Tuning Temporal Features within the Stochastic π-Calculus

Loïc Paulevé, Morgan Magnin, and Olivier Roux

Abstract—The stochastic π-calculus is a formalism that has been used for modeling complex dynamical systems where the stochasticity and the delay of transitions are important features, such as in the case of biochemical reactions. Commonly, durations of transitions within stochastic π-calculus models follow an exponential law. The underlying dynamics of such models are expressed in terms of continuous-time Markov chains, which can then be efficiently simulated and model-checked. However, the exponential law comes with a huge variance, making it difficult to model systems with accurate temporal constraints. In this paper, a technique for tuning temporal features within the stochastic π-calculus is presented. This method relies on the introduction of a stochasticity absorption factor by replacing the exponential distribution with the Erlang distribution, which is a sum of exponential random variables. This paper presents a construction of the stochasticity absorption factor in the classical stochastic π-calculus with exponential rates. Tools for manipulating the stochasticity absorption factor and its link with timed intervals for firing transitions are also presented. Finally, the model-checking of such designed models is tackled by supporting the stochasticity absorption factor in a translation from the stochastic π-calculus to the probabilistic model checker PRISM.

Index Terms—Temporal parameters, π-calculus, model-checking, Markov processes, stochastic processes.

1 Introduction

By introducing temporal and stochastic aspects within models, designers aim to reproduce the behavior of real systems more closely. This complexity of dynamics models generally requires a compromise between a precise specification of temporal parameters and efficient analysis techniques.

The π-calculus [1] is a concurrent process algebra suitable for modeling independently defined entities communicating through mobile channels. This formalism is widely used for analyzing, for instance, communication protocols [2]. Among the different extensions of this calculus, the stochastic π-calculus [3] introduces stochastic and temporal features within π-calculus models. The stochastic π-calculus is notably used to model biological systems [4], [5], [6], [7]. Temporal and stochastic features of biochemical reactions are prominent for analyzing and predicting the behaviors of such complex systems.

In the stochastic π-calculus framework, time and stochasticity are injected by making the duration of transitions a random variable. Usually, the random variable follows an exponential distribution. The parameter of this distribution is called the *use rate* and, informally, defines the number of times the transition is used within a time unit. Underlying semantics of exponentially distributed stochastic π-calculus expressions can be expressed with continuous-time Markov chains (CTMCs). By exploiting the memoryless property of the exponential law, efficient simulation algorithms have been designed—such as the *Gillespie algorithm* [8] or algorithms implemented in *BioSpi* [4] and *SPiM* [9]. The model-checking of stochastic π-calculus has been recently proposed in [10], which offers a translation of the exponentially distributed π-Calculus to PRISM, a probabilistic symbolic model checker [11].

Despite use rates bringing temporal features to models, the high variance of the exponential distribution forbids a precise modeling of temporal constraints. As an example, one may wonder how to model in stochastic π-calculus a transition taking place only within a given time interval, as is usually done with timed automata [12] or time Petri nets [13]. This kind of temporal specification is commonly used, easy to manipulate, and necessary to model systems where temporal features determine the dynamics.

In this paper, we present a technique to tune temporal features within stochastic π-calculus models and we provide a method to analyze such models using PRISM. This tuning permits to model both stochastic and temporal aspects of a system in a less dependent manner than in the classical stochastic π-calculus. Such modeling is enabled by attaching to actions, in addition to the use rate, a so-called *stochasticity absorption factor*. The higher the stochasticity absorption factor, the lower the temporal variance around the mean imposed by the use rate. The stochasticity absorption of transitions is obtained by replacing the exponential distribution by the Erlang distribution, which is the distribution of the sum of exponential random variables.

This paper presents three main and original contributions:

- *Translation of the stochastic π-calculus with Erlang distributions into the stochastic π-calculus with exponential distributions.* This translation allows us to simulate and analyze stochastic π-calculus models with Erlang distributions by using the numerous classical tools that apply on the Markovian stochastic π-calculus. The construction is purely syntactic, and does not rely on explicit construction of CTMCs.

- *The authors are with IRCCyN, UMR CNRS 6597, École Centrale de Nantes, France.*
E-mail: {loic.pauleve, morgan.magnin, olivier.roux}@irccyn.ec-nantes.fr.

Manuscript received 28 Aug. 2009; revised 3 June 2010; accepted 19 Sept. 2010; published online 19 Oct. 2010.
Recommended for acceptance by H. Schmidt.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-2009-08-0209.
Digital Object Identifier no. 10.1109/TSE.2010.95.

- *Estimators for the parameters of the Erlang distribution corresponding to a given time interval.* In this way, similarly to timed automata or time Petri nets, a transition can be parameterized by specifying a confidence time interval of firing, referred to as the *firing interval*. The transition only occurs within the specified time interval at a given confidence coefficient.
- *Model-checking of the stochastic π-calculus with Erlang distributions using PRISM*, by adapting the prior translation of the Markovian stochastic π-calculus into PRISM of Norman et al. [10] to support the stochasticity absorption factor. This brings the first straightforward model-checking of the stochastic π-calculus with Erlang distributions. Moreover, the proposed construction of the stochasticity absorption factor in PRISM can be applied to more general models than those resulting from the translation from the stochastic π-calculus.

The complexity of the translation of the stochastic π-calculus with Erlang distributions into the stochastic π-calculus with exponential distributions is done linearly in the size of the model. In the same manner, the addition of the stochasticity absorption factor within PRISM models resulting from a translation of the stochastic π-calculus is also done linearly in the size of the model. Whereas the simulation does not suffer from the addition of stochasticity absorptions, we observe that the model-checking of such models suffers from a state space explosion for large stochasticity absorption factors.

The applicability of the overall method is illustrated by a case study: the modeling of the biological segmentation of metazoans [14]. This biological case study relies on quite precise temporal specifications that are difficult to incorporate within a pure Markovian stochastic π-calculus. We propose here a stochastic π-calculus model of this biological system, and we study the influence of the stochasticity absorption factor in the reproduction of particular behaviors of this model.

This paper is structured as follows: Section 2 formally presents a variant of the stochastic π-calculus. Section 3 introduces the stochasticity absorption factor through the Erlang distribution. A construction of the Erlang distributed from the exponentially distributed stochastic π-calculus is provided. Section 4 establishes the parallel between the firing interval of a transition, its rate, and its stochasticity absorption factor. Section 5 adapts the translation from the standard stochastic π-calculus into PRISM to support the stochasticity absorption factor. The applicability of the overall approach is discussed and illustrated in Section 6. Finally, Sections 7 and 8 discuss related work and the contributions of this paper.

2 The Stochastic π-Calculus

The π-calculus is a concurrent process algebra where two processes communicate using a shared channel [15]. For establishing a communication, a sender process outputs on a channel and a receiver process inputs on the same channel. The sender process may output *values* on the channel to transmit data to the receiver, where a value can also be a channel. In this way, π-calculus allows modeling the mobility of communication channels between concurrent processes.

The stochastic extension of the π-calculus affects to each *action* (channel input/output or internal action) a probabilistic distribution for determining the delay until it is effective [3], [16]. Usually, the exponential distribution is preferred. The parameter of this distribution is referred to as the *use rate* for the action, and informally, gives the number of times such an action is to be fired within one time unit.

The syntax for the stochastic π-calculus used in this paper is presented in Definition 1. It is close to the definition of the stochastic π-calculus used in [17].

Definition 1 (Stochastic π-Calculus). *Using C, C_i, P, Q to range over terms, A to range over definitions of race conditions, t to range over internal action identifiers, π to range over actions, a, y to range over channels, and $m, z_1, \ldots, z_n$ to range over values:*

$$
\begin{aligned}
A(z_1,\ldots,z_n) &::= \sum_{i\in I} C_i \qquad \textit{with } \mathrm{fn}(C_i) \subseteq \{z_1,\ldots,z_n\}\\
C, C_i &::= \nu a C \mid [cond]\ C \mid \pi.P\\
\pi &::= \tau_t \mid am \mid \bar{a}m\\
P, Q &::= P \mid Q \mid A(z_1,\ldots,z_n) \mid\\
&\quad\ \nu x P \mid [cond]\ P \mid \mathbf{0},
\end{aligned}
$$

where I is a finite index set and $cond$ is a Boolean expression on values. $\mathrm{fn(C)}$ *is the set of channels that are free in C.*

$A()$ is abbreviated as A and $z_1, \ldots, z_n$ is abbreviated as $\tilde{z}$; $C_1 + C_2$ stands for $\sum_{i\in\{1,2\}} C_i$; am (respectively, $\bar{a}m$) is abbreviated as a (respectively, $\bar{a}$) when m is not used in the following term.

The actions π of a process are either internal action (τ_t), input of m on channel a (am), or output of m on channel a ($\bar{a}m$); m is a value: either a channel y, an extendable list of values, or a tuple of values $\tilde{m}$. As in [17], when outputting, m can also be a fresh channel νy, i.e., a bound output. After performing such an action π, the process can evolve as P, written $\pi.P$. The term $\mathbf{0}$ denotes the null process, $P|Q$ the parallel composition of processes P and Q, $[cond]P$ the process P enabled only if $cond$ is satisfied (generally, $cond$ is a conjunction of tests upon values), and $\nu a P$ is the restriction of a fresh channel a to the process P. $\sum_{i\in I} C_i$ is a race condition between processes C_i: Only the first fired C_i is considered. Each race condition has a definition of the form $A(\tilde{z}) = \sum_{i\in I} C_i$, where $\tilde{z}$ are the parameters of the process A. I is assumed to be finite and may depend on values in $\tilde{z}$. If I is empty, the sum is equivalent to the null process.

