \frac{P \xrightarrow{a,q} P' \quad Q \xrightarrow{a_*,q'} Q' \quad Q \xrightarrow{G_{S,P}}}{P \lfloor^p_S Q \xrightarrow{a,p\cdot q'\cdot q/\nu_P(G_{S,Q})} P' \|^p_S Q'}\ a \in S$$

$$(\mathbf{lm6})\ \frac{P \xrightarrow{a,q} P' \quad Q \xrightarrow{a_*,q'} Q' \quad Q \stackrel{G_{S,P}}{\not\longrightarrow}}{P \lfloor^p_S Q \xrightarrow{a,q'\cdot q/\nu_P(G_{S,Q})} P' \|^p_S Q'}\ a \in S$$

$$(\mathbf{sm1})\ \frac{P \xrightarrow{a_*,q} P' \ Q \xrightarrow{a_*,q'} Q'}{P |^p_S Q \xrightarrow{a_*,q\cdot q'} P' \|^p_S Q'}\ a \in S$$

Table 3
Semantic Rules for $\langle P\rangle^J_K$, $P \lfloor^p_S Q$, and $P |^p_S Q$.

tion merge operators this is a trivial consequence of the fact that the bundle structure is preserved by the parallel operator. ■

Theorem 4.2 If P is a process of $\mathcal{G}_{ext}$, then $[\![P]\!]$ is a *GRTS*.

Proof The result derives by structural induction by using lemma 3.7 and lemma 4.1. ∎

Theorem 4.3 $\sim_{PB}$ is a congruence w.r.t. the auxiliary operators $\langle P\rangle^J_K$, $P \lfloor^p_S Q$ and $P \,|^p_S\, Q$.

Proof As far as the operator $\langle P\rangle^J_K$ is concerned the result simply derives from the fact that, if $P_1 \sim_{PB} P_2$ then for any set of action types J we have that $\nu_{P_1}(J) = \nu_{P_2}(J)$, hence $\langle P_1\rangle^J_K \sim_{PB} \langle P_2\rangle^J_K$. Moreover, since the equivalence $\sim_{PB}$ does not abstract from τ transitions the congruence w.r.t. the auxiliary operators $P \lfloor^p_S Q$ and $P \,|^p_S\, Q$ is just a trivial consequence of the congruence with respect to the parallel operator (see [1]). ∎

The equational characterization of $\sim_{PB}$ is composed of the set $\mathcal{A}$ of axioms shown in Table 4. The axioms $\mathcal{A}_1 - \mathcal{A}_5$ express the basic properties of the choice operator. The axioms $\mathcal{BK}_1 - \mathcal{BK}_3$ generate the brackets $\langle P\rangle^J_K$ denoting the bundles executable by P. The axioms $\mathcal{R}_1 - \mathcal{R}_7$ and $\mathcal{P}$, $\mathcal{LM}_1 - \mathcal{LM}_9$, $\mathcal{SM}_1 - \mathcal{SM}_5$ refer to the relabeling and parallel composition operators, respectively.

Theorem 4.4 The axiom system $\mathcal{A}$ is sound for $\sim_{PB}$ over processes of $\mathcal{G}_{ext}$.

Proof Just a trivial verification of the consistency of the axioms with respect to the operational semantics of the operators. In particular note that the internal angular parentheses in the righthand term of axiom $\mathcal{BK}_3$ are necessary to achieve soundness. If we had considered external angular parentheses only, then the axiom system would have been complete but not sound, as can be seen by taking, e.g. $P \equiv \underline{0}$. Moreover, note that axiom $\mathcal{LM}_6$ is sound because we apply it when the lefthand process is guarded by a synchronizing generative action a waiting for the reactive counterpart a_* in the righthand process. ∎

The axiom $\mathcal{A}_2$ expresses idempotency and $\mathcal{A}_3$ a form of commutativity which complements the parameter p (as in [5]).

The axiom $\mathcal{A}_4$ expresses associativity for pure probabilistic choices, i.e. choices among terms enabling a single common bundle. The axiom defines how probabilistic parameters must be adjusted so that the probability of executing a given member of the sum is preserved (as in [5]).

Thanks to the properties of commutativity and associativity of pure probabilistic choices we can just use the notation [5]

$$\sum_{i\in\{1,\ldots,n\}} [p_i]\langle P_i\rangle^{J_i}_{\{\kappa\}}$$

[5] We assume $\sum_{i\in I} [p_i]\langle P_i\rangle^{J_i}_{\{\kappa\}} \equiv \underline{0}$ whenever $I = \emptyset$.

$(\mathcal{A}_1)$	$P +^{p} \underline{0} = P$	
$(\mathcal{A}_2)$	$P +^{p} P = P$	
$(\mathcal{A}_3)$	$P +^{p} Q = Q +^{1-p} P$	
$(\mathcal{A}_4)$	$(\langle P\rangle^{J}_{\{\kappa\}} +^{p} \langle Q\rangle^{J'}_{\{\kappa\}}) +^{q} \langle R\rangle^{J''}_{\{\kappa\}} = \langle P\rangle^{J}_{\{\kappa\}} +^{pq} (\langle Q\rangle^{J'}_{\{\kappa\}} +^{(1-p)\cdot q/(1-p\cdot q)} \langle R\rangle^{J''}_{\{\kappa\}})$	
$(\mathcal{A}_5)$	$(\langle P\rangle^{J}_{K} +^{p} \langle Q\rangle^{J'}_{K'}) +^{q} R = \langle P\rangle^{J}_{K} +^{q} (\langle Q\rangle^{J'}_{K'} +^{q} R)$	$K \cap K' = \emptyset$
$(\mathcal{BK}_1)$	$a.P = \langle a.P\rangle^{\{a\}}_{\{\bullet\}}$	
$(\mathcal{BK}_2)$	$a_*.P = \langle a_*.P\rangle^{\emptyset}_{\{a\}}$	
$(\mathcal{BK}_3)$	$\langle P\rangle^{J}_{K} +^{p} \langle Q\rangle^{J'}_{K'} = \langle \langle P\rangle^{J}_{K} +^{p} \langle Q\rangle^{J'}_{K'}\rangle^{J\cup J'}_{K\cup K'}$	
$(\mathcal{R}_1)$	$\underline{0}[a \to b]^{p} = \underline{0}$	
$(\mathcal{R}_2)$	$\theta.P[a \to b]^{p} = \theta.(P[a \to b]^{p})$	$\theta \notin \{a,\, a_*\}$
$(\mathcal{R}_3)$	$a.P[a \to b]^{p} = b.(P[a \to b]^{p})$	
$(\mathcal{R}_4)$	$a_*.P[a \to b]^{p} = b_*.(P[a \to b]^{p})$	
$(\mathcal{R}_5)$	$(\langle P\rangle^{J}_{K} +^{ND} \langle Q\rangle^{J'}_{K'})[a \to b]^{p} = \langle P\rangle^{J}_{K}[a \to b]^{p} +^{ND} \langle Q\rangle^{J'}_{K'}[a \to b]^{p}$	$a, b \notin K \wedge K \cap K' = \emptyset$
$(\mathcal{R}_6)$	$(\langle P\rangle^{\emptyset}_{\{a\}} +^{ND} \langle Q\rangle^{\emptyset}_{\{b\}})[a \to b]^{p} = \langle P\rangle^{\emptyset}_{\{a\}}[a \to b]^{p} +^{p} \langle Q\rangle^{\emptyset}_{\{b\}}[a \to b]^{p}$	
$(\mathcal{R}_7)$	$(\langle P\rangle^{J}_{\{\kappa\}} +^{q} \langle Q\rangle^{J'}_{\{\kappa\}})[a \to b]^{p} = \langle P\rangle^{J}_{\{\kappa\}}[a \to b]^{p} +^{q} \langle Q\rangle^{J'}_{\{\kappa\}}[a \to b]^{p}$	
$(\mathcal{P})$	$P \parallel^{p}_{S} Q = (P \lfloor^{p}_{S} Q +^{p} Q \lfloor^{1-p}_{S} P) +^{ND} P \mid^{p}_{S} Q$	
$(\mathcal{LM}_1)$	$\underline{0} \lfloor^{p}_{S} Q = \underline{0}$	
$(\mathcal{LM}_2)$	$a_*.P \lfloor^{p}_{S} Q = \underline{0}$	$a \in S$
$(\mathcal{LM}_3)$	$a.P \lfloor^{p}_{S} \underline{0} = \underline{0}$	$a \in S$
$(\mathcal{LM}_4)$	$a.P \lfloor^{p}_{S} \theta.Q = \underline{0}$	$a \in S \wedge \theta \neq a_*$
$(\mathcal{LM}_5)$	$a.P \lfloor^{p}_{S} a_*.Q = a.(P \parallel^{p}_{S} Q)$	$a \in S$
$(\mathcal{LM}_6)$	$a.P \lfloor^{p}_{S}(Q +^{q} R) = a.P \lfloor^{p}_{S} Q +^{q} a.P \lfloor^{p}_{S} R$	$a \in S$
$(\mathcal{LM}_7)$	$\theta.P \lfloor^{p}_{S} Q = \theta.(P \parallel^{p}_{S} Q)$	$\theta \notin S$
$(\mathcal{LM}_8)$	$(P +^{q} \langle Q\rangle^{J}_{\{\bullet\}}) \lfloor^{p}_{S} \langle R\rangle^{J'}_{K} = P \lfloor^{p}_{S} \langle R\rangle^{J'}_{K}$	$J \subseteq S \wedge J \cap K = \emptyset$
$(\mathcal{LM}_9)$	$(\langle P\rangle^{J}_{K} +^{q} \langle Q\rangle^{J'}_{K'}) \lfloor^{p}_{S} \langle R\rangle^{J''}_{K''} = \langle P\rangle^{J}_{K} \lfloor^{p}_{S} \langle R\rangle^{J''}_{K''} +^{q} \langle Q\rangle^{J'}_{K'} \lfloor^{p}_{S} \langle R\rangle^{J''}_{K''}$	$(J \cup J') \cap S \subseteq K''$
$(\mathcal{SM}_1)$	$P \mid^{p}_{S} Q = Q \mid^{1-p}_{S} P$	
$(\mathcal{SM}_2)$	$\underline{0} \mid^{p}_{S} Q = \underline{0}$	
$(\mathcal{SM}_3)$	$\theta.P \mid^{p}_{S} \theta'.Q = \underline{0}$	$\theta \notin \{a_* \mid a \in S\} \vee \theta \neq \theta'$
$(\mathcal{SM}_4)$	$a_*.P \mid^{p}_{S} a_*.Q = a_*.(P \parallel^{p}_{S} Q)$	$a \in S$
$(\mathcal{SM}_5)$	$(P +^{q} Q) \mid^{p}_{S} R = P \mid^{p}_{S} R +^{q} Q \mid^{p}_{S} R$	

Table 4
Axiomatization

as a shorthand notation for a probabilistic choice among $\langle P_1\rangle^{J_1}_{\{\kappa\}} \ldots \langle P_n\rangle^{J_n}_{\{\kappa\}}$, where p_i is the probability of executing $\langle P_i\rangle^{J_i}_{\{\kappa\}}$.

The axiom $\mathcal{A}_5$ expresses associativity for choices where two of the terms involved enable disjoint sets of bundle kinds. Two particular cases of $\mathcal{A}_5$ lead to two important properties of the choice operator.

