Expressing Processes with Different Action Durations through Probabilities

Mario Bravetti and Alessandro Aldini

Università di Bologna, Dipartimento di Scienze dell'Informazione,
Mura Anteo Zamboni 7, 40127 Bologna, Italy,
{bravetti,aldini}@cs.unibo.it

Abstract. We consider a discrete time process algebra capable of (i) modeling processes with different probabilistic advancing speeds (mean number of actions executed per time unit), and (ii) expressing probabilistic external/internal choices and multiway synchronization. We show that, when evaluating steady state based performance measures expressed by associating rewards with actions, such a probabilistic approach provides an exact solution even if advancing speeds are considered not to be probabilistic (i.e. actions of different processes have a different exact duration), without incurring in the state space explosion problem which arises with an intuitive application of a standard synchronous approach. We then 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.

1 Introduction

The modeling experience in the specification of probabilistic concurrent systems (see, e.g., [4] and the references therein) has revealed the importance of using languages expressing advancing speed of processes, probabilistic internal/external choices, and multi-way synchronization for representing the behavior of real systems. In [1] we have considered a probabilistic calculus that combines, in a natural way, these mechanisms. In particular, such a calculus adopts a mixture of the generative and reactive approaches [5] by considering an *asymmetric* form of synchronization where a process which behaves generatively may only synchronize with processes which behave reactively. The integration of the generative and reactive approaches is naturally obtained, similarly as in [12], by designating some actions, called *generative* actions, as predominant over the other ones, called *reactive* actions (denoted by a subscript "$*$"), and by imposing that generative actions can synchronize with reactive actions only. In particular, the parallel operator that we considered in [1] is similar to the CSP [6] operator $P \,\|_S\, Q$, where processes P and Q are required to synchronize over actions of 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 a generative action a and a reactive action a_* is a generative action a, while the result of the synchronization of two reactive actions a_* is again a reactive action a_*.

As a consequence, an n-way synchronization is composed of all reactive actions except at most one generative action: the choice of a generative action a determines the action type to be performed and the other processes internally react by *independently* choosing one of their reactive actions a_*. Similarly as in [3], our parallel operator is parameterized with a probability p, determining the process performing the next move in a term $P \,\|_S^p\, Q$: we choose P with probability p while we choose Q with probability $1-p$. We call p and $1-p$ the *probabilistic advancing speeds* of P and Q, respectively. As an example, let us consider a system composed of two sequential processes whose behavior is described by a term like[1]: $(a +^p b) \,\|^q (c +^r d)$. According to [3], 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 if the righthand process wins we locally choose between c and d according to probabilities r and $1-r$. Moreover, our approach integrates the generative-reactive approach inspired by [12] with the approach to probabilistic process choice inspired by [3] 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 possibly employing probabilistic choice among processes. Since 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, *fully specified* systems always give rise to probabilistic transition systems which are purely generative. Fully specified systems are therefore fully probabilistic systems (systems not including non-deterministic choices [9]), from which a Markov Chain can be trivially derived by discarding actions from transition labels. As a consequence, they can be easily analyzed to derive performance measures.

In this paper we start by showing that, if we interpret probabilistic transition systems produced with the calculus of [1] in a *discrete time setting*, where each transition takes a discrete time step to be executed, actions of processes are actually executed with the advancing speed expressed as parameters of the parallel operators.

Differently from existing discrete time process algebras, where parallel processes are executed in synchronous locksteps (see, e.g., [7, 11, 8]), the parallel 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. As we now show, $P \,\|_S^p\, Q$ represents a system where the mean action frequency (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 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; process Q, representing the

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

priority mail service, repeatedly executes actions b expressing the delivery of a letter via priority mail. Supposed that we take minutes to be the time unit on which the post office specification is based, in $P \,||^{0.2}_{\emptyset}\, Q$ 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 in the mean to be executed, while the actions of Q take 1 minute and 15 seconds in the mean to be executed.