A channel a is *bound* to a process P if there exists a preceding restriction νa within the expression of P. Otherwise, the channel a is free (or unbound). The set of bound channels in a process P is noted as $\mathrm{bn(P)}$ and the set of free channels as $\mathrm{fn(P)}$. $\mathrm{n(P)} = \mathrm{fn(P)} \cup \mathrm{bn(P)}$ is the set of channels present in the term P.

The first action performed by a process C involved in a race condition is denoted by $\Pi(C)$ and is defined as below. Hereafter, it is assumed that if the first action performed by C is an input or output on a channel, this channel is free in C.

$$
\begin{aligned}
&\Pi(\tau_t.P) = \tau_t \quad \Pi(\nu x C) = \Pi(C) \text{ if } \Pi(C) \notin \{x, \bar{x}\}\\
&\Pi(ay.P) = a \quad \Pi([cond]C) = \Pi(C)\\
&\Pi(\bar{a}y.P) = \bar{a}.
\end{aligned}
$$

$$\text{TAU}\ \frac{}{\tau_t.P \xrightarrow{\tau_t} P} \qquad \text{SUM}\ \frac{P_j \xrightarrow{\pi} P'_j}{(\sum_{i\in I} P_i) \xrightarrow{\pi} P'_j} j \in I \qquad \text{PRE}_{\text{IN}}\ \frac{}{ay.P \xrightarrow{ay} P} \qquad \text{PRE}_{\text{OUT}}\ \frac{}{\overline{a}y.P \xrightarrow{\overline{a}y} P}$$

$$\text{PRE}_{\text{BIND}}\ \frac{P \xrightarrow{\overline{a}y} P'}{\nu y P \xrightarrow{\overline{a}\nu y} P'} a \neq y \qquad \text{COM}\ \frac{P \xrightarrow{ay} P'\ \ Q \xrightarrow{\overline{a}z} Q'}{P|Q \xrightarrow{\dot{a}} P'\{z/y\}|Q'} \qquad \text{BCOM}\ \frac{P \xrightarrow{ay} P'\ \ Q \xrightarrow{\overline{a}\nu z} Q'}{P|Q \xrightarrow{\dot{a}} \nu z(P'\{z/y\}|Q')} z \notin n(P)$$

$$\text{RES}\ \frac{P \xrightarrow{\pi} P'}{\nu a P \xrightarrow{\pi} \nu a P'} a \notin n(\pi) \qquad \text{MATCH}\ \frac{P \xrightarrow{\pi} P'}{[cond]P \xrightarrow{\pi} P'} cond \qquad \text{PAR}\ \frac{P \xrightarrow{\pi} P'}{P|Q \xrightarrow{\pi} P'|Q} bn(\pi) \cap \text{fn}(\text{Q}) = \emptyset$$

Fig. 1. Operational semantics for the stochastic π-calculus (Definition 1).

Fig. 1 depicts the operational semantics of the stochastic π-calculus, which is close to [17]. Transitions are labeled by the action to be performed, τ_t denotes an internal transition identified by t, $\dot{a}$ denotes a communication on channel a, that is the synchronization of one output with one input on channel a. Free and bound channels on actions π are defined in Table 1.

We denote by $S\pi_e$ the stochastic π-calculus having the duration of each action on a (internal action τ_a or input/output on channel a) being an exponential random variable at use rate $r_a \in \mathbb{R}+^*$. The probability that an action at use rate r is fired within a delay t is $1 - \exp(-r.t)$. Its average duration is r^{-1} time units with a variance of r^{-2}. Given x actions having, respectively, use rates $r_1, \ldots, r_x$, the probability that the yth action is fired is $\frac{r_y}{r_1+\cdots+r_x}$. We also consider actions having an infinite rate $r_a = \infty$, i.e., which are instantaneous. Such actions are always fired first. If two instantaneous actions are possible, the choice of the one to fire is nondeterministic.

3 Stochasticity Absorption

The exponential law brings a strong binding between the average duration and the temporal variance of the duration for a transition. For instance, the higher the average duration (or equivalently, the lower the use rate), the higher the variance. The *stochasticity absorption factor* aims at coping with this strong binding between temporal and stochastic features.

Instead of having the duration of an action following one exponential random variable at rate r, we propose to have the duration of an action following the sum of sa exponential random variables at rate $r.sa$. This results in an unchanged average duration, but a variance divided by sa. In this way, sa stands for the stochasticity absorption factor. The obtained probabilistic distribution is known as the *Erlang distribution*. Therefore, the tuning of the temporal features within the stochastic π-calculus is achieved by attaching to each action an Erlang distribution at a fixed rate and stochasticity absorption factor. We denote by $S\pi_{Er}$ the stochastic π-calculus having the duration of transitions following an Erlang distribution.

This section starts by presenting the Erlang distribution and functions to compute some standard probabilities. The translation of $S\pi_{Er}$ into $S\pi_e$ is then presented, demonstrating that $S\pi_{Er}$ processes can be simulated using standard algorithms based on the exponential law for firing actions. Finally, the $S\pi_{Er}$ calculus is illustrated by a simple example.

TABLE 1
Free and Bound Channels for an Action π

π	description	$\text{fn}(\pi)$	$\text{bn}(\pi)$
τ_t	internal action	$\emptyset$	$\emptyset$
am	input of value m	$\{a, m\}$	$\emptyset$
$\bar{a}m$	output of value m	$\{a, m\}$	$\emptyset$
$\bar{a}\nu y$	bound output of channel y	$\{a\}$	$\{y\}$
$\dot{a}$	communication	$\{a\}$	$\emptyset$

3.1 The Erlang Distribution

The Erlang distribution is usually defined by two parameters: the shape $k \in \mathbb{N}^*$ and the rate $\lambda \in \mathbb{R}_+^*$. The Erlang distribution is then the distribution of the sum of k independent exponentially distributed random variables at use rate λ. The Erlang distribution is a particular case of the *gamma distribution* where the shape parameter may be any positive real.

For the sake of consistency, we refer to the Erlang distribution as the distribution of the sum of sa exponential random variables at use rate $r.sa$, where sa is the stochasticity absorption factor and r the use rate of the nonabsorbed exponential variable (i.e., when $sa = 1$). Equivalence between these two definitions is given by the relations $k = sa$ and $\lambda = r.sa$.

The probability density function (PDF) and cumulative distribution function (CDF) of an Erlang distribution at use rate r and stochasticity absorption factor sa are defined in (1) and (2). They match the classical Erlang distribution with shape sa and rate $r.sa$ [18]. PDF and CDF with different stochasticity absorption factors but a constant rate are plotted in Fig. 2.

$$f_{r,sa}(t) = \frac{(r.sa)^{sa} t^{sa-1} \exp(-r.sa.t)}{(sa-1)!}, \tag{1}$$

$$F_{r,sa}(t) = 1 - \exp(-r.sa.t) \sum_{n=0}^{sa-1} \frac{(r.sa.t)^n}{n!}. \tag{2}$$

Let an action have its duration following an Erlang distribution at use rate r and stochasticity absorption factor sa. The average duration of this action is r^{-1} with a variance of $r^{-2}sa^{-1}$. $F_{r,sa}(t)$ gives the probability of firing the action within a time t. Given x actions having, respectively, use rates $r_1, \ldots, r_x$, and stochasticity absorption factors $sa_1, \ldots, sa_x$, the probability that the yth action is fired is given by (3) [16]:

$$\int_0^\infty f_{r_y,sa_y}(t) \prod_{w \neq y} (1 - F_{r_w,sa_w}(t)) dt. \tag{3}$$

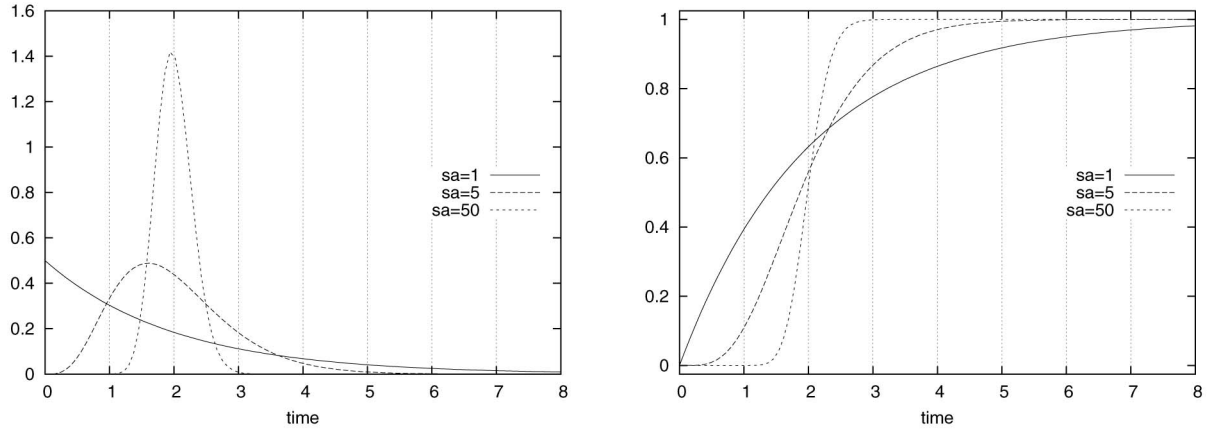

Fig. 2. Probability density function (left) and cumulative distribution function (right) of the Erlang distribution for different stochasticity absorption factors (sa) when rate is $\frac{1}{2}$.