By taking $R \equiv \underline{0}$ in $\mathcal{A}_5$ and applying axiom $\mathcal{A}_1$ we obtain:

$$\langle P\rangle^{J}_{K} +^{p} \langle Q\rangle^{J'}_{K'} \;=\; \langle P\rangle^{J}_{K} +^{q} \langle Q\rangle^{J'}_{K'} \text{ if } K \cap K' = \emptyset$$

This equation expresses the fact that whenever we have a choice between two terms enabling different bundles, i.e. a pure non-deterministic choice, the probabilistic parameter of the choice is not important. Thanks to this property, supposed $K \cap K' = \emptyset$, we can just use the notation $\langle P\rangle^{J}_{K} +^{ND} \langle Q\rangle^{J'}_{K'}$, to stand for any term $\langle P\rangle^{J}_{K} +^{p} \langle Q\rangle^{J'}_{K'}$ obtained for a particular choice of p. [6]

By taking $R \equiv \langle R'\rangle^{J''}_{K''}$, such that $K'' \cap K = \emptyset$ and $K'' \cap K' = \emptyset$, in $\mathcal{A}_5$ and by taking into account the previous equation we obtain:

$$(\langle P\rangle^{J}_{K} +^{ND} \langle Q\rangle^{J'}_{K'}) +^{ND} \langle R\rangle^{J''}_{K''} \;=\; \langle P\rangle^{J}_{K} +^{ND} (\langle Q\rangle^{J'}_{K'} +^{ND} \langle R\rangle^{J''}_{K''})$$
$$\text{if } K, K', K'' \text{ are pairwise disjoint}$$

This equation expresses associativity for pure non-deterministic choices, i.e. choices among terms enabling disjoint sets of bundles.

Thanks to the properties of commutativity and associativity of pure non-deterministic choices we can just use the notation

$$\sum_{i\in\{1,\ldots,n\}} [ND] \langle P_i\rangle^{J_i}_{K_i}$$

as a shorthand notation for a non-deterministic choice among $\langle P_1\rangle^{J_1}_{K_1} \ldots \langle P_n\rangle^{J_n}_{K_n}$, where K_i with $i \in \{1, \ldots, n\}$ are pairwise disjoint.

Besides the two properties above that the axiom $\mathcal{A}_5$ expresses, this axiom is important in that it allows us to turn any sequential term (a term made up

[6] This notation is employed just to improve the readability of axioms: whenever $P +^{ND} Q$ occurs in some equation, it could just be replaced by any choice of a particular parameter p.

of $\underline{0}$, prefix and choice) into *sum normal form*.

Definition 4.5 $P \in \mathcal{G}_{ext}$ is in *sum normal form* (*snf*) if and only if

$$P \equiv \sum_{i \in I}[p_i]a_i.P_i \; +^{ND} \sum_{a \in AType}[ND] \sum_{i \in I^a}[p_i]a_*.P_i$$

where every P_i is itself in *snf*. [7] We assume the sets of indexes I and I^a for each $a \in AType$ to be pairwise disjoint, i.e. $\forall a \in AType.\, I_a \cap I = \emptyset$ and $\forall a, b \in AType, a \neq b.\, I_a \cap I_b = \emptyset$. ■

The axioms $\mathcal{R}_1 - \mathcal{R}_7$ characterize the behavior of the relabeling operator and allow us to turn a term $P[a \rightarrow b]^p$, where P is in snf, into snf. More precisely, we resort to the axioms $\mathcal{R}_5 - \mathcal{R}_7$ for distributing the relabeling operator among the alternative behaviors of P, and then we use axioms $\mathcal{R}_1 - \mathcal{R}_4$ for applying the relabeling operator to each $\underline{0}$ or prefix term. In particular, axiom $\mathcal{R}_6$ specifies how the nondeterministic choice among reactive actions a_* and b_* in P becomes a probabilistic choice (with parameter p) among reactive actions b_* in $P[a \rightarrow b]^p$. Note that, thanks to the fact that relabeling employs the same choice mechanism between reactive actions a_* and b_* as that of alternative composition, it is possible to express relabeling with parameter p in terms of choice with parameter p.

The axiom $\mathcal{P}$ characterizes the parallel composition operator in terms of the left merge $\lfloor^p_S$ and the synchronization merge $|^p_S$ operators (see e.g. [5]) and allows us to turn a term $P \,\|^p_S\, Q$, where P and Q are in snf, into snf. In particular, we resort to the left merge in order to express the local reactive and generative moves of the lefthand process, including the generative moves which synchronize with a reactive response of the righthand process (note that such moves are those depending on the parameter p of the parallel composition, see axioms $\mathcal{LM}_1 - \mathcal{LM}_9$), and we resort to the synchronization merge in order to express the synchronizing reactive moves (see axioms $\mathcal{SM}_1 - \mathcal{SM}_5$). Note that, thanks to fact that the parallel composition operator employs the same choice mechanism between the generative and reactive actions locally executed by P and Q as for the alternative composition, it is possible to express parallel composition with parameter p in terms of choice with parameter p. As far as generative actions are concerned, the operator $\langle P \rangle^J_K$ used in axioms $\mathcal{LM}_8$ and $\mathcal{LM}_9$ allows us to deal with action restriction (which in our calculus arises from parallel composition), without resorting to axioms with implications as in [5]. Finally, it is worth noting that when considering only reactive actions we obtain a smooth probabilistic extension of the classical axiomatization for the parallel composition by means of the left and synchronization merge.

[7] The use of $\sum$ and ND is correct since we have $a_i.P_i = \langle a_i.P_i \rangle^J_{\{\bullet\}}$ (axiom $\mathcal{A}_4$) and $a_*.P_i = \langle a_*.P_i \rangle^{\emptyset}_{\{a\}}$ (axiom $\mathcal{A}_5$).

Lemma 4.6 For any $P \in \mathcal{G}_{fin}$ there exists $P' \in \mathcal{G}_{fin}$ in *snf* such that $\mathcal{A} \vdash P = P'$.

Proof The proof proceeds by structural induction over terms $P \in \mathcal{G}_{fin}$. The base step of the induction is just when P is a prefix operator or $\underline{0}$, hence it is already in snf.

For the inductive step we have the following cases:

- Supposed that terms P_1 and P_2 are in snf we show that $P \equiv P_1 +^p P_2$ can be turned into snf by using axiom $\mathcal{A}_5$.
  Since P_1 is in snf we can, by using axioms $\mathcal{BK}_1 - \mathcal{BK}_3$, write it as: $\langle P_1' \rangle_K^J +^{ND} \langle P_1'' \rangle_\kappa^{J'}$ with $\kappa \notin K$.
  From $P = (\langle P_1' \rangle_K^J +^{ND} \langle P_1'' \rangle_\kappa^{J'}) +^p P_2$ we derive, by using $\mathcal{A}_5$:
  $$P = \langle P_1' \rangle_K^J +^p (\langle P_1'' \rangle_\kappa^{J'} +^p P_2).$$
  Now, supposed that P_2 enables the bundle κ (the opposite case is trivial), since P_2 is in snf we can, by using axioms $\mathcal{BK}_1 - \mathcal{BK}_3$, write it as: $\langle P_2' \rangle_{K'}^{J''} +^{ND} \langle P_2'' \rangle_\kappa^{J'''}$ with $\kappa \notin K'$.
  Therefore we have: $P = \langle P_1' \rangle_K^J +^p (\langle P_1'' \rangle_\kappa^{J'} +^p (\langle P_2' \rangle_{K'}^{J''} +^{ND} \langle P_2'' \rangle_\kappa^{J'''}))$
  By applying axiom $\mathcal{A}_3$ we have:
  $$P = \langle P_1' \rangle_K^J +^p (((\langle P_2' \rangle_{K'}^{J''} +^{ND} \langle P_2'' \rangle_\kappa^{J'''}) +^{1-p} \langle P_1'' \rangle_\kappa^{J'})$$
  By applying the axiom $\mathcal{A}_5$ we have:
  $$P = \langle P_1' \rangle_K^J +^p (\langle P_2' \rangle_{K'}^{J''} +^{ND} (\langle P_2'' \rangle_\kappa^{J'''} +^{1-p} \langle P_1'' \rangle_\kappa^{J'}))$$
  By applying two times the axiom $\mathcal{A}_3$ we have:
  $$P = (((\langle P_2'' \rangle_\kappa^{J'''} +^{1-p} \langle P_1'' \rangle_\kappa^{J'}) +^{ND} \langle P_2' \rangle_{K'}^{J''}) +^{1-p} \langle P_1' \rangle_K^J$$
  By applying the axiom $\mathcal{A}_5$ we have:
  $$P = ((\langle P_2'' \rangle_\kappa^{J'''} +^{1-p} \langle P_1'' \rangle_\kappa^{J'}) +^{ND} ((\langle P_2' \rangle_{K'}^{J''} +^{1-p} \langle P_1' \rangle_K^J)$$
  By applying two times the axiom $\mathcal{A}_3$ we have:
  $$P = ((\langle P_1'' \rangle_\kappa^{J'} +^p \langle P_2'' \rangle_\kappa^{J'''}) +^{ND} ((\langle P_1' \rangle_K^J +^p \langle P_2' \rangle_{K'}^{J''})$$
  In this expression we have isolated the bundle κ from the other bundles. Now the same procedure is applied to $((\langle P_1' \rangle_K^J +^p \langle P_2' \rangle_{K'}^{J''})$ which no longer includes the bundle κ. When all the bundles have been isolated in this way we have obtained an expression in snf for term P.
- Supposed that term P_1 is in snf it is easy to see that $P \equiv P_1[a \to b]^p$ can be turned into snf by using axioms $\mathcal{R}_1 - \mathcal{R}_7$. We first employ the axioms $\mathcal{R}_5 - \mathcal{R}_7$ in order to properly apply the relabeling operator to each alternative behavior of P. We apply the relabeling operator to each component of P representing a single bundle by using axiom $\mathcal{R}_5$. It is worth noting that axiom $\mathcal{R}_5$ cannot be used to distribute the relabeling operator among the two reactive bundles a and b. In order to do this, we apply axiom $\mathcal{R}_6$, as

it recomputes the probability distribution of the reactive bundles a and b (by using the parameter p) according to the fact that the reactive actions a_* will be turned into the reactive actions b_*. Finally, for each component representing a single bundle, we use axiom $\mathcal{R}_7$ in order to distribute the relabeling operator among the possible behaviors of the bundle. Then we apply the relabeling operator to each $\underline{0}$ or prefix term by means of axioms $\mathcal{R}_1 - \mathcal{R}_4$.

- Supposed that terms P_1 and P_2 are in snf it is easy to see that $P \equiv P_1 \|_S^p P_2$ can be turned into snf by using axioms $\mathcal{P}$, $\mathcal{LM}_1 - \mathcal{LM}_9$ and $\mathcal{SM}_1 - \mathcal{SM}_5$.

  In particular, we resort to the synchronization merge in order to manage the actions of the synchronizing reactive bundles belonging to the set S, and the left merge for the local reactive and generative actions, including the generative actions which synchronize with a reactive action. In the case of the synchronization merge, by applying the axioms $\mathcal{SM}_1$ and $\mathcal{SM}_5$ we obtain a term in sum form on which we can apply the axioms $\mathcal{SM}_2$, $\mathcal{SM}_3$ and $\mathcal{SM}_4$. Note that the axiom $\mathcal{SM}_1$ is the exact counterpart of the axiom $\mathcal{A}_3$, and all the axioms we use are exactly as in the classical axiomatization except for the presence of the parameter p in the parallel operator. In the case of the left merge, by applying the axioms $\mathcal{LM}_8$ and $\mathcal{LM}_9$ we obtain a term in sum form on which we can apply the axioms $\mathcal{LM}_1 - \mathcal{LM}_7$. In particular, the axiom $\mathcal{LM}_8$ allows us to eliminate the transitions representing the subset of the generative bundle which is blocked because of the synchronization constraint expressed by the set S of the parallel operator. After this, we can apply the axiom $\mathcal{LM}_9$ in order to distribute the left merge operator among the components in sum form of the left side of the parallel composition. Axioms $\mathcal{LM}_1 - \mathcal{LM}_4$ allows us to eliminate the blocked components according to the synchronization set S of the parallel operator, and the axioms $\mathcal{LM}_5$ and $\mathcal{LM}_7$ determine all the synchronizing generative moves and the local moves of the lefthand process, respectively. Finally, axiom $\mathcal{LM}_6$ allows us to distribute the left merge operator among the components of the righthand process, in order to apply the axioms $\mathcal{LM}_4$ and $\mathcal{LM}_5$. ■