To be more precise the execution of a system $P \,||^{p}_{S}\, Q$ 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 which 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, the mean number of heads that have been extracted (steps P has made) is $n \cdot p$ while the mean number of tails that have been extracted (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 semantics of $P \,||^{p}_{S}\, Q$ 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 $P \,||^{p}_{S}\, Q$ 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 semantics of $P \,||^{p}_{S}\, Q$ as originated by a *single-processor machine* executing both processes (P and Q) via a probabilistic scheduler. In this view choices in the global states of a system $P \,||^{p}_{S}\, Q$ 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 memoryless distributed sojourn times as, e.g., in the continuous time model of [12], but we simply execute a system transition every discrete time unit. In the execution of $P \,||^{p}_{S}\, Q$ sometimes we perform a discrete move of P and sometimes we perform a discrete move of Q and what matters, from the performance behavior standpoint, is the frequency with which the actions of a given process are executed: 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 is the level of abstraction necessary in order to evaluate performance properties in discrete time. This because, since the states of a Discrete Time

Markov Chain (DTMC) are not endowed with different sojourn times (like e.g. in Continuous Time Markov Chains), the evaluation of performance measures in a DTMC is entirely based on the execution frequency of transitions. As we will see, the operator $P \parallel_S^p Q$ can be used both for modeling the execution of P and Q in a single processor machine by means of a probabilistic scheduler (strict interpretation of the semantics of $P \parallel_S^p Q$), and for expressing, as in the post office example, the actual concurrent execution of processes P and Q whose actions have a (possibly) different mean duration (interpretation of the semantics of $P \parallel_S^p Q$ as an abstract description of concurrency), depending on how we calculate the duration of the time unit to be considered for the composed system.

In this paper we also consider the problem of modeling discrete time systems where the different advancing speeds at which concurrent processes proceed are not probabilistic. This means that 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. Through a standard approach based on a synchronous parallel composition, where parallel processes are executed in synchronous locksteps (see, e.g., [7, 11, 8]), processes executing actions with different exact durations 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 subactions and we take *div* to be the action duration 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. Our approach constitutes an approximated solution to this problem, in that action frequencies of processes are probabilistic instead of being exact, but actions are not split. Nevertheless we show 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 correct performance measures when the system reaches a steady behavior. 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 [4]), our approach avoids action splitting, hence the state space explosion problem, while preserving the possibility of exactly analysing concurrent processes with exact advancing speeds.

Finally we present a case study which shows all the main features of our calculus. More precisely, we model and analyze an algebraic specification of a router implementing a probabilistic multi-path routing mechanism. This case study shows 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 half a microsecond 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 on how they are specified, their advancing speed 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 which is easily scalable to an arbitrary size of the routing table.

The paper is organized as follows. In Sect. 2 we briefly recall the syntax of the calculus of [1]. In Sect. 3 we show how to calculate the time unit to be considered for the composed system depending on the interpretation (processes executed by a single processor machine or actually concurrent processes) given to the parallel operator. In Sect. 4 we show that through our approach we can model systems where actions of different processes have a different exact duration. Finally, in Sect. 5 we present the case study on multi-path routing.

2 Syntax of the Calculus

In this section we briefly recall the generative-reactive calculus introduced in [1]. 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 π, $\pi', \ldots$. The syntax of the generative-reactive process algebra is defined as follows. Let $Const$ be a set of constants, ranged over by $A, B, \ldots$.

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

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

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

We use the following abbreviations to denote the relabeling of several action types. 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 $a_i \neq \tau$, 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 $a_1, \ldots, a_n$ into the visible actions $b_1, \ldots, b_n$, respectively. For the sake of simplicity, we omit probabilistic parameters in the operators of our calculus whenever they are not meaningful. For a presentation of the semantics of the calculus we refer to [1, 2].