3.2 Stochastic π-Calculus Construction

In this section, a construction of $S\pi_{Er}$ processes into $S\pi_e$ is proposed. This construction is purely syntactic: There is no explicit CTMCs building and its complexity is linear with the size of the $S\pi_{Er}$ expression. This mapping also allows the analysis of $S\pi_{Er}$ processes by classical tools (simulators, model-checkers, etc.) that assume exponential distributions for transitions.

The construction is done through a mapping operator $[\![.]\!]_e$, where . stands for an $S\pi_{Er}$ term. If P is an $S\pi_{Er}$ term, $[\![P]\!]_e$ is an $S\pi_e$ term showing the same behavior as P using only exponential distributions (this is to be more formally detailed in this section). The full mapping of $S\pi_{Er}$ terms into $S\pi_e$ is given by Definition 2. It is worthwhile to notice that this construction can be applied to any $S\pi_{Er}$ processes.

We sketch this transformation. Beforehand, to each couple of parameters (r_a, sa_a) of the $S\pi_{Er}$ process, a parameter $r'_a = r_a.sa_a$ is set for the resulting $S\pi_e$ process. It is assumed that $sa_a > 1$, i.e., the action does not follow an exponential distribution. The goal is then to execute sa_a times the exponential action on a (either an internal action τ_a or a communication on channel a) before applying it (i.e., moving to the next process). To achieve such a behavior, a counter is associated with the action on a in the process definition. If the counter equals sa_a, the next execution of the action on a is applied normally. If the counter is less than sa_a, then the execution of the action on a results in repeating the process, with this counter incremented by one. While this principle can be straightforwardly applied to internal actions (6), the handling of inputs and outputs is more tricky. Consider the following $S\pi_{Er}$ process A_1, present in the expression of A:

$$\begin{aligned} &A_1(a) ::= \bar{a}.\mathbf{0}, \quad A_2(a) ::= a.\mathbf{0}, \\ &A(a) ::= A_1(a)|A_2(a)|A_2(a). \end{aligned} \tag{4}$$

When counting the number of firings of the channel output $\bar{a}$, there must be a differentiation between the receiving instances of A_2. To achieve this differentiation, the $S\pi_e$ process A_1 first outputs a pair of new channels a', a'' on a. After $sa_a - 2$ outputs on the private channel a', A_1 finally outputs on a'', acting as the final action before evolving to the next process. Technically, the private channels and the counter of each instantiated communication absorption are stored in lists that are passed as arguments in the process definition (7). For the inputting process, no counting is necessary: The process first inputs the two private channels a' and a'', stores these channels in lists, and adds them to the race conditions. An input on a' means the stochasticity absorption is not complete, so the current process has to be reiterated. An input on a'' means the communication is fully absorbed; the next process is called. As for the channel output, the set of pairs of private channels has to be passed as arguments (8).

Basically, the construction replaces each action at use rate r and stochasticity absorption factor sa with sa actions following an exponential law at use rate $r.sa$. A communication over a channel a is not literally repeated sa_a times, but rather repeated sa_a times over channels a, a', and a'' together.

Definition 2 ($[[.]]_e$). *Let $[[.]]_e$ be the map from $S\pi_{Er}$ processes and definitions into $S\pi_e$ processes and definitions given by the rules below.*

Notations. *Given a set I of m indexes, $\tilde{\rho}$ stands for $c_1, \ldots, c_m, a'_1, \ldots, a'_m, a''_1, \ldots, a''_m$. $\forall 1 \leq i \leq m$, c_i is either a positive integer or a list of positive integers and a'_i and a''_i are lists of channels. $\#a'_i$ is the length of the list a'_i. The nth element of the list a'_i is denoted by a'_{i_n}. $\tilde{\rho}\{c_i + = 1\}$ stands for $\tilde{\rho}$ having c_i incremented by one. $\tilde{\rho}\{2 :: c_i, a' :: a'_i\}$ stands for $\tilde{\rho}$ having 2 (respectively, a') added to the list c_i (respectively, a'_i). $\bar{a}(a', a'')$ (respectively, $a(a', a'')$) stands for the outputting (respectively, inputting) on channel a of the couple of channels a', a''.*

$$[\![\mathbf{0}]\!]_e = \mathbf{0} \qquad [\![\nu x P]\!]_e = \nu x [\![P]\!]_e$$
$$[\![P|Q]\!]_e = [\![P]\!]_e | [\![Q]\!]_e \qquad [\![[cond]P]\!]_e = [cond][\![P]\!]_e.$$

- *Process definition:*

$$[\![A(\tilde{z}) ::= \sum_{i \in I} C_i]\!]_e = A(\tilde{z}, \tilde{\rho}) ::= \sum_{i \in I} [\![C_i]\!]_e, \tag{5}$$

  with $\tilde{\rho} \cap n(A(\tilde{z})) = \emptyset$.

- *Stochasticity absorption factor equals 1:*

$$[\![\tau_t.P]\!]_e = \tau_t.[\![P]\!]_e$$
$$[\![\bar{a}y.P]\!]_e = \bar{a}y.[\![P]\!]_e \qquad [\![ay.P]\!]_e = ay.[\![P]\!]_e.$$

  In the following, it is assumed that stochasticity absorption factors are greater than 1. i stands for the

$$\mathsf{TAU_{wait}}\ \frac{A(\tilde{z})\xrightarrow{\tau_t}_E B(\tilde{w})}{A_e(\tilde{z},\tilde{\rho})\xrightarrow{\tau_t}_e A_e(\tilde{z},\tilde{\rho}\{c_i+=1\})}c_i < sa_t \qquad \mathsf{TAU_{eff}}\ \frac{A(\tilde{z})\xrightarrow{\tau_t}_E B(\tilde{w})}{A_e(\tilde{z},\tilde{\rho})\xrightarrow{\tau_t}_e B_e(\tilde{w},\tilde{\varrho}_1)}c_i = sa_t$$

$$\mathsf{IN_{init}}\ \frac{A(\tilde{z})\xrightarrow{ay}_E B(\tilde{w})}{A_e(\tilde{z},\tilde{\rho})\xrightarrow{a(a',a'')}_e A_e(\tilde{z},\tilde{\rho}\{a'::a'_i,a''::a''_i\})} \qquad \mathsf{OUT_{init}}\ \frac{A(\tilde{z})\xrightarrow{\overline{a}y}_E B(\tilde{w})}{A_e(\tilde{z},\tilde{\rho})\xrightarrow{\overline{a}(a',a'')}_e A_e(\tilde{z},\tilde{\rho}\{2::c_i,a'::a'_i,a''::a''_i\})}$$

$$\mathsf{IN_{wait}}\ \frac{A(\tilde{z})\xrightarrow{ay}_E B(\tilde{w})}{A_e(\tilde{z},\tilde{\rho})\xrightarrow{a'}_e A_e(\tilde{z},\tilde{\rho})}\exists n, a'_{i_n} = a' \qquad \mathsf{OUT_{wait}}\ \frac{A(\tilde{z})\xrightarrow{\overline{a}y}_E B(\tilde{w})}{A_e(\tilde{z},\tilde{\rho})\xrightarrow{\overline{a'}}_e A_e(\tilde{z},\tilde{\rho}\{c_{i\,n}+=1\})}\exists n, a'_{i_n} = a', c_{i_n} < sa_a$$

$$\mathsf{IN_{eff}}\ \frac{A(\tilde{z})\xrightarrow{ay}_E B(\tilde{w})}{A_e(\tilde{z},\tilde{\rho})\xrightarrow{a''y}_e B_e(\tilde{w},\tilde{\varrho}_1)}\exists n, a''_{i\ n} = a'' \qquad \mathsf{OUT_{eff}}\ \frac{A(\tilde{z})\xrightarrow{\overline{a}y}_E B(\tilde{w})}{A_e(\tilde{z},\tilde{\rho})\xrightarrow{a''y}_e B_e(\tilde{w},\tilde{\varrho}_1)}\exists n, a''_{i\ n} = a'', c_{i_n} = sa_a$$

Fig. 3. Operational semantics derived from Definition 2 for internal actions and channels input/outputs. i is the index of the action π in the sum defined by the $S\pi_{Er}$ process A. $\tilde{\varrho}_1$ is computed as $\tilde{\rho}_1$ in (9).

index of the process in the race condition $A(\tilde{z}) = \sum_{i\in I} C_i$.

- *Internal action:*

$$[\![\tau_t.P]\!]_e = [c_i < sa_t]\tau_t.A(\tilde{z},\tilde{\rho}\{c_i+=1\}) + [c_i = sa_t]\tau_t.[\![P]\!]_e. \tag{6}$$

- *Channel output:*

$$\begin{aligned}[\![\overline{a}y.P]\!]_e &= \nu a'\nu a''\overline{a}(a',a'').A(\tilde{z},\tilde{\rho}\{2::c_i,a'::a'_i,a''::a''_i\}) \\ &+ \sum_{n=1}^{\#c_i}[c_{i_n} < sa_a]\overline{a'_{i_n}}.A(\tilde{z},\tilde{\rho}\{c_{i_n}+=1\}) \\ &+ [c_{i_n} = sa_a]\overline{a''_{i\ n}}y.[\![P]\!]_e\end{aligned} \tag{7}$$

with $r_{a'} = r_{a''} = r'_a$ *and* $\{a',a''\}\cap(fn(A)\cup fn(P)) = \emptyset$.

- *Channel input:*

$$[\![ay.P]\!]_e = a(a',a'').A(\tilde{z},\tilde{\rho}\{a'::a'_i,a''::a''_i\}) + \sum_{n=1}^{\#a'_i} a'_{i_n}.A(\tilde{z},\tilde{\rho}) + a''_{i\ n}y.[\![P]\!]_e \tag{8}$$

with $\{a',a''\}\cap(fn(A)\cup fn(P)) = \emptyset$.