Example 4.7 Let us consider the term

$$((((a_*.\underline{0} +^q a_*.\underline{0}) +^r a_*.\underline{0}) +^{ND} (b_*.\underline{0} +^s b_*.\underline{0})) +^{ND} a.\underline{0})[a \to b]^p$$

which is the relabeling of a process in snf. We first apply axiom $\mathcal{R}_5$ that allows us to isolate the components representing the reactive bundles a and b, thus obtaining:

$$(((a_*.\underline{0} +^q a_*.\underline{0}) +^r a_*.\underline{0}) +^{ND} (b_*.\underline{0} +^s b_*\underline{0}))[a \to b]^p +^{ND} a.\underline{0}[a \to b]^p$$

Then, we distribute the relabeling operator between these two components by applying axiom $\mathcal{R}_6$, hence obtaining:

$$(((a_*.\underline{0} +^q a_*.\underline{0}) +^r a_*.\underline{0})[a \to b]^p +^p (b_*.\underline{0} +^s b_*.\underline{0})[a \to b]^p) +^{ND} a.\underline{0}[a \to b]^p$$

In this way the relabeling of the two reactive bundles of type a and b is expressed in terms of a probabilistic choice among two terms executing reactive actions of type b. Then, by repeatedly applying axiom $\mathcal{R}_7$ we have:

$$(((a_*.\underline{0}[a \to b]^p +^q a_*.\underline{0}[a \to b]^p) +^r a_*.\underline{0}[a \to b]^p)+^p$$

$$(b_*.\underline{0}[a \to b]^p +^s b_*.\underline{0}[a \to b]^p)) +^{ND} a.\underline{0}[a \to b]^p$$

Finally, by applying axioms $\mathcal{R}_1 - \mathcal{R}_4$, we change the actions of type a into actions of type b as follows:

$$(((b_*.\underline{0} +^q b_*.\underline{0}) +^r b_*.\underline{0}) +^p (b_*.\underline{0} +^s b_*.\underline{0})) +^{ND} b.\underline{0}$$

Such a term correctly represents the behavior of the relabeled term, where the choice among the actions b_* obtained by relabeling actions a_* and the preexisting actions b_* is guided by parameter p. ■

Theorem 4.8 The axiom system $\mathcal{A}$ is complete for $\sim_{PB}$ over processes of $\mathcal{G}_{fin}$.

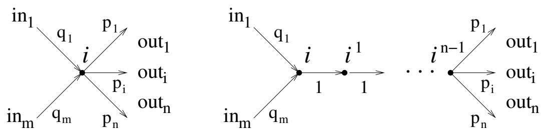
Proof The result is a trivial consequence of lemma 4.6. It is just sufficient to show that if two terms in snf are equivalent then they can be proved to be equal. Similarly as done in [8] this is proved in a rather standard way by inducing on the structure of the two terms in snf, in such a way that equivalent subterms are turned into equal subterms. In this procedure an important role is played by the idempotency axiom $\mathcal{A}_2$. ■

5 Processes with Exact Advancing Speeds

In this section we show that, when evaluating steady state based performance measures, we are able to deal with processes proceeding with different advancing speeds which are not probabilistic. In particular, while during an initial transient evolution considering the action frequency of processes as being exact instead of probabilistic may lead to different results when evaluating performance, we will show that in the case of non-blocking processes this does not happen when the system reaches a limiting steady behavior.

5.1 Exact Time Unit Scaling via Action Splitting

Let us assume that process P is a fully specified system such that the underlying DTMC $\mathbf{P} = [p_{i,j}]_{i,j \in S}$, where $S = \{1, \ldots, d\}$, possesses a (time averaged) steady state probability distribution [30]. Let us suppose that we aim at executing P in parallel with another process which proceeds with a different action frequency. As already explained in Sect. 1.3, this could be done through a standard approach based on a synchronous parallel composition by adequately scaling the time unit on which the specification of the process is based, i.e. by splitting each action of P in a certain number n of subactions. Let us consider the effect of such an action split on the DTMC $\mathbf{P}$ underlying the process specification. Considered a state i of the DTMC $\mathbf{P}$, the split is applied to all the transitions outgoing from i. It is easy to see that, since in a DTMC each action takes one time unit to be executed, scaling the time unit by a factor $1/n$ corresponds to expand i in n states representing the passage of time while the system sojourns in i. More precisely, for each state i we add $n-1$ states $i^1, \ldots, i^{n-1}$ and we modify state transitions as follows. The outgoing transitions of i^{n-1} are all the transitions previously leaving i, the incoming transitions of i are preserved, and, defined $i^0 = i$, for each $k \in \{1, \ldots, n-1\}$ we have a transition going from i^{k-1} to i^k with probability 1.



state i state i expanded

For the sake of simplicity, let us consider the case $n = 2$, i.e. we split each action into two subactions. The new DTMC $\mathbf{P}' = [p'_{i,j}]_{i,j \in S'}$ where $S' = \{1, \ldots, 2 \cdot d\}$ can be represented as follows. Let us suppose that each state i^1 added when expanding a state i of $\mathbf{P}'$ is numbered in $\mathbf{P}'$ by $d + i$, hence in $\mathbf{P}'$ the first d states are those of $\mathbf{P}$ and the states from $d+1$ to $2 \cdot d$ are those added in the expansion procedure. According to such a procedure $\mathbf{P}'$ turns out to be as shown below:

$$
\begin{array}{c l}
\begin{array}{cc} 1 \;\ldots\; d & d{+}1 \ldots 2{\cdot}d \end{array} & \\
\left[\begin{array}{c|c} 0 & \mathrm{I}_d \\ \hline \mathrm{P} & 0 \end{array}\right] & \begin{array}{l} 1 \\ \vdots \\ d \\ d+1 \\ \vdots \\ 2{\cdot}d \end{array}
\end{array}
$$

where I_d is the identity matrix of size d.

Now, let us evaluate the (time averaged) steady state behavior of the DTMC $\mathbf{P}'$. Called $\boldsymbol{\pi} = [\pi_i]_{i \in S}$ the steady state probability vector of $\mathbf{P}$, it turns out that in $\mathbf{P}'$ the stationary probability of a state $i \in S$ is equally partitioned between the states i and $d + i$ obtained by expanding i. Formally, the steady state probability vector $\boldsymbol{\pi'} = [\pi'_i]_{i \in S'}$ of $\mathbf{P}'$ is given by:

$$\pi'_i = \begin{cases} \pi_i/2 & \text{if } 1 \leq i \leq d \\ \pi_{i-d}/2 & \text{if } d+1 \leq i \leq 2 \cdot d \end{cases}$$

or more intuitively by:

$$\boldsymbol{\pi'} = [\ \boldsymbol{\pi}/2 \ \mid \ \boldsymbol{\pi}/2\ ]$$

It is immediate to verify that the multiplication of such vector $\boldsymbol{\pi'}$ by the matrix $\mathbf{P}'$ yields the vector $\boldsymbol{\pi'}$ again.

In general for an arbitrary time unit scaling factor $1/n$ we represent the DTMC $\mathbf{P}' = [p'_{i,j}]_{i,j \in S'}$ where $S' = \{1, \ldots, n \cdot d\}$ as follows. Each state i^k added when expanding a state i of $\mathbf{P}'$ is numbered in $\mathbf{P}'$ by $k \cdot d + i$, hence in $\mathbf{P}'$ the first d states are those of $\mathbf{P}$, the states from $d+1$ to $2 \cdot d$ represent the second step of the expansion of the states of $\mathbf{P}$ and so on. Therefore in general $\mathbf{P}'$ is given by:

$$\begin{array}{c} \begin{array}{cc} 1 \ldots d & d{+}1 \ldots n \cdot d \end{array} \\ \left[\begin{array}{c|c} 0 & \mathrm{I}_{(n-1)\cdot d} \\ \hline \mathrm{P} & 0 \end{array}\right] \end{array} \begin{array}{l} 1 \\ \vdots \\ (n-1)\cdot d \\ (n-1)\cdot d+1 \\ \vdots \\ n \cdot d \end{array}$$

In $\mathbf{P}'$ the stationary probability of a state $i \in S$ is equally partitioned between the states $i, d+i, 2 \cdot d+i, \ldots, (n-1) \cdot d+i$ obtained by expanding i. This because it is immediate to verify that the steady state probability vector $\boldsymbol{\pi'} = [\pi'_i]_{i \in S'}$ of $\mathbf{P}'$ turns out to be:

$$\boldsymbol{\pi'} = [\ \underbrace{\boldsymbol{\pi'}/n \ \mid \ \ldots \ \mid \ \boldsymbol{\pi'}/n}_{n \text{ times}}\ ]$$

Now, given an arbitrary performance measure m that we want to evaluate for the process P, m can be expressed by considering a reward structure $\mathbf{B} = [b_{i,j}]_{i,j \in S}$ corresponding to m (see Sect.2). If $\mathbf{B}' = [b'_{i,j}]_{i,j \in S}$ is the reward structure obtained from $\mathbf{B}$ by scaling the time unit by a factor $1/n$, then $\mathbf{B}'$ associates rewards to transitions between expanded states of S, i.e. $b'_{i,j}$ is the reward associated to the transition from the expanded state i to the expanded state j. More precisely the reward $b'_{i,j}$ is associated to the transition representing the execution of the final step of the transition from i to j, i.e. to the transition from state $i^{n-1} = (n-1) \cdot d + i$ to state j, while, for each $k \in 1, \ldots, n-1$ the transitions going from states i^{k-1} to states i^k are endowed with reward zero. Therefore we actually consider the reward structure $\mathbf{B}'' = [b''_{i,j}]_{i,j \in S'}$ depicted below:

$$\begin{array}{cc} \begin{array}{cc} 1 \ldots d & d{+}1 \ldots n \cdot d \end{array} & \\ \left[\begin{array}{c|c} 0 & 0 \\ \hline \mathrm{B'} & 0 \end{array}\right] & \begin{array}{l} 1 \\ \vdots \\ (n-1) \cdot d \\ (n-1) \cdot d + 1 \\ \vdots \\ n \cdot d \end{array} \end{array}$$

The performance measure m is then given by:

$$m = \sum_{i \in S'} \sum_{j \in S'} \pi'_i \cdot b''_{i,j} \cdot p'_{i,j}$$

Since $b''_{i,j}$ is not null only if $i \in \{(n-1) \cdot d + 1, \ldots, n \cdot d\}$ and $j \in S$; and for all $i \in \{(n-1) \cdot d + 1, \ldots, n \cdot d\}$ and $j \in S$ we have that $b''_{i,j} = b'_{i-(n-1) \cdot d, j}$, $p'_{i,j} = p_{i-(n-1) \cdot d, j}$ and $\pi'_i = \pi_{i-(n-1) \cdot d}/n$, it turns out:

$$m = \sum_{i \in S} \sum_{j \in S} \frac{\pi_i}{n} \cdot b'_{i,j} \cdot p_{i,j} = \frac{1}{n} \cdot \sum_{i \in S} \sum_{j \in S} \pi_i \cdot b'_{i,j} \cdot p_{i,j} \qquad (*)$$

Example 5.1 Let us consider a communication system composed of a processing unit that receives messages from an incoming channel and after some internal computation it sends out them to an outgoing channel. Suppose that P is a specification of such a system where the time unit is considered to be a second, i.e. an action takes one second to be executed, and $\mathbf{P} = [p_{i,j}]_{i,j \in S}$ is the underlying DTMC.

Suppose that we want to evaluate the throughput of the system in terms of the number of messages sent out per time unit. In order to do this we consider a reward structure $\mathbf{B_t} = [b_{i,j}]_{i,j \in S}$ determined as follows. We associate a reward equal to 1 to each action representing the sending of a message and a reward equal to 0 to each other action of the system specification. As explained in Sect. 2, called $\boldsymbol{\pi} = [\pi_i]_{i \in S}$ the vector of the steady state probabilities of $\mathbf{P}$, the value m_t of the throughput of the system is given by $\sum_{i \in S} \sum_{j \in S} \pi_i \cdot b_{i,j} \cdot p_{i,j}$. Now, let us suppose that we want to express the behavior of the system with respect to a different time unit, e.g. tenth of seconds instead of seconds. We may need to do this because we want to execute it in parallel with another process whose specification is made in tenth of seconds. As already explained, scaling the time unit by a factor 1/10 can be made by splitting each action a of P in 10 subactions (9 *idle* actions followed by action a), thus obtaining a scaled DTMC $\mathbf{P}'$. The reward structure $\mathbf{B}'_\mathbf{t}$ that we consider for the time scaled system is unchanged with respect to the reward structure $\mathbf{B_t}$ considered for the original system (we associate reward 0 to *idle* actions). This because the reward gained by actions is not related with their duration but it is just used to count the occurrences of the actions, i.e. rewards 0 and 1 associated to actions are not durations expressed in seconds, but just numbers. By calculating the throughput of the time scaled system with the formula $(*)$ we obtain the value $m_t/10$, i.e. one tenth of the throughput of the original system. This is an expected result because the throughput is a frequency which is expressed in number of actions executed per time unit and we changed the time unit from seconds to tenth of seconds.