3 Expressing Processes with Different Action Durations

In this section we show how our probabilistic parallel composition operator $P \parallel^p_S Q$ can be used to express the concurrent execution of processes P and Q whose actions have a different duration. If we strictly follow this single-processor interpretation of the semantics of $P \parallel^p_S Q$, 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 $P \parallel^p_S Q$ also take time u to be executed, i.e. u is also the time unit of $P \parallel^p_S Q$. 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. 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 $1/f_P$ in the mean 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 $P \,\|^p_\emptyset\, Q$ 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.

4 Expressing 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, in the case of non-blocking processes this does not happen when the system reaches a limiting steady behavior.

As already explained in Sect. 1, executing a process P in parallel with another process which proceeds with a different action frequency, could be done through a standard approach based on a synchronous parallel composition by adequately scaling the time unit on which the specification of P is based, i.e. by splitting each action of P in a certain number n of subactions (see [2] for the details).

Example 1. Let us consider a communication system composed of a processing unit receiving messages from an incoming channel and, after some internal computation, sending 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 $\mathbf{P}$ 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, where 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. 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. 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 that we consider for the time scaled system is unchanged with respect to the reward structure considered for the original system. 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 (see [2]), we obtain $m_t/10$, i.e. one tenth of the throughput m_t 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 where we associate 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. 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 scaling $\mathbf{P}$, we have to evaluate the reward structure to be associated with the time scaled system. Differently from the case of the message throughput, 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. By calculating the utilization of the time scaled system (see [2]), we obtain the same utilization m_u 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. $\square$

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 $P \,\|^p_\emptyset\, Idle$, where $Idle \stackrel{\Delta}{=} idle.Idle$ and $p = 1/n$.

Theorem 1. *The steady state based performance measures of $P \,\|^p_\emptyset\, Idle$ expressible by attaching rewards to the generative actions of P are axactly as those derived by executing the generative actions of P with an exact frequency p.* [2] $\square$

[2] In this theorem and in the following Theorem 2 we consider performance measures of periodic DTMCs to be evaluated from their time averaged steady state probabilities. The proof of both theorems can be found in [2].

Example 2. Let us consider the communication system P of Example 1, where the time unit is taken to be one second and $\mathbf{P}$ is the DTMC underlying P. Now we see how to evaluate the two performance measures of Example 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 its underlying DTMC $\mathbf{P}'$. Both in the case of the system throughput and in the case of the processing unit utilization (by using the same reward structure for the time scaled system as that considered in Example 1), we obtain the same performance measures as with the approach based on action splitting (see [2]). □

It is worth noting that, with respect to the standard approach 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: from $1/u_P = p/u$ we derive $u = u_P \cdot p$ where u represents action duration in the behavior of $P \,\|_{\emptyset}^{p}\, \mathit{Idle}$.

Finally, we have that with our probabilistic parallel operator we obtain a correct time unit scaling at the steady state also for non-blocking processes which are part of a larger system. Hence we can express the parallel composition of processes specified with respect to different time units through a common time unit by considering a different scaling factor for the time unit of each process.

Theorem 2. *Supposed that both P and Q 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 axactly as those derived by executing the generative actions of P and Q with an exact frequency p and $1 - p$, respectively.* □

Therefore given two different time units u_P and u_Q representing action duration in the specifications of P and Q in isolation, respectively, then there exists a common time unit u and a probability p (determined from $f_P = 1/u_P$ and $f_Q = 1/u_Q$ as explained in Sect. 3) such that $P \,\|_S^p\, Q$ just expresses the concurrent execution of the two processes without affecting their advancing speed.

5 A Case Study: Multi-path Routing

In this section we present a case study showing how our approach provides a 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 [10], and we model and analyze an internetworking node (termed Interface Message Processor, IMP for short), whose arriving packets have several possible destinations with several possible ways to reach a destination.

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 algorithm used to determine an optimal 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 and substantially provides better throughput and reliability.