- *Process call, all counters and lists are initialised to 0 and empty, respectively:*

$$[\![A(\tilde{z})]\!]_e = A(\tilde{z},\tilde{\rho}_1) \tag{9}$$

with, $\forall i\in I$, $a'_i = a''_i = \emptyset$, *and*

$$c_i = \begin{cases}1, & if\ \Pi(C_i) = \tau_t,\\ \emptyset, & else.\end{cases}$$

The mapping of $S\pi_{Er}$ processes defined in (4) using the $[\![.]\!]_e$ operator results in the following $S\pi_e$ processes:

$$\begin{aligned}A_1(a,c_1,a'_1,a''_1) &::= \nu a'\nu a''\overline{a}(a',a'').A_1(a,2::c_1,a'::a'_1,a''::a''_1) \\ &+ \sum_{n=1}^{\#c_1}[c_{1_n} < sa_a]\overline{a'_{1_n}}.A_1(a,c_1\{c_{1_n}+=1\},a'_1,a''_1) \\ &+ [c_{1_n} = sa_a]\overline{a''_{1_n}}y.\mathbf{0},\end{aligned}$$

$$\begin{aligned}A_2(a,c_1,a'_1,a''_1) &::= a(a',a'').A_2(a,c_1,a'::a'_1,a''::a''_1) \\ &+ \sum_{n=1}^{\#a'_1} a'_{1_n}.A_2(a,c_1,a'_1,a''_1) + a''_{1_n}.\mathbf{0},\end{aligned}$$

$$A(a) ::= A_1(a,\emptyset,\emptyset,\emptyset)|A_2(a,\emptyset,\emptyset,\emptyset)|A_2(a,\emptyset,\emptyset,\emptyset).$$

The major shortcoming of the presented encoding is the complete lack of identification between processes. At each iteration of the stochasticity absorption, the process involves a race between unguarded new initiation and guarded continuation and completion choices; thus there may be a redundant output of new channels a' and a'' when communicating with a same process. Also, no explicit garbage collection is done by the encoding, preventing the release of abandoned channels. A more efficient, yet more technical, implementation of the $S\pi_e$ construction of $S\pi_{Er}$ processes addressing these points is given in Supplemental Material A, which can be found in the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSE.2010.95.

3.3 Correctness of the Construction

This section establishes the necessary lemmas to prove that both qualitative and quantitative behaviors are preserved by the $S\pi_e$ construction of $S\pi_{Er}$ processes. Fig. 3 identifies the transition rules for a $S\pi_e$ process resulting from the construction defined in Definition 2.

Hereafter, processes A, B, C, D are assumed to be distinct. The $S\pi_{Er}$ process definition $A(\tilde{z})$ mapped using the $[\![.]\!]_e$ operator is denoted by $A_e(\tilde{z},\tilde{\rho})$ (and similarly for the other processes). Transitions between $S\pi_{Er}$ (respectively, $S\pi_e$) processes are denoted by $\rightarrow_e$ (respectively, $\rightarrow_E$). $\rightarrow_e^*$ stands for a sequence of $\rightarrow_e$.

The following lemma verifies that for each $S\pi_e$ process transition, there exists the corresponding transition between $S\pi_{Er}$ processes.

Lemma 1. $A_e(\tilde{z},\tilde{\rho})\rightarrow_e B_e(\tilde{w},\tilde{\varrho})$ *only if* $A(\tilde{z})\rightarrow_E B(\tilde{w})$.

Proof. Straightforward from Definition 2. □

The following lemmas ensure, for both internal actions and communications, that for each $S\pi_{Er}$ process transition, a sequence of transitions between $S\pi_e$ processes exists and this sequence follows the same distribution as the $S\pi_{Er}$ transition.

Lemma 2. $A(\tilde{z})\rightarrow\tau_t{}_E B(\tilde{w}) \Longrightarrow A_e(\tilde{z},\tilde{\rho}_1)\rightarrow_e^* B_e(\tilde{w},\tilde{\varrho}_1)$.

Proof. Let i be the index of the internal action τ_t in the sum defined by the process A. At the initial call of $A_e(\tilde{z}, \tilde{\rho}_1)$, $c_i = 1$ (9). As long as $c_i < sa_t$, only the transition $\mathrm{TAU}_{\mathrm{wait}}$ can be applied. After each of these transitions, c_i is incremented by 1. When c_i reaches sa_t, $\mathrm{TAU}_{\mathrm{eff}}$ is the only transition applicable, resulting in a transition toward $B_e(\tilde{w}, \tilde{\varrho}_1)$. □

Lemma 3. *The duration of* $A_e(\tilde{z}, \tilde{\rho}_1) \rightarrow_e^* B_e(\tilde{w}, \tilde{\varrho}_1)$ *follows the same distribution as the transition* $A(\tilde{z}) \xrightarrow[E]{\tau_t} B(\tilde{w})$.

Proof. From the previous lemma, there exists the sequence of transitions $\mathrm{TAU}_{\mathrm{wait}}, \ldots, \mathrm{TAU}_{\mathrm{wait}}, \mathrm{TAU}_{\mathrm{eff}}$:

$$A_e(\tilde{z}, \tilde{\rho}_1) \xrightarrow{\tau_t}_e \cdots A_e(\tilde{z}, \tilde{\rho}_k) \xrightarrow{\tau_t}_e B_e(\tilde{w}, \tilde{\varrho}_1),$$

where $\tilde{\rho}_k$ is $\tilde{\rho}_1$ having $c_i = k$. From the conditions for applying $\mathrm{TAU}_{\mathrm{eff}}$ transition, $k = sa_t$. □

Lemma 4. *If* $A(\tilde{z}) \xrightarrow{ay}_E B(\tilde{w})$ *and* $C(\tilde{z}') \xrightarrow{\overline{a}y}_E D(\tilde{w}')$,

$$A(\tilde{z})|C(\tilde{z}') \xrightarrow{\dot{a}}_E B(\tilde{w})|D(\tilde{w}') \Longrightarrow A_e(\tilde{z}, \tilde{\rho}_1)|C_e(\tilde{z}', \tilde{\rho}'_1) \rightarrow_e^* B_e(\tilde{w}, \tilde{\varrho}_1)|D_e(\tilde{w}', \tilde{\varrho}'_1).$$

Proof. The $\mathrm{IN}_{\mathrm{eff}}$ transition is always reachable after an $\mathrm{IN}_{\mathrm{init}}$ transition and a certain number of $\mathrm{IN}_{\mathrm{wait}}$ transitions, so $B_e(\tilde{w}, \tilde{\varrho}_1)$ is always reachable from $A_e(\tilde{z}, \tilde{\rho}_1)$. Similarly, $D_e(\tilde{w}', \tilde{\varrho}'_1)$ is always reachable from $C_e(\tilde{z}', \tilde{\rho}'_1)$ using $\mathrm{OUT}_{\mathrm{init}}$, $sa_a - 2\mathrm{OUT}_{\mathrm{wait}}$ transition, and one $\mathrm{OUT}_{\mathrm{eff}}$ transition. □

Lemma 5. *The duration of the sequence* $A_e(\tilde{z}, \tilde{\rho}_1)|C_e(\tilde{z}', \tilde{\rho}'_1) \rightarrow_e^* B_e(\tilde{w}, \tilde{\varrho}_1)|D_e(\tilde{w}', \tilde{\varrho}'_1)$ *follows the same distribution as the transition* $A(\tilde{z})|C(\tilde{z}') \xrightarrow{\dot{a}}_E B(\tilde{w})|D(\tilde{w}')$.

Proof. From the previous lemma and constraints on the $\mathrm{OUT}_{\mathrm{wait}}$ transitions, the following sequence of transitions exists and is always enabled:

$$\begin{aligned} A_e(\tilde{z}, \tilde{\rho}_1)|C_e(\tilde{z}', \tilde{\rho}'_1) &\xrightarrow{\dot{a}}_e A_e(\tilde{z}, \tilde{\rho}_2)|C_e(\tilde{z}', \tilde{\rho}'_2) \\ &\xrightarrow{\dot{a}'}_e \cdots A_e(\tilde{z}, \tilde{\rho}_k)|C_e(\tilde{z}', \tilde{\rho}'_k) \\ &\xrightarrow{\dot{a}''}_e B_e(\tilde{w}, \tilde{\varrho}_1)|D(\tilde{w}', \tilde{\varrho}'_1), \end{aligned}$$

with exactly $sa_a - 2$ transitions $\rightarrow \dot{a'}_e$. □

3.4 Simple Example

We apply the results obtained in this section to a simple example: an infinite looping process (10):

$$A(l) ::= \tau_t.A(1-l) \qquad l \in \{0, 1\}. \tag{10}$$

Basically, starting from $A(0)$, we expect to observe an infinite sequence of value change for l $(0, 1, 0, 1, \ldots)$. Between each transition, the internal action τ_t is performed.

The result of the map of the $S\pi_{Er}$ process defined in (10) to a $S\pi_e$ process is given by (11):

$$\begin{aligned} A(l, c, \emptyset, \emptyset) ::= {} & [c < sa_t]\tau_t.A(l, c+1, \emptyset, \emptyset) \\ & + [c = sa_t]\tau_t.A(1-l, 1, \emptyset, \emptyset). \end{aligned} \tag{11}$$

Fig. 4 plots the value of the argument l of process A during simulations of $[\![A(0)]\!]_e = A(0, 1)$ under SPiM [9], [19] with an identical rate but different stochasticity absorption factors.