Now let us suppose that we want to evaluate the utilization of the processing unit in terms of the percentage of time occupied by the system in performing internal computations for message processing. We consider a reward structure $\mathbf{B_u} = [b_{i,j}]_{i,j \in S}$ determined by associating a reward 1 to each action representing an internal computation of the processing unit and a reward equal to 0 to each other action of the system specification. Let us call m_u the value of the utilization of the processing unit derived from $\mathbf{B_u}$. Now, similarly as in the previous case, let us suppose that we want to scale the time unit by a factor $1/10$. Considered the DTMC $\mathbf{P}'$ obtained by expanding the states of $\mathbf{P}$, we have to evaluate the reward structure $\mathbf{B}'_\mathbf{u}$ for the time scaled system. Differently from the case of the message throughput, in the definition of $\mathbf{B_u}$ the reward gained by an action is related to the duration of the activity it represents, i.e. rewards 0 and 1 are durations expressed in seconds. Hence, when we consider as the time unit tenth of seconds instead of seconds each reward must be multiplied by 10: it is like as all the subactions obtained by splitting an action with reward 1 of the original system would contribute to the evaluation of the utilization of the processing unit by all inheriting reward 1. Therefore we have $\mathbf{B}'_\mathbf{u} = \mathbf{B_u} \cdot 10$. By calculating the utilization of the time scaled system with the formula $(*)$ we obtain the value m_u, i.e. the same utilization of the original system. This is an expected result because the percentage of utilization of the processing unit does not change if we scale the time unit for each activity of the system. ■

5.2 Exact Time Unit Scaling via Probabilistic Advancing Speed

Now we will show that the above analysis of steady state based performance measures gives the same results when the time scaling of P is probabilistic instead of being exact. By employing our probabilistic parallel composition operator we can approximate the scaling of a factor $1/n$ of the time unit used in a specification P by executing P with a probabilistic action frequency $1/n$. This is obtained by considering, e.g., the term $P \parallel^p_\emptyset \mathit{Idle}$, where $\mathit{Idle} \triangleq \mathit{idle}.\mathit{Idle}$ is a process which repeatedly executes the action *idle* and $p = 1/n$.

Let us see what is the effect of executing process P with a probabilistic action frequency p on the DTMC $\mathbf{P}$ underlying the process specification. The set of states is unchanged, in particular the states of the GRTS underlying $P \,\|^p_\emptyset\, \mathit{Idle}$ are of the form $P' \,\|^p_\emptyset\, \mathit{Idle}$, where P' is a state of the semantics of P. The transitions leaving a state i of the DTMC $\mathbf{P}$ change as follows. A new transition corresponding to the execution of the action *idle* is added, which is executed with probability $1 - p$ and goes to i itself (a self-loop). The probability of every old transition leaving i is multiplied by p. Therefore the new DTMC $\mathbf{P}' = [p'_{i,j}]_{i,j \in S}$ where $S = \{1, \ldots, d\}$ turns out to be as follows:

$$p'_{i,j} = \begin{cases} p_{i,j} \cdot p + (1 - p) & \text{if } i = j \\ p_{i,j} \cdot p & \text{if } i \neq j \end{cases}$$

or equivalently:

$$\mathbf{P}' = \mathbf{P} \cdot p + I_d \cdot (1 - p)$$

where I_d is the identity matrix of size d.

Now, let us evaluate the (time averaged) steady state behavior of the DTMC $\mathbf{P}'$. It turns out that the stationary probability vector $\boldsymbol{\pi}$ of $\mathbf{P}$ represents the state probabilities at a steady behavior also for the new matrix $\mathbf{P}'$. This can be easily seen as follows:

$$\begin{aligned} \boldsymbol{\pi} \cdot \mathbf{P}' &= \boldsymbol{\pi} \cdot (\mathbf{P} \cdot p + I \cdot (1 - p)) \\ &= (\boldsymbol{\pi} \cdot \mathbf{P}) \cdot p + (\boldsymbol{\pi} \cdot I) \cdot (1 - p) \\ &= \boldsymbol{\pi} \cdot p + \boldsymbol{\pi} \cdot (1 - p) \\ &= \boldsymbol{\pi} \end{aligned}$$

Now, suppose that we scale the time unit of process P with our probabilistic parallel operator and we want to evaluate the performance measure m expressed by the reward structure $\mathbf{B}' = [b'_{i,j}]_{i,j \in S}$ obtained from $\mathbf{B} = [b_{i,j}]_{i,j \in S}$ by scaling the time unit of a factor $1/n$. In the DTMC $\mathbf{P}'$ obtained from $P \,\|^p_\emptyset\, \mathit{Idle}$ the reward $b'_{i,j}$ is associated to the transitions representing moves of P that go from state i to state j, while we associate reward zero to the transitions derived from the moves *idle* of *Idle*. Since in $\mathbf{P}'$ the value $p'_{i,j}$ represents the total probability of going from state i to state j, in the case $i = j$ it accounts also for transitions going from i to itself due to the execution of the *idle* action. Therefore, according to the formula presented in Sect. 2, the total reward that we must associate with such an event is given by $b'_{i,i}$ multiplied by the probability that the passage from state i to itself is due to a self-loop representing a move of P (and not to an *idle* action). Such a probability is given by $p_{i,i} \cdot p$, i.e. the probability associated to the execution of a self-loop representing a move of P, divided by the total probability $p'_{i,i}$. Therefore for the DTMC $\mathbf{P}'$

we consider the reward structure $\mathbf{B}'' = [b''_{i,j}]_{i,j \in S'}$ defined as follows:

$$b''_{i,j} = \begin{cases} b'_{i,j} \cdot \frac{p_{i,j} \cdot p}{p'_{i,j}} & \text{if } i = j \\ b'_{i,j} & \text{if } i \neq j \end{cases}$$

The performance measure m is then given by:

$$m = \sum_{i \in S} \sum_{j \in S} \pi_i \cdot b''_{i,j} \cdot p'_{i,j}$$

Since it holds that $b''_{i,j} \cdot p'_{i,j} = b'_{i,j} \cdot p_{i,j} \cdot p$ both for $i = j$ and for $i \neq j$, it turns out:

$$m = \sum_{i \in S} \sum_{j \in S} \pi_i \cdot b'_{i,j} \cdot p_{i,j} \cdot p = p \cdot \sum_{i \in S} \sum_{j \in S} \pi_i \cdot b'_{i,j} \cdot p_{i,j} \qquad (**)$$

which, recalling that we assumed $p = 1/n$, is the same formula that we obtained with the standard synchronous approach.

Example 5.2 Let us consider the communication system P of the previous example 5.1. We recall that in P the time unit is considered to be a second and $\mathbf{P} = [p_{i,j}]_{i,j \in S}$ is the DTMC underlying P. Now we show how to evaluate the two performance measures of example 5.1 by employing our approach based on probabilistic parallel composition. We scale the time unit of P to tenth of seconds by executing P with a probabilistic action frequency 1/10. In particular we consider the term $P \,||_{\emptyset}^{1/10}\, \mathit{Idle}$ and the DTMC $\mathbf{P'}$ obtained from such a term. Let us consider the throughput of the system as defined by the reward structure $\mathbf{B_t}$ of example 5.1. As explained in example 5.1, since $\mathbf{B_t}$ is not based on action durations, the reward structure $\mathbf{B'_t}$ that we must apply to the time scaled system is the same as $\mathbf{B_t}$. If we instead consider the utilization of the processing unit as defined by the reward structure $\mathbf{B_u}$ of example 5.1 we must proceed as follows. As explained in example 5.1, since rewards are related to durations of actions, the reward structure $\mathbf{B'_u}$ that we must apply to the time scaled system is $\mathbf{B'_u} = \mathbf{B_u} \cdot 10$. Both in the case of the system throughput and in the case of the processing unit utilization, by using the same reward structures for the time scaled system $\mathbf{B'_t} = \mathbf{B_t}$ and $\mathbf{B'_u} = \mathbf{B_u} \cdot 10$ as those considered in Example 5.1, since we showed that formula $(**)$ coincides with formula $(*)$, we obtain the same performance measures as with the approach based on action splitting. ■

Finally it is worth noting that, with respect to the approach of Sect 5.1 based on action splitting, scaling the time unit via our probabilistic parallel operator gives the new possibility of using scaling factors p which are not of the form $p = 1/n$ with n natural number. In general $P \,||_{\emptyset}^{p}\, \mathit{Idle}$, when considered at the steady state, scales the time unit u_P used in the specification of P of a factor p, i.e. $u = u_P \cdot p$ is the time unit representing action duration in the behavior of $P \,||_{\emptyset}^{p}\, \mathit{Idle}$. This because $P \,||_{\emptyset}^{p}\, \mathit{Idle}$ executes a mean of p actions of P per time unit u, hence the mean action frequency f_P of P is given by $f_P = p/u$.

Since an action of P takes a mean of $u_P = 1/f_P$ time to be executed we have $u = u_P \cdot p$. It is easy to see that also in the general case the performance measures obtained from $P \,\|^p_\emptyset\, Idle$ at the steady state are the same obtained with an exact time unit scaling of factor p. In the case of performance measures representing a frequency (as e.g. the throughput of the system considered in examples 5.1 and 5.2), since the reward structure $\mathbf{B}'$ considered for the scaled system is the same as that ($\mathbf{B}$) of the original system, the formula ($**$) yields $m \cdot p$, where m is the value of the measure in the original system. The value $m \cdot p$ is correct for the scaled system because when we take time units into account, we have $m/u_P = (m \cdot p)/u$. In the case of performance measures representing percentage of time (as e.g. the utilization of the processing unit in examples 5.1 and 5.2), since the reward structure $\mathbf{B}'$ considered for the scaled system is determined from that ($\mathbf{B}$) of the original system by letting $\mathbf{B}' = \mathbf{B}/p$, according to formula ($**$) the value of the measure in the scaled system is the same as in the original system. This is correct because a percentage value does not refer to the time unit considered.

Summing up, we have the following theorem.

Theorem 5.3 Let $P \in \mathcal{G}$ be a fully specified system such that the underlying DTMC possesses a time averaged steady state probability distribution. The steady state based performance measures of $P \,\|^p_\emptyset\, Idle$ expressible by attaching rewards to the generative actions of P are exactly as those derived by executing the generative actions of P with an exact frequency p. [8] ■

5.3 Exact Time Unit Scaling of System Components

Now we extend the result above by showing that even in the case of non-blocking processes which are part of a larger system (hence not performance closed processes), our probabilistic parallel operator determines, when the system is considered at steady state, an exact scaling of the time unit on which the specification of the processes in isolation are based. As a consequence we can express the parallel composition of processes specified with respect to different time units (hence proceeding at different exact action frequencies) through a common time unit by considering a different scaling factor for the time unit of each process. In Sect. 3.2 we showed that given two probabilistic action frequencies f_P and f_Q there exists a time unit u and a probability p such that in $P \,\|^p_S\, Q$ the processes P and Q advance with mean action frequencies f_P and f_Q respectively. What we show now is that, if we suppose that both processes P and Q never block during the execution of $P \,\|^p_S\, Q$ (in all the states

[8] In this theorem and in the following Theorem 5.4 we consider performance measures of periodic DTMCs to be evaluated from their time averaged steady state probabilities [30].

of $P \parallel^p_S Q$ at least one generative action of P and one generative action of Q are executable), at the steady state the representation of system behavior given by $P \parallel^p_S Q$ is correct even when the action frequencies f_P and f_Q are considered as being exact instead of probabilistic, i.e. when each action of P (Q) is assumed to take exactly $u_P = 1/f_P$ ($u_Q = 1/f_Q$) time to be executed. In other words, if we assume that the specifications of processes P and Q in isolation are based on two different time units u_P and u_Q representing action duration, then $P \parallel^p_S Q$ just expresses, by performing an exact scaling of the time units u_P and u_Q to the new common time unit u, the concurrent execution of the two processes without affecting their behavior (apart from the consequences of action synchronization). In particular, as explained in Sect 5.2, u_P turns out to be scaled of a factor p ($u = u_P \cdot p$) and similarly u_Q turns out to be scaled of a factor $1 - p$.