5.1 Algebraic Specification of the Multi-path Router

The overall model of our IMP (term *Multipath*) is shown in Table 1 in the 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). 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: (i) 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. (ii) Term *Router* represents a process whose time unit is half a microsecond, i.e. 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 via a probabilistic scheduler. (iii) Term *Channels* is composed of four concurrent processes modeling the possible 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 time unit for the channel a_1 is 10 milliseconds, i.e. it can send out 100 packets per second, while the time unit for the channel a_2 is 2.5 milliseconds (400 packets per second). The time units for the channels b_1 and b_2 are 5 milliseconds (200 packets per second) and 4 milliseconds (250 packets per second), respectively.

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. 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. More precisely, 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. Such a global time unit u determines the action frequencies to be considered as parameters of the parallel operators used to describe the concurrent ex-

Table 1. Multi-path Routing Model

$$
\begin{array}{rl}
\mathit{Multipath} & \stackrel{\Delta}{=} \mathit{Arrivals} \parallel^{p_1}_{S} (\mathit{Router} \parallel^{p_2}_{C} \mathit{Channels}) \\
S & = \{\mathit{receivea},\ \mathit{receiveb}\} \\
C & = \{\mathit{avail\_cha}_1,\ \mathit{avail\_cha}_2,\ \mathit{avail\_chb}_1, \\
 & \quad \mathit{avail\_chb}_2,\ \mathit{transma}_1,\ \mathit{transma}_2,\ \mathit{transmb}_1,\ \mathit{transmb}_2\} \\
\mathit{Arrivals} & \stackrel{\Delta}{=} \mathit{Arrivala} \parallel^{\frac{1}{2}} \mathit{Arrivalb} \\
\mathit{Channels} & \stackrel{\Delta}{=} (\mathit{Channel}[\varphi'_1] \parallel^{pa} \mathit{Channel}[\varphi'_2]) \parallel^{v} (\mathit{Channel}[\varphi''_1] \parallel^{pb} \mathit{Channel}[\varphi''_2]) \\
\mathit{Router} & \stackrel{\Delta}{=} (\mathit{Queues} \parallel_{S_1 \cup I} \mathit{Switch}) \parallel_{I} \mathit{Idle} \\
S_1 & = \{\mathit{accepta},\ \mathit{acceptb}\} \qquad I = \{\mathit{idle}\} \\
\mathit{Queues} & \stackrel{\Delta}{=} \mathit{Queue}[\varphi'] \parallel_{I} \mathit{Queue}[\varphi''] \\
\mathit{Switch} & \stackrel{\Delta}{=} (\mathit{Manager}[\varphi'] \parallel_{S'_2} (\mathit{Routing}[\varphi'][\varphi'_1] \parallel^{qa}_{B'} \mathit{Routing}[\varphi'][\varphi'_2])\,) \parallel^{p}_{I} \\
 & \quad (\mathit{Manager}[\varphi''] \parallel_{S''_2} (\mathit{Routing}[\varphi''][\varphi''_1] \parallel^{qb}_{B''} \mathit{Routing}[\varphi''][\varphi''_2])\,) \\
S'_2 & = \{\mathit{senda},\ \mathit{busya}\} \qquad S''_2 = \{\mathit{sendb},\ \mathit{busyb}\} \\
B' & = \{\mathit{busya}\} \qquad B'' = \{\mathit{busyb}\} \\
\varphi' & = \langle(\mathit{receive},\ \mathit{receivea}),\ (\mathit{accept},\ \mathit{accepta}),\ (\mathit{send},\ \mathit{senda}),\ (\mathit{busy},\ \mathit{busya})\rangle \\
\varphi'' & = \langle(\mathit{receive},\ \mathit{receiveb}),\ (\mathit{accept},\ \mathit{acceptb}),\ (\mathit{send},\ \mathit{sendb}),\ (\mathit{busy},\ \mathit{busyb})\rangle \\
\varphi'_i & = \langle(\mathit{transm},\ \mathit{transma}_i),\ (\mathit{avail\_ch},\ \mathit{avail\_cha}_i)\rangle\ i \in \{1,2\} \\
\varphi''_i & = \langle(\mathit{transm},\ \mathit{transmb}_i),\ (\mathit{avail\_ch},\ \mathit{avail\_chb}_i)\rangle\ i \in \{1,2\} \\
\mathit{Arrivala} & \stackrel{\Delta}{=} \mathit{receivea}.\mathit{Arrivala} +^{ra} \mathit{wait}.\mathit{Arrivala} \\
\mathit{Arrivalb} & \stackrel{\Delta}{=} \mathit{receiveb}.\mathit{Arrivalb} +^{rb} \mathit{wait}.\mathit{Arrivalb} \\
\mathit{Queue} & \stackrel{\Delta}{=} \mathit{receive}_*.\mathit{Queue}' + \mathit{idle}_*.\mathit{Queue} \\
\mathit{Queue}' & \stackrel{\Delta}{=} \mathit{receive}_*.\mathit{accept}_*.\mathit{Queue}' + \mathit{accept}_*.\mathit{Queue} \\
\mathit{Manager} & \stackrel{\Delta}{=} \mathit{accept}.\mathit{Manager}' + \mathit{idle}_*.\mathit{Manager} \\
\mathit{Manager}' & \stackrel{\Delta}{=} \mathit{send}.\mathit{Manager} + \mathit{busy}.\mathit{Manager}' \\
\mathit{Routing} & \stackrel{\Delta}{=} \mathit{send}_*.\mathit{Routing}' + \mathit{avail\_ch}_*.\mathit{Routing} \\
\mathit{Routing}' & \stackrel{\Delta}{=} \mathit{transm}_*.\mathit{Routing} + \mathit{busy}_*.\mathit{Routing}' \\
\mathit{Idle} & \stackrel{\Delta}{=} \mathit{idle}.\mathit{Idle} \\
\mathit{Channel} & \stackrel{\Delta}{=} \mathit{avail\_ch}.\mathit{Channel} + \mathit{transm}.\mathit{Channel}
\end{array}
$$