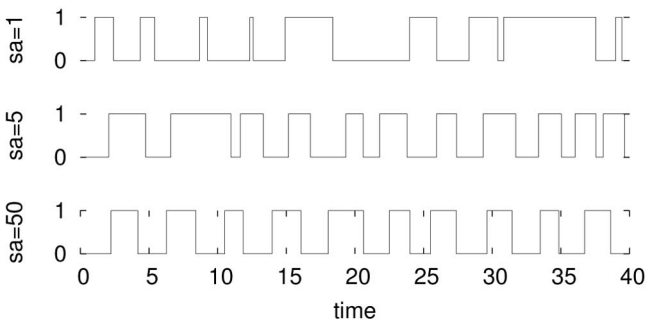

Fig. 4. Plot of the l value during simulations of the $S\pi_{Er}$ process defined by (10) with rate $\frac{1}{2}$ and different absorption factors sa.

When no stochasticity absorption is applied, we observe a strong variance of the duration of the internal actions, as imposed by the exponential distribution. By increasing the stochasticity absorption factor, this variance reduces. Regular oscillations are observed with a high stochasticity absorption factor.

4 Firing Intervals

The aim of this section is to point out a link between the use rate and stochasticity absorption factor of an action and the interval of time in which it is fired at a given confidence. This time interval is called the *firing interval* (Definition 3). This section provides a set of statistical tools that may help the modeling and checking of temporal and stochastic systems where durations of transitions follow an Erlang distribution.

Definition 3 (Firing Interval). *Given a use rate* r *and a stochasticity absorption factor* sa, *the* Firing Interval *at confidence coefficient* $1 - \alpha$ *is noted* $FI_\alpha(r, sa) = [d; D]$, *where the probabilities of firing the action within times* d *and* D *are given by* $F_{r,sa}(d) = \frac{\alpha}{2}$ *and* $F_{r,sa}(D) = 1 - \frac{\alpha}{2}$, *respectively;* $F_{r,sa}$ *is the CDF defined by (2) in Section 3.1.*

At modeling time, and more especially when parameterizing a model, it may be more natural to reason with firing intervals than with rate and stochasticity absorption factor pairs. Here, we provide estimators and approximating functions to translate between a firing interval and rate and stochasticity absorption parameters.

These bring different uses to the stochastic absorption factor, depending on the nature of the modeled system. On the one hand, the stochasticity absorption factor expresses the confidence in the precise duration of the action: the higher the confidence in the action duration, the more the stochasticity absorption factor can be raised. On the other hand, the stochasticity absorption supplies the ability to reproduce actions with intrinsic stochasticity where time bounds are known.

As supplementary modeling references, we point out that there exist methods for inferring the shape [20] and the scale [21] parameters of a gamma distribution from a set of time measurement data. The conversion from gamma parameters to Erlang parameters is discussed below.

4.1 From Parameters to Firing Interval

Given the rate and stochasticity absorption parameters of an action, we are interested in computing the confidence interval

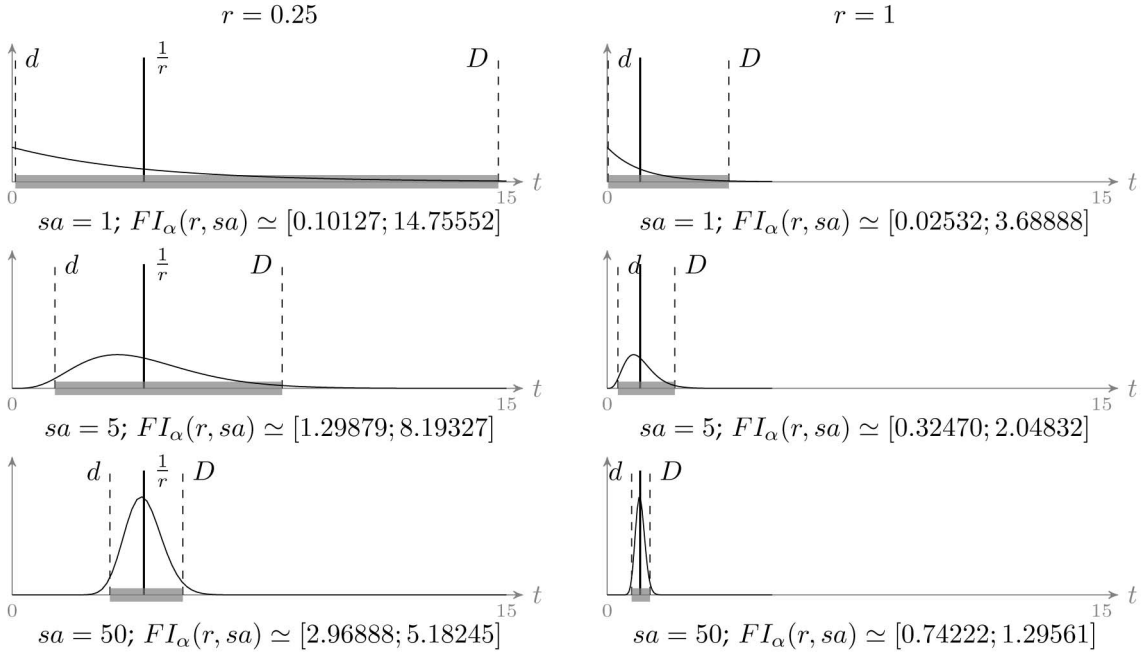

Fig. 5. Evolution of the firing interval when the stochasticity absorption factor sa increases. The thick vertical line is the average duration for each parameter. The confidence coefficient has been fixed to 95 percent ($\alpha = 0.05$).

for the time at which the action will be fired. Let $1-\alpha$ be the confidence coefficient for computing the firing interval. We search d and D such that $F_{r,sa}(d) = \frac{\alpha}{2}$ and $F_{r,sa}(D) = 1 - \frac{\alpha}{2}$, where $F_{r,sa}$ is the CDF of the Erlang distribution of use rate r and stochasticity absorption factor sa (2). The function which associates the time t with $0 \leq x \leq 1$ such that $F_{r,sa}(t) = x$ is known as the *quantile* function, and is noted F^{-1}.

Because of its relation with the incomplete gamma function, the quantile function of the gamma distribution, hence of the Erlang distribution, has no easy analytical expression and cannot be used directly [22]. However, efficient approximation algorithms for F^{-1} exist [23], [24]. The widely used statistical tool R [25] proposes an implementation of such an algorithm. As R is distributed with a C programming language library, one can easily access this implementation from independent programs. To compute the quantiles for the Erlang distribution using R, the `qgamma` function can be used. Here is an instance of a R session for computing the firing interval of an action at use rate r and stochasticity absorption factor sa:

$$d \leftarrow \texttt{qgamma}(\alpha/2, \texttt{shape} = sa, \texttt{rate} = r * sa),$$
$$D \leftarrow \texttt{qgamma}(1 - \alpha/2, \texttt{shape} = sa, \texttt{rate} = r * sa).$$

Fig. 5 shows the influence of the use rate and the stochasticity absorption factor on the firing interval.

4.2 From Firing Interval to Parameters

Given a firing interval $[d; D]$ at a confidence coefficient $1-\alpha$, we look for a rate r and stochastic absorption factor sa such that $FI_\alpha(r, sa) = [d, D]$. To achieve this goal, estimators of r and sa (respectively, $\hat{r}_\alpha$ and $\hat{sa}_\alpha$) are built as a function of the lower and upper bounds of the firing interval, d and D, respectively.

In a first phase, the integer constraint on the stochasticity absorption factor is released to consider the gamma distribution at use rate $r \in \mathbb{R}_+^*$ and stochasticity absorption factor $sa \in \mathbb{R}_+^*$. As there is no analytical expression of the quantile function of the gamma distribution, expressing r and sa as a function of the confidence interval cannot be done analytically either. Estimators have been obtained by regression on a set of generated rates and stochasticity absorption factors for which the firing intervals have been computed. For this paper, d and D were computed at a confidence coefficient of 95 percent for all rounded $\log sa$ between 0 and 4.5 (step of 0.1) and all $\log r$ between -8 and 2 (step of 0.1). Fig. 6 shows the heat map for the r and sa parameters as a function of the bounds d and D of firing intervals for such a generation.

From the 3D plots, we have manually determined regressions fitting the data. Parameters of these regressions have then been abstracted as they may depend on the confidence coefficient value. The estimator we propose for the use rate, noted $\hat{r}_\alpha$, is given by (12), and the one we propose for the stochasticity absorption factor, noted $\hat{sa}_\alpha$, is given by (13):

$$\hat{r}_\alpha = (w + x\exp(-y.d))(d + D)^{-1}, \tag{12}$$

$$\hat{sa}_\alpha = \exp\left(u\left(\frac{D}{d}\right)^v\right), \tag{13}$$

where u, v, w, x, y are parameters depending on the confidence coefficient $1-\alpha$.