In the following we show how the result of Theorem 5.3 can be smoothly extended to non-blocking system components.

In Sect 5.2 we showed that, supposed P a fully specified system such that the underlying DTMC possesses a (time averaged) steady state distribution, the term $P \parallel^p_\emptyset \mathit{Idle}$, where $\mathit{Idle} \triangleq \mathit{idle}.\mathit{Idle}$, at the steady state correctly represents the system behavior even when we execute exactly p actions of P per time unit u.

We generalize this result by first considering the case in which a process P with all the properties above is in parallel with an arbitrary process Q with the same properties (instead of the particular process *Idle*) but we have no synchronization between P and Q. It is trivial to see that, since $P \parallel^p_\emptyset \mathit{Idle}$ at steady state may be interpreted as executing exactly p actions of P per time unit u, then the same holds also for $P \parallel^p_\emptyset Q$. This because, since both processes P and Q enable at least one generative action in each system state (otherwise they could not be fully specified processes), given a state P' of process P, each system state $P' \parallel^p_\emptyset Q'$ (for any state Q' of Q) executes with probability p a transition of P' and with probability $1 - p$ a transition of Q'. Therefore the particular behavior of process Q in the states Q' reached through state changes while process P sojourns in state P' does not affect the behavior of P. Hence as far as the behavior of P is concerned considering the process $\mathit{Idle} \triangleq \mathit{idle}.\mathit{Idle}$ which has a single state is just like considering any arbitrary process Q with the properties above. More formally it is easy to see that in $P \parallel^p_\emptyset Q$ when we replace Q by the *Idle* process the sum of the steady state probabilities of being in states $P' \parallel^p_\emptyset Q'$ for any state Q' of Q becomes the steady state probability of being in the state $P' \parallel^p_\emptyset \mathit{Idle}$. Since in each state $P' \parallel^p_\emptyset Q'$ the process P behaves in the same way (as in $P' \parallel^p_\emptyset \mathit{Idle}$), at the steady state the behavior of process P in $P \parallel^p_\emptyset Q$ is the same as in $P \parallel^p_\emptyset \mathit{Idle}$. Note that by a symmetric argument we have also that $P \parallel^p_\emptyset Q$ at steady state executes exactly $1 - p$ actions of Q per time unit u.

Now we consider a more general case in which the process P may also synchronize with process Q. More precisely we consider a system $P \,\|_S^p\, Q$ such that both P and Q (which are no longer required to be fully specified processes) never get blocked during execution, i.e. in all the states of $P \,\|_S^p\, Q$ at least one generative action of P and one generative action of Q are executable. Since, whenever a process is a non interacting system component executed with mean action frequency p, at steady state it may be interpreted as executing exactly p generative actions per time unit u, it is easy to see that the same holds when the interacting process P is executed by $P \,\|_S^p\, Q$. This because, with respect to the execution of P with no interactions, the synchronization requirement with the reactive actions of process Q may change the set of generative actions of P executable in a system state, but, since P and Q never get blocked, not the frequency p at which generative actions of P are executed. More precisely, since in every system state both processes P and Q may execute at least one generative action, the probability for P to be the process performing the next generative action is always p. Therefore independently on which are the generative actions of P actually enabled in system states, we have that the frequency of firings of P generative transitions is still exactly p per time unit u. By a symmetric argument we have also that the frequency of firings of Q generative transitions is exactly $1 - p$ per time unit u.

Summing up, we have the following theorem.

Theorem 5.4 Supposed that both $P \in \mathcal{G}$ and $Q \in \mathcal{G}$ never block during the execution of $P \,\|_S^p\, Q$ (in all the states of $P \,\|_S^p\, Q$ at least one generative action of P and one generative action of Q are executable), the steady state based performance measures of $P \,\|_S^p\, Q$ expressible by attaching rewards to the generative actions of P and Q are exactly as those derived by executing the generative actions of P and Q with an exact frequency p and $1-p$, respectively. ■

6 A Case Study: Multi-Path Routing

In this section we present a case study showing how our approach provides a modular, compositional, and intuitive method for specifying concurrent systems in a scalable way. In particular, we consider a multi-path routing mechanism of the OSI network layer [29], and we model and analyze an internetworking node, whose arriving packets have several possible destinations with several possible ways to reach a destination. In the following, we refer to the node as the Interface Message Processor (IMP).

Before presenting the model, let us briefly recall some assumptions used in routing protocols. The routing algorithm decides, at the network layer, on which output link an arriving packet should be sent, depending on the destination of that packet. We abstract from the particular algorithm used to determine an optimal routing path between two nodes of a network, and we assume that the modeled IMP has a routing table including the route information with several possible choices for each destination. A weight is associated to each possible path and these weights are used as probabilities to decide where to send the present packet. Supporting multiple paths to the same destination, unlike single-path algorithms, permits traffic multiplexing over multiple lines. The advantages of multi-path are obvious: they can provide substantially better throughput and reliability.

6.1 Algebraic Specification of the Multi-Path Router

The overall model of our IMP (term *Multipath*) is shown in Table 5 in the particular case of two possible destinations (a and b) and two possible paths for each destination (a_1, a_2 for a, and b_1, b_2 for b). [9]

The algebraic specification is composed of several processes which are actually concurrent and are specified with respect to different time units. In particular, system *Multipath* consists of three concurrent components: a term *Arrivals* modeling the incoming traffic, a term *Router* modeling the core of the IMP, and a term *Channels* modeling the outgoing channels. The structure of the three components is as follows.

- The term *Arrivals* is composed of two concurrent processes *Arrivala* and *Arrivalb* which model the incoming traffic directed to destinations a and b, respectively. The time unit representing action duration that we consider for both processes is one millisecond. The adoption of such a time unit makes it easy to represent a realistic workload for the IMP. In particular we assume that for each destination at most one packet per millisecond can arrive to the IMP, i.e. the maximum frequency of the incoming traffic is 2000 packets per second.
- The term *Router* represents a process whose time unit is half a microsecond, meaning that it can execute 2000000 actions per second. As we will see, the *Router* term, which is the core of the IMP, is a single processor machine managing the packets directed to the two destinations a and b via a probabilistic scheduler.
- Finally, the term *Channels* is composed of four concurrent processes modeling the four possible outgoing channels a_1, a_2, b_1, and b_2. Since we take packet transmission to be represented by the execution of a corresponding action, their time units are defined on the basis of their bandwidth. The

[9] In Table 5 we omit the parameters of the probabilistic operators if they are not meaningful for the system specification.

$$
\begin{aligned}
Multipath &\triangleq Arrivals \parallel_S^{p_1} (Router \parallel_C^{p_2} Channels)\\
S &= \{receivea,\ receiveb\}\\
C &= \{avail\_cha_1,\ avail\_cha_2,\ avail\_chb_1,\ avail\_chb_2,\\
&\qquad transma_1,\ transma_2,\ transmb_1,\ transmb_2\}\\
Arrivals &\triangleq Arrivala \parallel^{\frac{1}{2}} Arrivalb\\
Channels &\triangleq (Channel[\varphi'_1] \parallel^{pa} Channel[\varphi'_2]) \parallel^{v} (Channel[\varphi''_1] \parallel^{pb} Channel[\varphi''_2])\\
Router &\triangleq (Queues \parallel_{S_1 \cup I} Switch) \parallel_I Idle\\
S_1 &= \{accepta,\ acceptb\} \qquad I \ = \ \{idle\}\\
Queues &\triangleq Queue[\varphi'] \parallel_I Queue[\varphi'']\\
Switch &\triangleq (Manager[\varphi'] \parallel_{S'_2} (Routing[\varphi'][\varphi'_1] \parallel_{B'}^{qa} Routing[\varphi'][\varphi'_2])\,) \quad \parallel_I^{p}\\
&\qquad (Manager[\varphi''] \parallel_{S''_2} (Routing[\varphi''][\varphi''_1] \parallel_{B''}^{qb} Routing[\varphi''][\varphi''_2])\,)\\
S'_2 &= \{senda,\ busya\} \qquad S''_2 \ = \ \{sendb,\ busyb\}\\
B' &= \{busya\} \qquad B'' \ = \ \{busyb\}\\
\varphi' &= \langle (receive,\ receivea),\ (accept,\ accepta),\ (send,\ senda),\ (busy,\ busya)\rangle\\
\varphi'' &= \langle (receive,\ receiveb),\ (accept,\ acceptb),\ (send,\ sendb),\ (busy,\ busyb)\rangle\\
\varphi'_i &= \langle (transm,\ transma_i),\ (avail\_ch,\ avail\_cha_i)\rangle \quad i \in \{1,2\}\\
\varphi''_i &= \langle (transm,\ transmb_i),\ (avail\_ch,\ avail\_chb_i)\rangle \quad i \in \{1,2\}\\
Arrivala &\triangleq receivea.Arrivala +^{ra} wait.Arrivala\\
Arrivalb &\triangleq receiveb.Arrivalb +^{rb} wait.Arrivalb\\
Queue &\triangleq receive_*.Queue' + idle_*.Queue\\
Queue' &\triangleq receive_*.accept_*.Queue' + accept_*.Queue\\
Manager &\triangleq accept.Manager' + idle_*.Manager\\
Manager' &\triangleq send.Manager + busy.Manager'\\
Routing &\triangleq send_*.Routing' + avail\_ch_*.Routing\\
Routing' &\triangleq transm_*.Routing + busy_*.Routing'\\
Idle &\triangleq idle.Idle\\
Channel &\triangleq avail\_ch.Channel + transm.Channel
\end{aligned}
$$

Table 5
Multi-Path Routing Model

time unit for the channel a_1 directed to a is 10 milliseconds, i.e. it can send out 100 packets per second, while the time unit for the channel a_2 directed to a is 2.5 milliseconds, i.e. it can send out 400 packets per second. The time unit for the channel b_1 directed to b is 5 milliseconds, i.e. it can send out 200 packets per second while the time unit for the channel b_2 directed to b is 4 milliseconds, i.e. it can send out 250 packets per second.

Note that it is possible to compose in parallel the above processes, which are specified with respect to different time units, because, as it can be easily verified, each of them never blocks during system execution (see Sect. 5.3). According to what we have explained in Sect. 5.3, in order to express the actual concurrent execution of such processes all the time units used in their specification are scaled to a common global time unit u. In particular u is evaluated by computing the inverse of the global action frequency of the composed system. Hence in our case study u is the inverse of $1000 + 1000 + 2000000 + 100 + 400 + 200 + 250$ actions per second, i.e. $u = 1/2002950$ seconds. The global time unit u adopted determines the action frequencies to be considered as parameters of the parallel operators used to describe the concurrent execution of system components in Table 5. In particular, parameter p_1 representing the advancing speed of term *Arrivals* in $\mathit{Arrivals} \,\|_S^{p_1} (\mathit{Router} \,\|_C^{p_2}\, \mathit{Channels})$ is given by the ratio of the action frequency of term *Arrivals* over the global action frequency of system *Multipath*, i.e. if we express action frequency in seconds $p_1 = 2000/2002950 \approx 0.000999$ (see Sect 3.2). Similarly, parameter p_2 representing the advancing speed of term *Router* in $\mathit{Router} \,\|_C^{p_2}\, \mathit{Channels}$ is given by the ratio of the action frequency of term *Router* over the global action frequency of term $\mathit{Router} \,\|_C^{p_2}\, \mathit{Channels}$, i.e. $p_2 = 2000000/(2000000 + 950) \approx 0.999525$. As far as the specification of the *Arrivals* component is concerned, the parallel composition of the two concurrent processes *Arrivala* and *Arrivalb* has parameter $\frac{1}{2}$ because their action frequency is the same (1000 actions per second). As far as the specification of the *Channels* component is concerned, in $(\mathit{Channel}[\varphi_1'] \,\|^{pa}\, \mathit{Channel}[\varphi_2']) \,\|^{v}\, (\mathit{Channel}[\varphi_1''] \,\|^{pb}\, \mathit{Channel}[\varphi_2''])$ we take: pa to be $100/(100{+}400)$, i.e. $pa = 0.2$; pb to be $200/(200{+}250)$, i.e. $pb \approx 0.444444$; and v to be $500/(500 + 450)$, i.e. $v \approx 0.526316$. It is easy to see that the parameters adopted for the parallel operators give rise to the correct action frequencies. For instance process *Arrivala* executes $p_1 \cdot \frac{1}{2} \approx 0.000499$ actions per time unit u, i.e. 1000 actions per second, and process *Router* executes $(1 - p_1) \cdot p_2 \approx 0.998527$ actions per time unit u, i.e. 2000000 actions per second.