ecution of system components in Table 1. In particular, parameter p_1 representing the advancing speed of term *Arrivals* in $\mathit{Arrivals} \parallel^{p_1}_{S} (\mathit{Router} \parallel^{p_2}_{C} \mathit{Channels})$ 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 $p_1 = 2000/2002950 \approx 0.000999$. Similarly, parameter p_2 representing the advancing speed of term *Router* in $\mathit{Router} \parallel^{p_2}_{C} \mathit{Channels}$ is given by the ratio of the action frequency of term *Router* over the global action frequency of term $\mathit{Router} \parallel^{p_2}_{C} \mathit{Channels}$, 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 $(\mathit{Channel}[\varphi'_1] \parallel^{pa} \mathit{Channel}[\varphi'_2]) \parallel^{v} (\mathit{Channel}[\varphi''_1] \parallel^{pb} \mathit{Channel}[\varphi''_2])$ we take: pa to be $100/(100 + 400) = 0.2$; pb to be $200/(200 + 250) \approx 0.444444$; and v to be

$500/(500+450) \approx 0.526316$. As an example of the calculated action frequencies, 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: (i) 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). (ii) Process *Router* is the core of the IMP and is composed of a term *Queues* collecting the arriving packets, a term *Switch* delivering 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*). 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. (*iii*) The four *Channel* processes model the outgoing channels a_1, a_2, b_1 and b_2. 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, we have that the transition system underlying the algebraic specification of Table 1 is composed of 576 states and 4768 transitions only. This is a crucial result, because if we model the same system by resorting to a classical 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 while 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 terms *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 [2] we have scaled the system of the router specification to four possible channels for each destination, by simply adding several instances of *Routing* and *Channel* terms, and by appropriately adjusting the parameters of parallel composition operators.