The parameters of these expressions have been estimated using the tool R by fitting (12) and (13) to generated data. Table 2 sums up the estimators found for r and sa at different confidence coefficients. The lack of usable analytical expressions around the gamma distribution makes it difficult to evaluate. Fig. 7 shows an attempt to evaluate the quality of these estimators by comparing, for some generated data, the expected value and the estimated value. This kind of evaluation is similar to a confusion matrix, as used in machine learning [26]. It is worthwhile noticing that the estimators are bijective functions, i.e., each firing

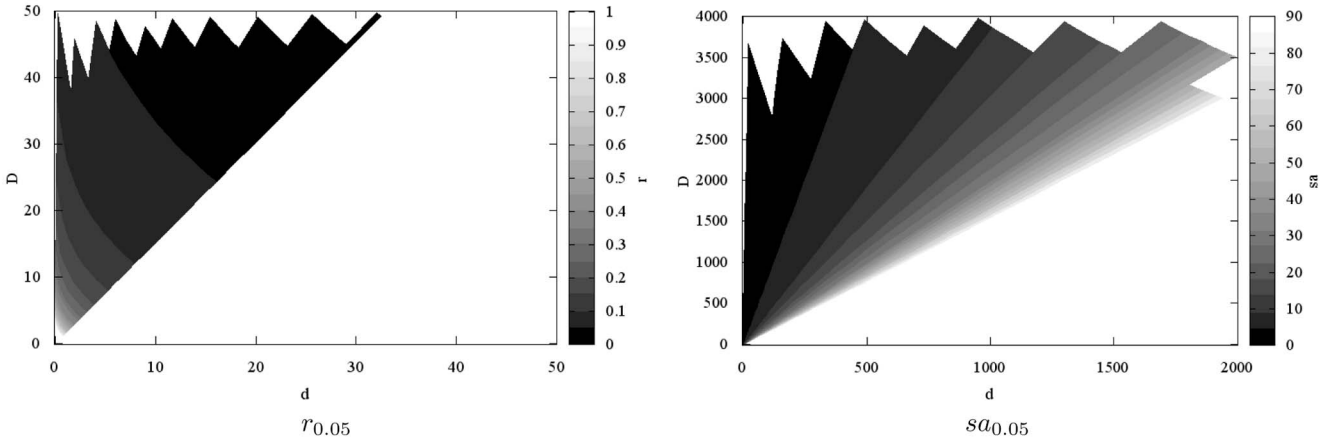

Fig. 6. Heat map of the use rate (left) and stochasticity absorption factor (right) as a function of the bounds d and D of the firing Interval $FI_{0.05} = [d; D]$ for some generated data.

TABLE 2
Obtained Estimators of Parameters r and sa for the Firing Interval $[d; D]$ at Different Confidence Coefficients

	$1-\alpha = 0.90$	$1-\alpha = 0.95$	$1-\alpha = 0.99$
$\hat{r}_\alpha$	$(2.13 + 2.89\exp(-44.46d))(d+D)^{-1}$	$(2.06 + 1.93\exp(-36.49d))(d+D)^{-1}$	$(2.03 + 1.39\exp(-33.33d))(d+D)^{-1}$
$\hat{sa}_\alpha$	$\exp\left(6.39\left(\frac{d}{D}\right)^{-0.66}\right)$	$\exp\left(6.41\left(\frac{d}{D}\right)^{-0.87}\right)$	$\exp\left(6.41\left(\frac{d}{D}\right)^{-1.04}\right)$

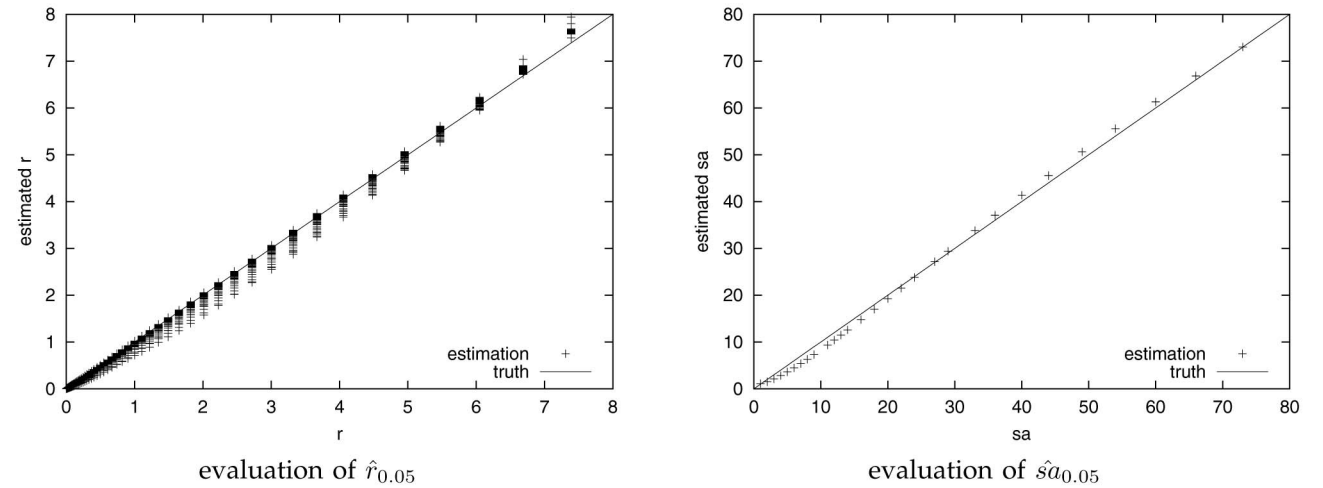

Fig. 7. Evaluation of obtained estimators for use rate r and stochasticity absorption factor sa with confidence coefficient of 95 percent on some generated data. Points are the estimated parameter value. A perfect estimator would put all points on the truth line.

interval corresponds to one and only one use rate and stochasticity absorption factor.

The resting point is the integer constraints on the stochasticity absorption factor. Because of the bijectivity of the estimators, if the estimated stochasticity absorption is not an integer, there is no Erlang distribution fitting with the given firing interval. From this observation, the search for approximating Erlang distribution parameters is at the confidence of the modeler. However, one can notice that rounding the estimated stochasticity absorption factor to the upward (respectivley, downward) integer results in a firing interval included by (respectively, including) the originally given firing interval. In that way, one can easily estimate an over or underapproximation of the couple of parameters matching an arbitrary firing interval. One can observe that the estimation of parameters for a given firing interval is really fast as it is an evaluation of the function exponential.

4.3 Sequence of Actions

To conclude this section, we briefly discuss the distribution of a sequence of actions and its relation with firing intervals.

Let us consider k actions with their respective use rates $r_1, \ldots, r_k$ and stochasticity absorption factors $sa_1, \ldots, sa_k$. Considering they are fired successively, what is the firing interval of the sum of the actions?

Unfortunately, it can be easily checked that the firing interval $[d; D]$ of a sequence of Erlang distributed actions is not the sum of the firing intervals of the individual actions—i.e., $d \neq d_1 + \cdots + d_k$ and $D \neq D_1 + \cdots + D_k$ where d_i (respectively, D_i) is the lower (respectively, upper) bound of the firing interval of the ith action. However, the sum of Erlang distributed random variables with different parameters has been studied in [27] and [28], and [29] gives an easily computable expression of the CDF for such a distribution. In this way, the firing interval of the sum of

Erlang distributed random variables can be computed using standard approximation techniques of the quantile function (like bisections). The dual operation, consisting of inferring the parameters of the Erlang sum from the firing interval, raises several difficulties, such as the loss of bijectivity. We consider such an issue as out of the scope of this paper.

5 Model-Checking Using PRISM

The probabilistic model checker PRISM [11] offers efficient model-checking for CTMCs. In PRISM, transitions are specified within PRISM modules. Each module has a finite set of local variables. The union of the local variables of all modules gives the global state of the model, denoted by V. A transition is the result of an action specified as follows:

$$[act]\ guard \rightarrow r : (x_1' = u_1)\ \&\ \ldots \&(x_k' = u_k),$$

where act is an optional action label, $guard$ a predicate over V, x_i is a local variable, and u_i a function over V. $r \in \mathbb{R}_+^*$ is the use rate of the action and is assumed to be 1 when omitted. To be applicable, a labeled action has to be synchronized with an action of the same label in another module. The rate of such a synchronized action is the product of the rates of both actions. x_i' stands for the value of x_i once the action is applied.

An efficient translation of the classical exponentially distributed stochastic π-calculus ($S\pi_e$) into PRISM has been proposed by Norman et al. [10]. Their translation requires the overall process structure to be rearrangeable to the form $P = \nu x_1 \ldots \nu x_k(P_1|\ldots|P_n)$, where each P_i contains no ν operator nor recursive use of the $|$ operator, especially to ensure a finite number of states. However, given an $S\pi_{Er}$ process P respecting these constraints, the constructed $S\pi_e$ process $[[P]]_e$ does not respect this limitation. Indeed, the proposed construction of the output action into $S\pi_e$ (7) involves a recursive generation of fresh channels a', a''. An obvious solution is to directly translate the $S\pi_{Er}$ process to PRISM.

In this section, the translation of $S\pi_e$ processes to PRISM proposed by Norman et al. [10] is adapted to the translation of $S\pi_{Er}$ process. Therefore, this allows an efficient model-checking of $S\pi_{Er}$ processes, which is a new result. To end, the overall approach of this paper is illustrated again by a simple example.

5.1 PRISM Construction of the Stochasticity Absorption Factor

Let P be a stochastic π-calculus expression of the form $P = \nu x_1 \ldots \nu x_k(P_1|\ldots|P_n)$, where each P_i contains neither the ν nor $|$ operator. In this way, each process P_i can be described by a transition graph where nodes are race conditions annotated by $Q_i, R_i, \ldots$ [10]. Hence, each $Q_i, R_i, \ldots$ stands for a state of the process P_i. Using the translation detailed in [10], the PRISM model corresponding to P can be computed. It results in n PRISM modules, one per P_i, each having a variable s_i representing the current state of the process P_i. Variables representing channels to be sent or to be received are also attached to modules. Obtained actions are of three different main forms:

$$[](s_i = Q_i)\ \&\ M \rightarrow r_t : (s_i' = R_i), \tag{14}$$

$$[a\_P_i\_P_j\_y](s_i = Q_i)\ \&\ M \rightarrow r_a : (s_i' = R_i), \tag{15}$$

$$[a\_P_j\_P_i\_y](s_i = Q_i)\ \&\ M \rightarrow (s_i' = R_i)\&(z' = y). \tag{16}$$

Each of these forms represents the process P_i at state Q_i applying a certain action under the condition M and thus changing to state R_i. These actions are, respectively, the internal action (14), the output (either bound or free) of y on a channel a to P_j (15), and the input of y as z on a channel a from P_j (16). Depending on the boundedness of the sent channel y, supplementary conditions are added to M. In the remainder of this section, we assume that any identical action labels inside the same module have disjoint guards, i.e., both are never part of the same race condition.