Now let us describe in detail the behavior of each process of the system *Multipath*.

The process *Arrivala* (*Arrivalb*) models the incoming traffic through a Bernoulli distribution with parameter ra (rb). In particular, an arriving packet is represented by the action *receivea* (*receiveb*) which synchronizes with the corresponding reactive action in the queue for packets a (b) of term *Router*. In the case such a queue is full the action *receivea* (*receiveb*) is not enabled and the arriving packets are lost (the generative action *wait* is executed with probability 1).

The process *Router* is the core of the IMP and is composed of a term *Queues* collecting the arriving packets, a term *Switch* which delivers the packets to the outgoing channels and a term *Idle* modeling the phases of router inactivity. They are defined as follows.

- Term *Queues* consists of two *Queue* processes, one for each kind of packet, which behave reactively. In particular, they receive packets destined to a (b) through reactive actions of type *receivea* (*receiveb*) and pass them to the *Switch* term through reactive actions of type *accepta* (*acceptb*). For the sake of simplicity we assume both queues to be of size 2.
- Term *Switch* is a single-processor machine executing two different terms, each one managing packets with a certain destination (a or b), via a probabilistic scheduler with parameter p. In this way, by varying p, we can model an IMP that delivers packets with a particular destination more efficiently than packets with another destination, e.g. for commercial reasons. The term delivering packets to destination a (b) is composed of a *Manager* and two *Routing* terms, each one delivering packets to a particular channel a_1 or a_2 (b_1 or b_2). Term *Manager* accepts packets destined to a (b) from the dedicated queue through action *accepta* (*acceptb*) and afterwards either immediately passes them to one of the two *Routing* terms through action *senda* (*sendb*), or waits until at least one channel is available for transmission by performing action *busya* (*busyb*). This behavior is realized through a generative-reactive mechanism as follows. The two *Routing* terms behave reactively and each of them accepts packets through a reactive action of type *senda* (*sendb*) and transmits them through the corresponding channel via a reactive action of type *transma* (*transmb*). Whenever the generative action *senda* (*sendb*) is enabled by the *Manager* term, the *Routing* term accepting the packet is chosen according to the probability *qa* (*qb*) parameterizing the parallel composition of the two *Routing* terms. Note that a *Routing* term may be not available for accepting a packet because it is currently transmitting through action *transma* (*transmb*) a packet previously received. Therefore, whenever only one *Routing* process is available for accepting a packet coming from the *Manager* term, the packet is transmitted through the corresponding channel with probability 1. Whenever both *Routing* processes are busy the transmission of packets destined to a (b) is not possible and this is signalled to the *Manager* term through a multiway synchronization by enabling the reactive action of type *busya* (*busyb*) whose execution requires the synchronization of the two *Routing* processes.
- Term *Idle* executes an action *idle* (representing the fact that the IMP is idle) whenever term *Router* has nothing else to do. More precisely, action *idle*

is executed through a multiway synchronization with all the other *Router* components if and only if the input queues (term *Queues*) are empty and the core of the IMP (term *Switch*) is not waiting for delivering a packet to the channel. In particular, term *Idle* prevents the term *Router* from blocking, thus allowing the advancing speed of terms *Arrivals*, *Router*, and *Channels* to be preserved and satisfying the condition needed for composing processes with different time units (see Sect. 5.3).

The four *Channel* processes model the two outgoing channels a_1 and a_2 directed to destination a and the two outgoing channels b_1 and b_2 directed to destination b. Each process *Channel* can either be transmitting a packet when the generative action *transm* is synchronized with the corresponding reactive action of term *Routing* managing that channel, or be available for transmission when the generative action *avail_ch* is synchronized with the corresponding reactive action of term *Routing*. For instance in the case of channel a_1 the generative actions $transma_1$ and $avail\_cha_1$ must synchronize with the reactive actions $transma_{1_*}$ and $avail\_cha_{1_*}$ of the *Routing* term managing channel a_1, respectively. In this way the generative actions of a *Channel* process are executed in mutual exclusion in the sense that in every system state one and only one of them is enabled. As a consequence term *Channels* never blocks.

Thanks to our approach which allows processes specified with respect to different time units to be modeled without splitting actions (see Sect. 4), we have that the transition system underlying the algebraic specification of Table 5 is composed of only 576 states and 4768 transitions. This is a crucial result, because if we want to deal with the same system by resorting to a classical and intuitive approach which scales the time unit by splitting each action, we have to cope with the serious problem of a greatly increased size of the state space. For instance, since the basic time unit for the router is half a microsecond whereas the basic time unit for the input channels is a millisecond, in order to compose in parallel terms *Arrivals* and *Router* we have to split the actions of term *Arrivals* in thousands of subactions thus causing a state space explosion.

Moreover, we point out that the generative-reactive behavior of the *Switch* process represents the core of this case study. In particular, process *Switch* generatively decides, according to probability p, which of the two *Manager* terms performs a send action (*senda* for the manager delivering packets to destination a or *sendb* for the other one), while it reactively decides, according to probability qa or qb (depending on the send action performed) which of the term *Routing* synchronizes with such an action. A calculus capable of expressing generative-reactive choices is, therefore, very suitable (if not necessary) to model systems with such a behavior.

Finally it is worth noting that, thanks to the choice of putting probabilities in the operators (instead of, e.g., attaching them to actions) and to the expressive power of our generative-reactive approach, it was possible to specify the IMP in such a way that all the probabilistic mechanisms on which its behavior is based (and which are not related with the internal behavior of a process) depend on the parameters of parallel composition operators only. As a consequence scaling the system specification to a higher number of components does not make it necessary to change the internal behavior of processes. For instance in the router specification we can scale the system to a higher number of destinations or different channels for each destination by simply adding as many instances of *Switch* and *Channel* processes as we want and adjust appropriately the parameters of parallel composition operators. Here we consider the case of a router with 4 possible channels for each destination a and b instead of 2. In Table 6 we show the algebraic specification of the terms which are modified by such a scaling. Assumed that the bandwidth of the outgoing channels a_1, a_2, a_3, and a_4 is 50, 100, 150, and 200 packets per second, respectively, whereas the bandwidth of the outgoing channels b_1, b_2, b_3, and b_4 is 60, 120, 180, and 240 packets per second, respectively, the parameters of the parallel operators of Table 6 turn out to be as follows: $p1 \approx 0.000998$, $p2 \approx 0.998452$, $v \approx 0.454545$, $pa' = pb' \approx 0.333333$, $pa'' = pb'' = 0.3$, $pa''' = pb''' \approx 0.428571$. The transition system derived from such a specification is composed of 9216 states and 120640 transitions.

6.2 Performance Analysis

In order to derive performance measures from the multi-path router specification, we resorted to the software tool TwoTowers [8], that has been recently extended to support the generative-reactive approach presented in this paper. Such a tool also implements the algebraic reward based method described in Sect. 5 to specify and derive performance measures. The results of our performance analysis are shown in Fig. 5-8. In particular, we concentrated on two main metrics.

On the one hand we evaluate the throughput of the system at steady state, represented by occurrences of actions of type $transma_1$, $transma_2$, $transmb_1$, and $transmb_2$, by attaching a reward equal to 1 to the above actions and a reward equal to 0 to each other action. Since the throughput is a frequency expressed in terms of number of actions executed per time unit and the time unit is 1/2002950 seconds, we have to multiply the throughput resulting from the Markov chain analysis by 2002950 in order to obtain the results (expressed in seconds) shown in our tables.

On the other hand we evaluate the router idleness at steady state in terms of the percentage of time the IMP is inactive. The router is considered to be idle when no packet is currently inside the IMP, i.e. when it executes actions

$$
\begin{aligned}
\mathit{Multipath} &\triangleq \mathit{Arrivals} \parallel_S^{p_1} (\mathit{Router} \parallel_C^{p_2} \mathit{Channels})\\
C &= \{\mathit{avail\_cha}_1,\ \mathit{avail\_cha}_2,\ \mathit{avail\_cha}_3,\ \mathit{avail\_cha}_4,\\
&\quad\ \mathit{avail\_chb}_1,\ \mathit{avail\_chb}_2,\ \mathit{avail\_chb}_3,\ \mathit{avail\_chb}_4,\\
&\quad\ \mathit{transma}_1,\ \mathit{transma}_2,\ \mathit{transma}_3,\ \mathit{transma}_4\\
&\quad\ \mathit{transmb}_1,\ \mathit{transmb}_2,\ \mathit{transmb}_3,\ \mathit{transmb}_4\}\\
\\
\mathit{Channels} &\triangleq ((\mathit{Channel}[\varphi_1'] \parallel^{pa'} \mathit{Channel}[\varphi_2']) \parallel^{pa''}\\
&\quad\ (\mathit{Channel}[\varphi_3'] \parallel^{pa'''} \mathit{Channel}[\varphi_4']))\\
&\quad \parallel^{v}\\
&\quad ((\mathit{Channel}[\varphi_1''] \parallel^{pb'} \mathit{Channel}[\varphi_2'']) \parallel^{pb''}\\
&\quad\ (\mathit{Channel}[\varphi_3''] \parallel^{pb'''} \mathit{Channel}[\varphi_4'']))\\
\\
\mathit{Switch} &\triangleq (\mathit{Manager}[\varphi'] \parallel_{S_2'} ((\mathit{Routing}[\varphi'][\varphi_1'] \parallel_{B'}^{qa'} \mathit{Routing}[\varphi'][\varphi_2']) \parallel_{B'}^{qa''}\\
&\qquad\qquad\qquad (\mathit{Routing}[\varphi'][\varphi_3'] \parallel_{B'}^{qa'''} \mathit{Routing}[\varphi'][\varphi_4']))\\
&\quad \parallel_I^{p}\\
&\quad (\mathit{Manager}[\varphi''] \parallel_{S_2''} ((\mathit{Routing}[\varphi''][\varphi_1''] \parallel_{B''}^{qb'} \mathit{Routing}[\varphi''][\varphi_2'']) \parallel_{B''}^{qb''}\\
&\qquad\qquad\qquad (\mathit{Routing}[\varphi''][\varphi_3''] \parallel_{B''}^{qb'''} \mathit{Routing}[\varphi''][\varphi_4'']))\\
\varphi_i' &= \langle(\mathit{transm},\ \mathit{transma}_i),\ (\mathit{avail\_ch},\ \mathit{avail\_cha}_i)\rangle \quad i \in \{1,2,3,4\}\\
\varphi_i'' &= \langle(\mathit{transm},\ \mathit{transmb}_i),\ (\mathit{avail\_ch},\ \mathit{avail\_chb}_i)\rangle \quad i \in \{1,2,3,4\}
\end{aligned}
$$

Table 6
Multi-Path Routing Model with 4 Possible Channels per Destination

of type *idle*. Therefore we attach a reward equal to 1 to such actions and a reward equal to 0 to each other action. Since the time unit of the *Router* process (half a microsecond) is scaled by a factor $(1 - p_1) \cdot p_2 \approx 0.998527$ and the reward gained by actions is related to the duration of the corresponding activity expressed in half microseconds (see Sect. 4), due to the time unit change we must multiply each reward by $1/0.998527$ before analyzing the Markov chain.