5.2 Performance Analysis

In order to derive performance measures from the multi-path router specification, we resorted to the software tool TwoTowers [4], that has been recently extended to support the generative-reactive approach presented in this paper. Such a tool also implements an algebraic reward based method to specify and derive performance measures. The results of our performance analysis are shown in Fig. 1-3. 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 analysis of the Markov Chain by 2002950 in order to obtain the results (expressed in seconds) shown in our tables [2]. 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 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, due to the time unit change we must multiply each reward by 1/0.998527 before analyzing the Markov Chain [2].

For each conducted analysis, we assumed that the incoming traffic for the destinations a and b follows the same Bernoulli distribution of parameter $r = ra = rb$. The figures show how the performance measures change when we vary r from 0.1 (sometimes 0.01) to 0.9. In this way we 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. 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 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. 1. As we can see in Fig. 1(B) the curve representing the total system throughput is characterized by a high slope in correspondence of a low workload and a

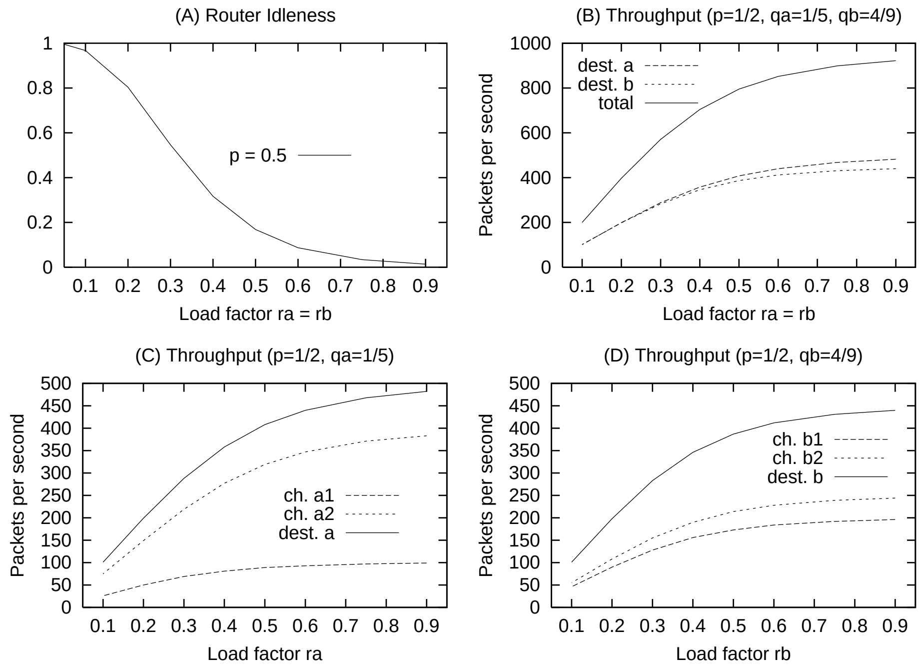

Fig. 1. Idleness and Throughput of the Multi-path Router

quite flat slope when the load factor increases over the 50%. This because, for packets with a given destination, the bandwidth associated with the 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 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 channels directed to a is 500 packets per second, while it is 450 packets per second for the channels directed to b. Fig. 1(C) and 1(D) 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 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. 2. In particular, Fig. 2(A) shows how the throughput of a_1 and a_2 varies when changing the value