Support for the stochasticity absorption within this translation is added in a way similar to the construction of the stochasticity absorption factor within $S\pi_e$ presented in Section 3: A counter is attached to each action. When this counter reaches the stochasticity absorption factor, the transition is enabled. For identifying internal actions, a label is attached to them: P_i_t is the label of the internal action τ_t of P_i. The set of labels of internal and output actions of P_i is denoted by $\mathcal{L}_{P_i\diamond} = \{a\_P_i\_P_j\_y, \ldots, P_i\_t, \ldots\}$, and the set of labels of input actions of P_i is denoted by $\mathcal{L}_{\diamond P_i} = \{a\_P_j\_P_i\_y, \ldots\}$. Basically, there is one counter for each action having a label $l \in \mathcal{L}_{P_i\diamond}$. This counter is defined as a local variable c_l inside the PRISM module for P_i. Each time an output or an internal action is performed, the corresponding counter is incremented by one. The update of the action is processed when this counter reaches the expected stochasticity absorption factor.

When P_i changes state, each counter related to its new state has to be properly initialized. However, as PRISM forbids the update of variables belonging to other modules, P_i cannot directly update the counters it does not own that are related to all actions labeled by $l \in \mathcal{L}_{\diamond P_i}$. To cope with such an issue, a Boolean variable d_l is defined in the module P_i for each $l = a\_P_j\_P_i\_y \in \mathcal{L}_{\diamond P_i}$, and is set to true when the reset of c_l is required. The module owning c_l has to reset the counter when d_l is true.

The following section precisely describes the transformations needed to add support for the stochasticity absorption factor to each of the PRISM modules P_i resulting from translation of Norman et al [10].

5.1.1 Additional Local Variables

For each label $l \in \mathcal{L}_{P_i\diamond}$, the variable c_l stands for the counter of the stochasticity absorption of the action labeled by l:

$$c\_l : [1..sa\_l]\ init\ 1;$$

For each label $l \in \mathcal{L}_{\diamond P_i}$, the Boolean variable d_l is true if P_i has changed its state while the absorption of the action played by l has started. In other words, d_l is true if the associated counter c_l has to be reset:

$$d\_l : bool\ init\ false;$$

Hereafter, the PRISM update for the reset of all stochasticity absorption counters is denoted by $R_{P_i\diamond}$ (17). The update of d_l variables, $l \in \mathcal{L}_{\diamond P_i}$, is denoted by $S_{\diamond P_i}$ (18). Basically, d_l is set to true if and only if the associated counter c_l is different from its default value:

$$R_{P_i\diamond} \stackrel{def}{=} \&_{l\in\mathcal{L}_{P_i\diamond}} (c\_l' = 1), \tag{17}$$

$$S_{\diamond P_i} \stackrel{def}{=} \&_{l\in\mathcal{L}_{\diamond P_i}} (d\_l' = c\_l > 1), \tag{18}$$

with $\&_{i\in\{1,\dots,k\}} u_i = u_1 \& \cdots \& u_k$.

5.1.2 Internal Action

Let $l = P_i\_t$ be the label of an action resulting from the translation of an internal action τ_t. The use rate and the stochasticity absorption factor of this internal action are, respectively, r_t and sa_t. The PRISM action respects the following form:

$$[]\ G \rightarrow r_t : U;$$

where G stands for the guard of the action and U for the updates to perform. The stochasticity absorption of the action is achieved by replacing the previous statement by the actions below:

$$[]\ G\ \&\ (c\_l < sa_t) \rightarrow r_t * sa_t : (c\_l' = c\_l + 1);$$
$$[]\ G\ \&\ (c\_l = sa_t) \rightarrow r_t * sa_t : U\ \&\ R_{P_i\diamond}\ \&\ S_{\diamond P_i};$$

Therefore, the update U is only applied after exactly sa_t transitions at rate $r_t \cdot sa_t$ since P_i changed its state.

5.1.3 Channel Output

Let $l = a\_P_i\_P_j\_y$ be the label of an action resulting from the translation of an output of channel y on channel a. The use rate and the stochasticity absorption factor of this channel are, respectively, r_a and sa_a. The PRISM action respects the following form:

$$[l]\ G \rightarrow r_a : U;$$

where G stands for the guard of the action and U for the updates to be performed. The stochasticity absorption of the action is achieved by replacing the previous statement by the actions below:

$$[l\_wait]\ G\ \&\ d\_l \rightarrow r_a * sa_a : (c\_l' = 2);$$
$$[l\_wait]\ G\ \&\ !d\_l\ \&\ (c\_l < sa_a) \rightarrow r_a * sa_a : (c\_l' = c\_l + 1);$$
$$[l]\ G\ \&\ !d\_l\ \&\ (c\_l = sa_a) \rightarrow r_a * sa_a : U\ \&\ R_{P_i\diamond}\ \&\ S_{\diamond P_i};$$

To perform the update U, P_i has first to perform $sa_a - 1$ synchronizations on the l_wait label and finally one synchronization on the l label. The statement where d_l is true corresponds to a reset of the counter c_l to 1. Therefore, the value of this counter becomes 2 after the action is performed.

5.1.4 Channel Input

Let $l = a\_P_j\_P_i\_y$ be the label of an action resulting from the translation of an input of channel y on channel a. The PRISM action respects the following form:

$$[l]\ G \rightarrow U;$$

where G stands for the guard of the action and U for the updates to be performed. The stochasticity absorption of the action is achieved by replacing the previous statement by the actions below:

$$[l\_wait]\ G \rightarrow (d\_l' = false);$$
$$[l]\ G \rightarrow U\ \&\ R_{P_i\diamond}\ \&\ S_{\diamond P_i};$$

To perform the update U, P_i has first to perform $sa_a - 1$ synchronizations on the l_wait label and finally one synchronization on the l label. From the transformation corresponding to the channel output on P_j, the counter c_l is reset after a synchronization on l_wait. In this way, d_l has to be set to false in the update so the counter c_l can increment in future synchronizations.

By applying these transformations, it is ensured that each update U is performed after a duration following the sum of sa exponential random variables at rate $r \cdot sa$.

We point out that the proposed encoding of Erlang transitions within PRISM may be applied to other models than those resulting from the translation of a stochastic π-calculus expression. The main point of this construction is to reset the stochasticity absorption counter as soon as the corresponding synchronization is no longer possible. At each update, all synchronizations that are no longer possible have to be correctly detected. While this requires a precise handling of synchronization guards, this should be manageable for numerous PRISM models.

5.2 Simple Example

As an application of the use of the PRISM model checker for analyzing $S\pi_{Er}$ processes, and as an illustration of the overall method presented by this paper, we propose the study of an example process P defined in (19).

$$\begin{aligned} A_1(a,b,c) &::= a.A_0(c) + \overline{b}.A_1(a,b,c) \qquad A_0(c) ::= c.\mathbf{0}, \\ B_1(a,b,c) &::= b.B_0(c) + \overline{a}.B_1(a,b,c) \qquad B_0(c) ::= c.\mathbf{0}, \\ P &::= \nu a \nu b \nu c (A_1(a,b,c) | B_1(a,b,c)). \end{aligned} \tag{19}$$

Intuitively, two scenarios can happen: either A_1 outputs first on b then B_1 becomes B_0 and the system ends in deadlock; or B_1 outputs first on a then A_1 becomes A_0 and the system ends in deadlock.

For this simple example, we search for reducing close to zero the probability that A_1 becomes A_0, i.e., the probability that B_1 outputs on a. As supplementary constraints, the average duration for using channels a and b is fixed to, respectively, 4 and 1 time units (i.e., $r_a = 0.25$ and $r_b = 1$). For instance, these constraints may have been imposed by observations of the real system modeled by P. The property to be checked is expressed in PRISM as $\mathbf{P} = ?\ [\mathbf{F}(\mathtt{a} = 0)]$ which means the probability that A_0 is eventually run.

We first study this model as a $S\pi_e$ process. Listing 1 shows the result of the translation of P into PRISM using the method of Norman et al. Verifying the previously given property with PRISM results in a probability for running A_0 of 0.2.

```
module proc_A
    a: [0..1] init 1;
    [a_B1_A1] (a=1) -> (a'=0);
    [b_A1_B1] (a=1) -> r_b: (a'=1);
endmodule

module proc_B
    b: [0..1] init 1;
    [b_A1_B1] (b=1) -> (b'=0);
    [a_B1_A1] (b=1) -> r_a: (b'=1);
endmodule
```

Listing 1. Translation of the example stochastic π-calculus model (19) as a $S\pi_e$ process into PRISM.

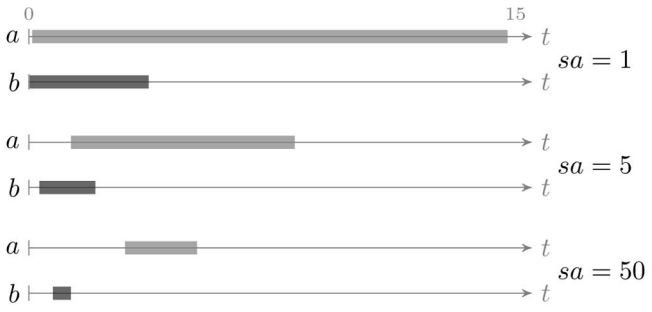

Fig. 8. Firing intervals at confidence coefficient 95 percent for a and b (19) with different stochasticity absorption factors but constant use rates $r_a = 0.25$ and $r_b = 1$.