For each conducted analysis, we assumed that the incoming traffic for each kind of destination a and b follows the same Bernoulli distribution of parameter $r = ra = rb$. The pictures are built by showing how the performance measure

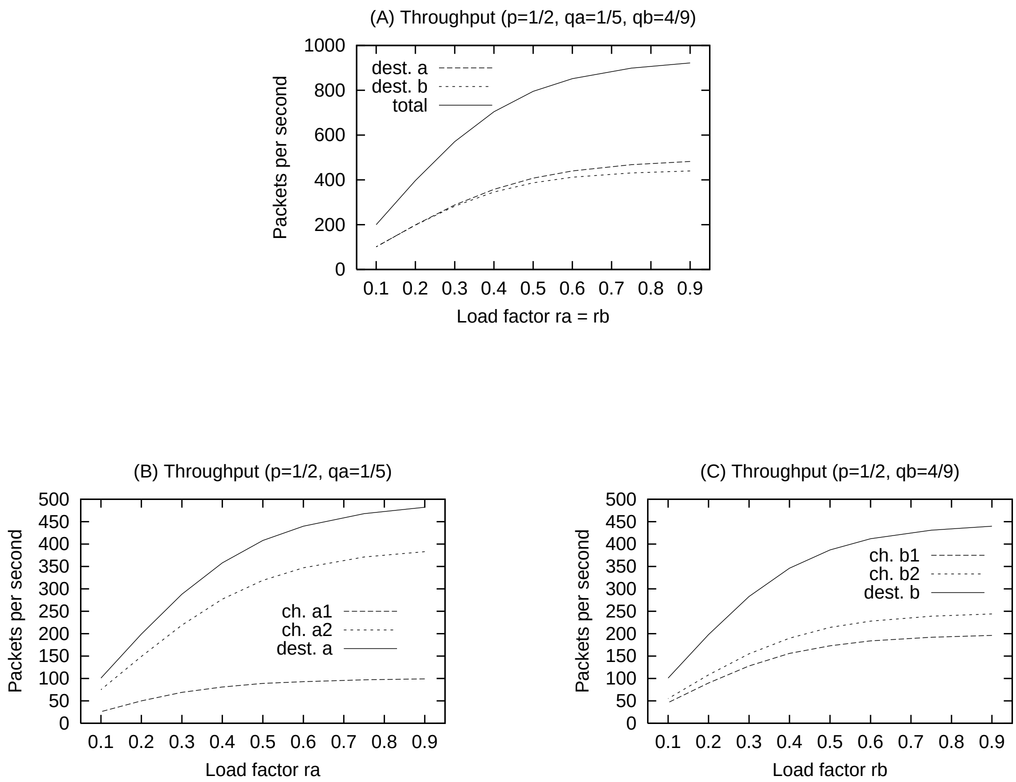


Fig. 5. Throughput of the Multi-Path Router

under analysis changes when we vary r from 0.1 (sometimes 0.01) to 0.9. In this way we can observe the system behavior under various levels of workload ranging from 10% (or 1%) to 90%.

We start by evaluating the system throughput under different circumstances. We first consider the situation in which $p = \frac{1}{2}$, i.e. the packets destined to a and b are managed at the same speed by the *Switch* process, and parameters qa and qb reflect the bandwidth distribution over channels directed to destinations a and b, respectively. In particular, since channel a_1 can deliver 100 packets per second and channel a_2 can deliver 400 packets per second, we take $qa = 100/(100+400) = 1/5$ (the ratio of the bandwidth of channel a_1 over the overall bandwidth of the channels directed to destination a), so that packets are probabilistically distributed between channels a_1 and a_2 in the optimal way. Similarly, since channel b_1 can deliver 200 packets per second and channel b_2 can deliver 250 packets per second, we take $qb = 200/(200+250) = 4/9$. The obtained results are reported in Fig. 5. As we can see in the first table of Fig. 5 the curve representing the total system throughput is characterized by a high slope in correspondence of a low workload and a quite flat slope when the load factor increases over the 50%. This because, for packets with a given destination, the bandwidth associated with the outgoing channels directed to that destination is about one half of the maximum bandwidth of the incoming traffic. Simply put, when the parameter r of the Bernoulli distribution representing the incoming traffic reaches the 50%, the outgoing channel is almost fully occupied, hence a further increment of r gives rise to a very small increment of the outgoing throughput. Another expected result that we can observe in Fig. 5 is that the throughput of packets destined to a is slightly greater than the throughput of packets destined to b. This because the overall bandwidth of the outgoing channels directed to a is 500 packets per second, while it is 450 packets per second for the outgoing channels directed to b. The other two tables of Fig. 5 report the throughput for each single channel a_1, a_2, b_1 and b_2. In the case of a_1 and a_2 the distance between the two curves is quite great, this because a_1 has just one fourth of the bandwidth of a_2. As expected such a difference is smaller in the case of b_1 and b_2, because their bandwidth is quite similar (200 and 250 packets per second, respectively).

Since in a realistic framework the value of parameters qa and qb are established by the multi-path routing algorithm governing the IMP according to the network conditions (e.g. estimated time for a packet to reach a destination via a particular path), we study the effect on the throughput of the router of adopting parameters qa and qb which do not reflect the bandwidth distribution over the outgoing channels. The results of such an analysis are reported in Fig. 6. The first table shows how the throughput of a_1 and a_2 varies when changing the value of qa from $\frac{1}{5}$ to $\frac{4}{5}$, i.e. by exchanging the value of qa and $1 - qa$. For the sake of clarity we report the curves obtained for both $qa = \frac{1}{5}$ and $qa = \frac{4}{5}$. We can observe that the parameter qa does not play a significant role when the router is congested. This because under a heavy workload both routing processes are hardly occupied and in most cases at least one of them is busy. In such a situation the parameter qa is often not used, because when a routing process is busy an arriving packet destined to a is passed to the other routing process with probability 1. As a consequence the curves of the throughput converge to the same values when the load factor ra gets over 50%, i.e. when almost all the bandwidth of each channel is exploited. On the other hand, when the incoming workload is low, the parameter qa becomes important as it probabilistically decides which routing process will deliver the packet, hence increasing the throughput of a routing process with respect to the other one (see the quite evident difference among the curves when ra gets under the 30%). This because in the presence of a low workload both routing processes stay idle for most of time and an arriving packet a is directed to a particular channel depending on the choice made according to qa. The second table of Fig. 6 shows how the throughput of the two channels destined to b varies when exchanging the value of qb and $1 - qb$. With respect to the case of the channels destined to a, in this case the difference between the old value of qb $(\frac{4}{9})$ and its new value $(\frac{5}{9})$ is smaller. This is reflected on the results presented in Fig. 6, where the curves for the old and the new value of qb are almost overlapped for each value of rb.

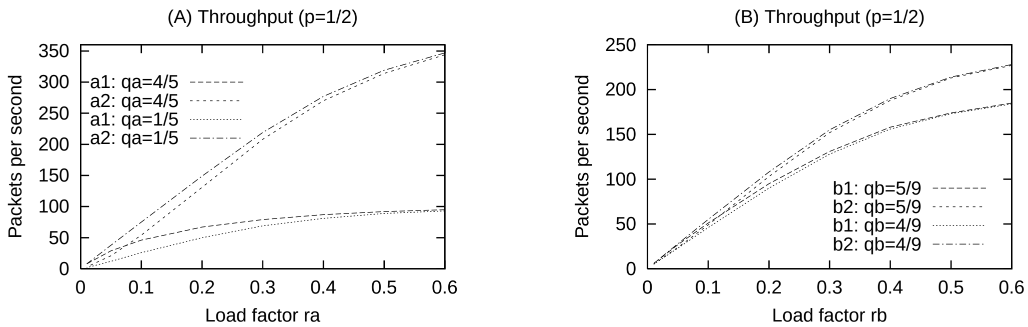


Fig. 6. Throughput Obtained by Varying the Probabilistic Choice between the *Routing* Processes

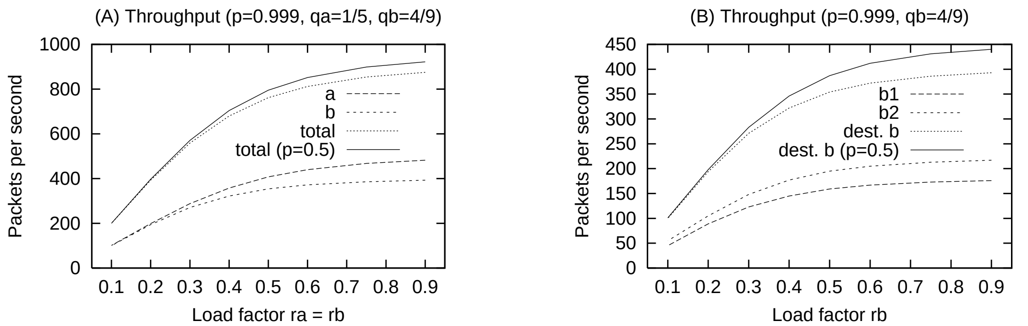


Fig. 7. Throughput Obtained by Varying Parameter p.

Now we show the role played by parameter p on the system throughput. In order to merely concentrate on the effects of varying parameter p we just consider the situation in which parameters qa and qb reflect the bandwidth distribution over channels directed to destinations a and b, respectively. Parameter p can be chosen in order to favour the internal computations of the IMP dedicated to packets destined to a (b) with respect to those dedicated to packets destined to b (a). To this aim, in Fig. 7 we report the throughput of the multi-path router in the case $p = 0.999$, hence when packets destined to b are managed by the IMP much more slowly than packets destined to a. As a consequence of the unfair behavior of the router, we have that the IMP delivers the packets destined to a at the usual speed (the curve for packets destined to a in the first table of Fig. 7 is the same as that in the first table of Fig. 5), but it delays the packets destined to b, hence compromising the throughput of such packets. Therefore with respect to the case $p = 0.5$ the overall system throughput decreases (for easy of comparison in the first

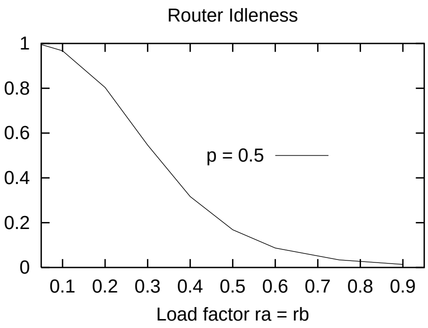


Fig. 8. Idleness of the Multi-Path Router

table of Fig. 7 we also report the curve obtained in the case $p = 0.5$). The comparison with the case $p = 0.5$ is even more evident in the second table of Fig. 7, where we report the throughput of the outgoing channels directed to destination b.

As far as the idleness of the router is concerned, we simply consider the situation in which $p = \frac{1}{2}$ and parameters qa and qb reflect the bandwidth distribution over channels directed to destinations a and b, respectively. The curve presented in Fig. 8 shows the relation among the inactivity of the router and the load factor for the incoming traffic. As expected, the router is almost always idle if the workload is low, but the duration of its inactivity phases rapidly converges to zero for a load factor greater than 40%.