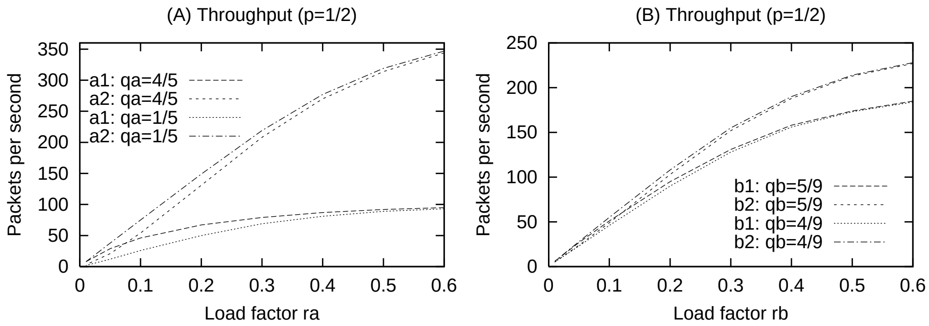

Fig. 2. Throughput obtained by varying parameters qa and qb

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%). Fig. 2(B) 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 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. 2, where the curves for the old and the new value of qb are almost overlapped for each value of rb.

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. 3 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 Fig. 3(A) is the same as that in Fig. 1(B)), 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 Fig. 3(A) we also report the curve obtained in the case $p = 0.5$). The comparison with the case

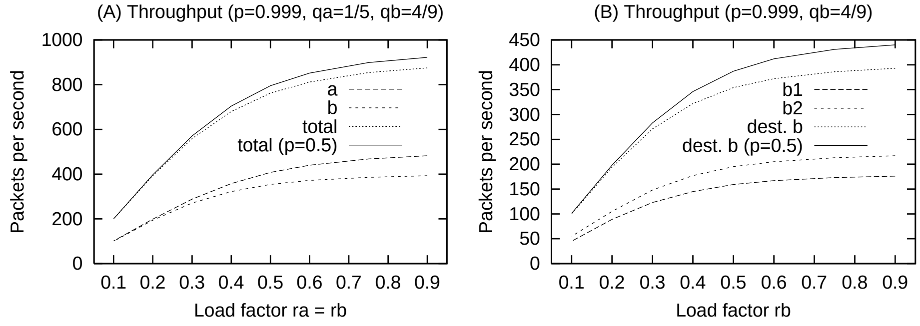

Fig. 3. Throughput obtained by varying parameter p

$p = 0.5$ is even more evident in Fig. 3(B), 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. 1(A) 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%.

References

1. A. Aldini, M. Bravetti, *"An Asynchronous Calculus for Generative-Reactive Probabilistic Systems"*, in Proc. of 8th Int. Workshop on Process Algebra and Performance Modeling, pp. 591-605, 2000
2. M. Bravetti, A. Aldini, *"An Asynchronous Calculus for Generative-Reactive Probabilistic Systems"*, Tech. Rep. UBLCS-2000-03, Univ. Bologna, Italy, 2000
3. J.C.M. Baeten, J.A. Bergstra, S.A. Smolka, *"Axiomatizing Probabilistic Processes: ACP with Generative Probabilities"*, in Inf. and Comp. 121:234-255, 1995
4. M. Bernardo, *"Theory and Application of Extended Markovian Process Algebra"*, Ph. D. Thesis, Univ. Bologna, Italy, 1999
5. R.J. van Glabbeek, S.A. Smolka, B. Steffen, *"Reactive, Generative and Stratified Models of Probabilistic Processes"*, in Inf. and Comp. 121:59-80, 1995
6. C.A.R. Hoare, *"Communicating Sequential Processes"*, Prentice Hall, 1985
7. 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, 1990
8. K.G. Larsen, A. Skou, *"Compositional Verification of Probabilistic Processes"*, in Proc. of 3rd Int. Conf. on Concurrency Theory, LNCS 630:456-471, 1992
9. R. Segala, *"Modeling and Verification of Randomized Distributed Real-Time Systems"*, Ph. D. Thesis, MIT, Boston (MA), 1995
10. A. S. Tanenbaum, *"Computer Networks"*, Prentice Hall, 1996
11. C. Tofts, *"Processes with Probabilities, Priority and Time"*, in Formal Aspects of Computing 6:536-564, 1994
12. 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