Finally, we show in Fig. 9 the throughput and the idleness for the router with four possible channels for each destination presented in Table 6. Once again, the figures report the relation among the different metrics under the usual scenario where $p = \frac{1}{2}$ and parameters qa', qa'' and qa''', and qb', qb'' and qb''' reflect the bandwidth distribution over channels directed to destinations a and b, respectively (hence $qa' = qb' \approx 0.333333$, $qa'' = qb'' = 0.3$, $qa''' = qb''' \approx 0.428571$). As far as the throughput is concerned, since in the case of destination a we have preserved the overall outgoing bandwidth (500 packets per second) with respect to the router with two channels, the curve for destination a of the first and third table of Fig. 9 is the same as that we showed in Fig. 5. The total throughput for destination a is distributed among the four possible channels according to their bandwidth (see the third table of Fig. 9). On the other hand, the overall outgoing bandwidth for destination b is increased with respect to the router with two channels (600 packets per second vs. 450 packets per second). This is reflected on the curves of Fig. 9 which show that, in the case of the router with four channels for each destination, the total throughput for destination b is greater than the total throughput for destination a. The fourth table of Fig. 9 shows that, also in the case of b,

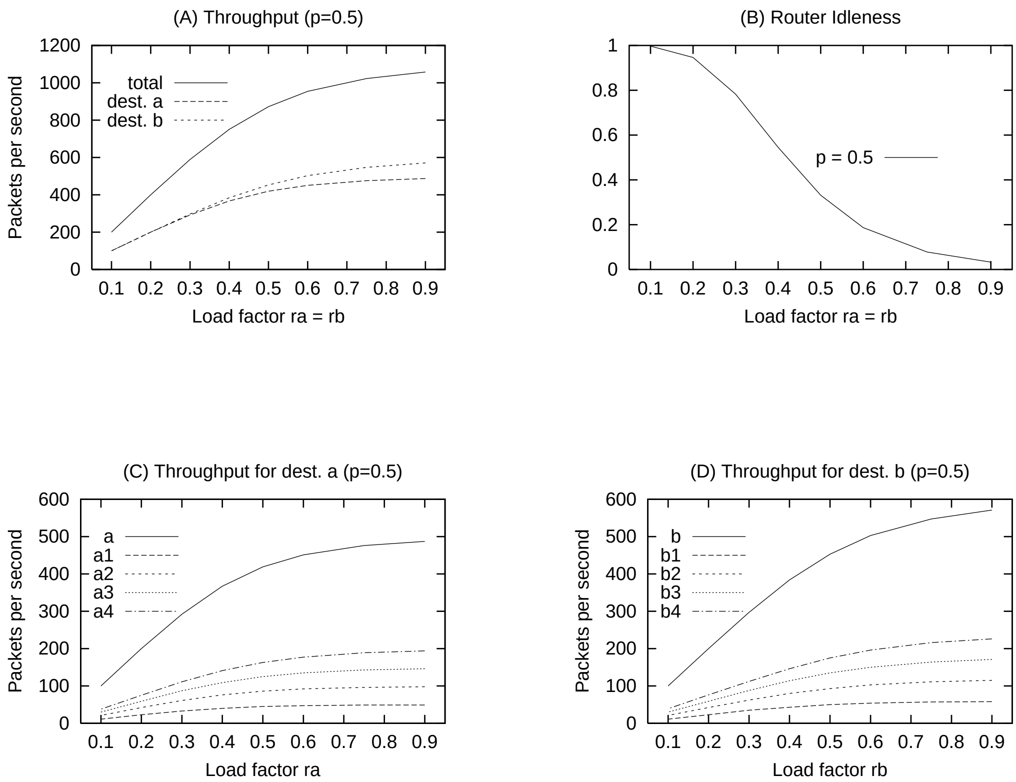


Fig. 9. Throughput and Idleness of the Multi-Path Router with 4 *Routing* per Destination.

the total throughput of packets destined to b is distributed among the four possible channels according to their bandwidth. As far as the router idleness is concerned, we can observe that the curve of the second table of Fig. 9 has the same shape as that seen in Fig. 8.

7 Conclusion

In this paper we have presented a discrete time probabilistic calculus which integrates in a simple way a hybrid between the generative and reactive approaches [15] with probabilistic parallel composition [5]. Our generative-reactive model seems to be an adequate solution for modeling probabilistic behaviors of real systems, as it profitably joins the characteristics of both the generative and the reactive models, and mechanisms like probabilistic internal/external choice and multiway synchronization. Moreover, adopting a probabilistic asynchronous parallel operator in a discrete time setting gives our calculus a great modeling power. In particular, depending on how we calculate the time unit to be considered for the composed system, the parallel operator has a twofold interpretation and allows us to model:

(1) multiple processes executed by a single processor machine with a probabilistic scheduling policy (in such a case, all parallel processes are assumed to be specified with respect to the same basic time unit as that representing action duration in the composed system),
(2) concurrent processes which proceed with different advancing speeds, meaning that, differently from the classical synchronous approach, they may adopt different basic durations for their actions.

In the latter case, when considering the behavior of the system at a steady state, we showed that the representation of system behavior arising from our parallel operator is correct also when process advancing speeds are considered as being exact instead of probabilistic. This means that, e.g., if the action frequency of a process is $1/n$ then each action of the process takes exactly n time units to be executed. Thanks to this result we can model systems whose components are specified with respect to different basic action durations without incurring the problem of state space explosion which arises with an intuitive application (based on action splitting) of a standard synchronous approach. For instance, our technique made it possible to analyze the algebraic model of a multi-path router with 8 outgoing channels whose components are specified with respect to largely different time units: the Interface Message Processor of the router is, e.g., specified with actions (representing process computations) whose duration is half a microsecond, while the outgoing channel a_1 is specified with actions (representing the time time to send a packet through the channel) whose duration is 10 milliseconds. Specifying the router with a synchronous parallel operator would have required splitting the actions of the outgoing channels into thousands of subactions, hence making the model hard (if not impossible) to be analyzed. Moreover, the idea of expressing the probabilistic advancing speed of processes by means of a parameter of the parallel operator (as in [5]) instead of, e.g., using weights attached to actions, has turned out to be adequate from the modeling viewpoint because:

(1) the modeler can first specify the behavior of processes in isolation and then establish, independently on how they are specified, their relative advancing speed when composing them in parallel,
(2) it leads to specifications which are easily scalable to any number of components (e.g. outgoing channels in our case study) without changing the internal behavior of processes.

As a further advantage of adopting a probabilistic parallel operator, we have that in the case of fully specified systems (where all the reactive actions get synchronized) the resulting generative-reactive transition system is fully probabilistic and can be trivially turned into a discrete time Markov chain. Therefore deriving performance measures from specifications of complete systems can be done by applying standard techniques. For instance, we have shown how to employ an algebraic reward based method for expressing performance measures of systems and we have applied it to evaluate the throughput and idleness of our multi-path router model.

Finally, we have obtained a sound and complete axiomatization for (a simple variant of) strong probabilistic bisimulation over nonrecursive processes of our calculus which:

(1) by employing a simple auxiliary operator expresses restriction of generative actions in a clean way without resorting to axioms with implications as in [5],
(2) in the case of reactive actions is a smooth extension of the axiom system for a nonprobabilistic standard process algebra which expresses parallel composition through the left merge and synchronization merge operators.

Note that we did not consider a weak version of our probabilistic bisimulation (see, e.g., [7]) because, since in the context of discrete time each transition takes one time unit to be executed, we cannot merge several τ transitions into a single τ transition while preserving the temporal behavior of the system.

Acknowledgements

This research has been partially funded by MURST Progetto TOSCA.

References

[1] L. Aceto, *"On Axiomatising Finite Concurrent Processes"*, in SIAM Journal on Computing 23(4):852-863, 1994

[2] L. Aceto, D. Murphy, *"On the Ill-Timed but Well-Caused"*, in Proc. of 4th Int. Conf. on Concurrency Theory, LNCS 715:97-111, Springer-Verlag, 1993

[3] A. Aldini, M. Bernardo, R. Gorrieri, *"An Algebraic Model for Evaluating the Performance of an ATM Switch with Explicit Rate Marking"*, in Proc. of 7th Int. Workshop on Process Algebras and Performance Modeling, Prensas Universitarias de Zaragoza, pp. 119-138, 1999

[4] A. Aldini, M. Bernardo, R. Gorrieri, M. Roccetti, *"Comparing the QoS of Internet Audio Mechanisms via Formal Methods"*, in ACM Transactions on Modeling and Computer Simulation 11:1-42, 2001

[5] J.C.M. Baeten, J.A. Bergstra, S.A. Smolka, *"Axiomatizing Probabilistic Processes: ACP with Generative Probabilities"*, in Information and Computation 121:234-255, 1995

[6] J.C.M. Baeten, W.P. Weijland, *"Process Algebra"*, Cambridge University Press, 1990

[7] C. Baier, H. Hermanns, *"Weak Bisimulation for Fully Probabilistic Processes"*, in Proc. of 9th Int. Conf. on Computer Aided Verification, LNCS 1254:119-130, Springer-Verlag, 1997

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

[9] M. Bernardo, M. Bravetti, *"Functional and Performance Modeling and Analysis of Token Ring using EMPA"*, in Proc. of 6th Italian Conf. on Theoretical Computer Science, World Scientific, pp. 204-215, 1998

[10] M. Bernardo, R. Gorrieri, M. Roccetti, *"Formal Performance Modelling and Evaluation of an Adaptive Mechanism for Packetised Audio over the Internet"*, in Formal Aspects of Computing 10:313-337, 1999

[11] M. Bravetti, M. Bernardo, R. Gorrieri, *"A Note on the Congruence Proof for Recursion in Markovian Bisimulation Equivalence"*, in Proc. of 6th Int. Workshop on Process Algebras and Performance Modeling, C. Priami editor, pp. 153-164, 1998

[12] L. Cardelli, *"Real time agents"*, in Proc. of 9th Int. Colloquium on Automata, Languages and Programming, LNCS 140:94-106, Springer-Verlag, 1982

[13] F. Corradini, M. Pistore, *"Closed Interval Process Algebra versus Interval Process Algebra"*, in Acta Informatica 37, pp. 467-509, 2001

[14] P.R. D'Argenio, H. Hermanns, J.P. Katoen, *"On Generative Parallel Composition"*, in Proc. of 1st Int. Workshop on Probabilistic Methods in Verification (PROBMIV '98), in Electronic Notes in Theoretical Computer Science, Vol. 22, 2000

[15] R.J. van Glabbeek, S.A. Smolka, B. Steffen, *"Reactive, Generative and Stratified Models of Probabilistic Processes"*, in Information and Computation 121:59-80, 1995

[16] J. Groote, *"Transition System Specifications with Negative Premises"*, in Theoretical Computer Science, 118(2):263-299, 1993

[17] M. Hennessy, T. Regan, *"A Process Algebra for Timed Systems"*, in Information and Computation, 117(2):221-239, 1995

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

[19] C.A.R. Hoare, *"Communicating Sequential Processes"*, Prentice Hall, 1985

[20] R.A. Howard, *"Dynamic Probabilistic Systems"*, John Wiley & Sons, 1971

[21] C.C. Jou, S.A. Smolka, *"Equivalences, Congruences, and Complete Axiomatizations for Probabilistic Processes"*, in Proc. of 1st Int. Conf. on Concurrency Theory, LNCS 458:367-383, Springer-Verlag, 1990

[22] L. Kleinrock, *"Queueing Systems"*, John Wiley & Sons, 1975

[23] K.G. Larsen, A. Skou, *"Bisimulation through Probabilistic Testing"*, in Information and Computation 94:1-28, 1991

[24] K.G. Larsen, A. Skou, *"Compositional Verification of Probabilistic Processes"*, in Proc. of 3rd Int. Conf. on Concurrency Theory, LNCS 630:456-471, Springer-Verlag, 1992

[25] N.A. Lynch, M.R. Tuttle, *"Hierarchical Correctness Proofs for Distributed Algorithms"*, in Proc. 6th ACM Symp. on Principles of Distributed Computing, pp. 137-151, 1987

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

[27] F. Moller, C. Tofts, *"A Temporal Calculus of Communicating Systems"*, in Proc. of 1st Int. Conf. on Concurrency Theory, LNCS 458:401-415, Springer-Verlag, 1990

[28] R. Segala, *"Modeling and Verification of Randomized Distributed Real-Time Systems"*, Ph.D. Thesis, MIT, Boston (MA), 1995

[29] A.S. Tanenbaum, *"Computer Networks"*, Prentice Hall, 1996

[30] H.M. Taylor, S. Karlin, *"An Introduction to Stochastic Modeling"*, 3rd ed., Academic Press, 1998

[31] C. Tofts, *"Processes with Probabilities, Priority and Time"*, in Formal Aspects of Computing 6:536-564, 1994

[32] S.H. Wu, S.A. Smolka, E.W. Stark, *"Composition and Behaviors of Probabilistic I/O Automata"*, in Theoretical Computer Science 176:1-38, 1997