Let us consider P as an $S\pi_{Er}$ process. For the sake of simplicity, we will consider that both a and b have the same stochasticity absorption factor.

In terms of firing intervals, we look for a stochasticity absorption factor for which the firing interval of b is entirely before the firing interval of a. By computing the firing interval for both these actions with different stochasticity absorptions factors—as shown in Fig. 8—one can observe that the probability of firing a before b will be significantly decreased with a stochasticity absorption factor of 5. By using a stochasticity absorption factor of 50, we expect the probability of firing a first to be close to zero at confidence coefficient 95 percent.

To confirm these results, we now turn to the model-checking of the $S\pi_{Er}$ process P using PRISM. Listing 2 shows the translation of P into PRISM using the construction presented above. With a stochasticity absorption of 5, the probability that A_0 runs is divided by 100 (almost 0.02) compared to the case of no stochasticity absorption. Increasing this stochasticity absorption factor to 50 reduces this probability to approximately 10^{-11}.

```
module proc_A
    a: [0..1] init 1;
    c_b_A1_B1: [1..sa_b] init 1;
    d_a_B1_A1: bool init false;

    [a_B1_A1_wait] (a=1) -> (d_a_B1_A1'=false);
    [a_B1_A1] (a=1) -> (a'=0) & (d_a_B1_A1'=false) &
         (c_b_A1_B1'=1);

    [b_A1_B1_wait] (a=1) & d_b_A1_B1 -> r_b:
         (c_b_A1_B1'=2);
    [b_A1_B1_wait] (a=1) & !d_b_A1_B1 & (c_b_A1_B1<sa_b)
         -> r_b: (c_b_A1_B1'=c_b_A1_B1+1);
    [b_A1_B1] (a=1) & !d_b_A1_B1 & (c_b_A1_B1=sa_b) ->
         r_b: (a'=1) & (c_b_A1_B1'=1) &
         (d_a_B1_A1'=c_a_B1_A1>1);
endmodule

module proc_B
    b: [0..1] init 1;
    c_a_B1_A1: [1..sa_a] init 1;
    d_b_A1_B1: bool init false;

    [b_A1_B1_wait] (b=1) -> (d_b_A1_B1'=false);
    [b_A1_B1] (b=1) -> (b'=0) & (d_b_A1_B1'=false) &
         (c_a_B1_A1'=1);

    [a_B1_A1_wait] (b=1) & d_a_B1_A1 -> r_a:
         (c_a_B1_A1'=2);
    [a_B1_A1_wait] (b=1) & !d_a_B1_A1 & (c_a_B1_A1<sa_a)
         -> r_a: (c_a_B1_A1'=c_a_B1_A1+1);
    [a_B1_A1] (b=1) & !d_b_A1_B1 & (c_a_B1_A1=sa_a) ->
         r_a: (b'=1) & (c_a_B1_A1'=1) &
         (d_b_A1_B1'=c_b_A1_B1>1);
endmodule
```

Listing 2. Translation of the example stochastic π-calculus model (19) as a $S\pi_{Er}$ process into PRISM.

The $S\pi_e$ translation of this example is given in Supplemental Material B, which can be found in the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSE.2010.95.

6 Applicability

This section discusses the applicability of the results presented in this paper. At first, the complexity and scalability of the translation of the Erlang models to exponentially distributed models is tackled. A case study demonstrating the benefits of using the stochasticity absorption factor concludes this section.

6.1 Complexity and Scalability

The proposed translations (either in stochastic π-calculus or in PRISM) are both linear with the size of the model, as they are simple rewritings. The addition of stochasticity absorption within stochastic π-calculus models should stay manageable by simulation tools, as it only decomposes actions into multisteps actions. The model-checking of models with numerous actions having high stochasticity absorption factors may suffer from a state space explosion, however. Techniques to overcome this difficulty are discussed in the Section 7. Finally, the conversion of a time interval into parameters of an Erlang distribution is efficient since it is simply the evaluation of the exponential function.

6.2 Case Study

We show the benefits of the contributions presented in this paper through a biological model of the segmentation processes for the metazoan. This system has already been studied in a differential equation framework in [14]. We propose to extend the study of the system by introducing stochasticity within reactions, modeled in stochastic π-calculus. The use of stochasticity absorption allows to reproduce and verify more precisely some timed properties.

The segmentation process is modeled as a marker, referred to as A, which periodically becomes active or inactive. The activity of the marker is controlled by a clock C and a global switch F. While F is active, the clock regularly changes level (active or inactive), and the marker, if deactivated, can become active. Finally, when the clock is active, it deactivates (if active) the marker. After a certain duration, the switch deactivates itself, preventing the clock and the marker from being active again. We propose a modeling of this system by the following stochastic π-calculus expression:

$$
\begin{aligned}
F_1(h) &::= \tau_f.F_0(h) \qquad F_0(h) ::= \bar{h}.\mathbf{0} \\
C_0(a,d,h) &::= \bar{a}.C_0(a,d,h) + \tau_c.C_1(a,d,h) + h.\mathbf{0} \\
C_1(a,d,h) &::= \bar{d}.C_1(a,d,h) + \tau_c.C_0(a,d,h) \\
A(l,a,d) &::= a.A(1,a,d) + d.A(0,a,d) \\
&\nu a \nu d \nu h(F_1(h)|C_0(a,d,h)|A(0,a,d)).
\end{aligned}
$$

The switch, initially active (F_1), waits for a while before becoming inactive (F_0); F_0 outputs on the channel h (for "halt"), indicating that the system has to stop. If the clock is inactive (C_0 is running), it outputs on a to activate the marker (A sets its level l to 1 when inputting on a). Concurrently, the inactive clock waits for a delay τ_c to activate itself (C_1). As soon as C_0 inputs on h (i.e., the switch

is off), it becomes the null process, disabling the whole system. Finally, the active clock outputs on d to deactivate the marker (A sets its level l to 0 when inputting on d); concurrently, it waits for a delay τ_c to deactivate itself.

The challenge of this modeling is to obtain a system producing a fixed number of segments, i.e., the marker becomes active a fixed number of time. We first use exponentially distributed actions, and set the following rates: $r_f = 0.02$ (on average, the switch remains active 50 time units); $r_c = 0.1$ (on average, the clock changes its level every 10 time units); $r_a = r_d = 1$ (the activation/deactivation of the marker takes one time unit, on average); and $r_h = 10$. Thus, on average, we expect to observe three marker activations.

Using PRISM, we compute the probability of observing exactly three marker activations and obtain 0.15, which is rather low. This result is not surprising since the exponential distribution has a large variance. By reducing the variance of the delays τ_f and τ_c, we intend to reduce the variance of the number of clock activations, and thus, by transitivity, the variance of the number of marker activations. This variance reduction is achieved through the use of stochasticity absorption factors. We arbitrarily fix the stochasticity absorption factor $sa_c = 15$ (leading to a firing interval for τ_c between 4.6 and 17.9 time units, at confidence coefficient 99 percent) and use the estimators of Section 4.2 to obtain the parameters for a firing interval between 45 and 55 time units (giving $r_f = 0.0202$ and $sa_f = 608$). Having set these parameters and using the translation of the Erlang distribution within PRISM proposed in Section 5 (Listing 3), the probability of observing exactly three marker activations increases to 0.86.

```
module proc_a
    a: [0..1] init 0; // state
    marks: [0..10] init 0;
    [act] a=0 & marks<10 -> (a'=1) & (marks'=marks+1);
    [inh] a=1 -> (a'=0);
endmodule

module proc_c
    c: [0..2] init 0; // state
    c_wait: [1..sa_c] init 1;
    [] c=0 & c_wait<sa_c -> r_c*sa_c: (c_wait'=c_wait+1);
    [] c=0 & c_wait=sa_c -> r_c*sa_c: (c'=1) & (c_wait'=1);
    [] c=1 & c_wait<sa_c -> r_c*sa_c: (c_wait'=c_wait+1);
    [] c=1 & c_wait=sa_c -> r_c*sa_c: (c'=0) & (c_wait'=1);
    [halt] c=0 -> (c'=2);
    [act] c=0 -> r_a: (c'=0);
    [inh] c=1 -> r_a: (c'=1);
endmodule

module proc_f
    f: [0..1] init 1; // state
    f_wait: [1..sa_f] init 1;
    [] f=1 & f_wait<sa_f -> r_f*sa_f: (f_wait'=f_wait+1);
    [] f=1 & f_wait=sa_f -> r_f*sa_f: (f'=0) & (f_wait'=1);
    [halt] f=0 -> r_h: (f'=0);
endmodule
```

Listing 3. PRISM model from the translation of the Erlang stochastic π-calculus models of metazoan segmentation.

The computation time of the probability rises from a few milliseconds to a few minutes, showing a large growth of the state space caused by the introduction of stochasticity absorption factors. Supplemental Material C, which can be found in the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSE.2010.95, shows a study on the sensitivity of stochasticity absorption factors sa_c and sa_f for the intended probability.

The case study, based on the contributions of this paper, has allowed us to model quite precisely temporal properties in a stochastic framework. It also demonstrated the influence of the stochasticity on the observed results, such as the number of segments produced by the system.

7 Our Contribution and Related Work

The main contributions of this paper are the construction of the stochastic π-calculus with Erlang distribution using the Markovian stochastic π-calculus, together with the construction of Erlang distribution in PRISM models translated from the stochastic π-calculus. This paper also provides tools establishing the link between temporal intervals and parameters of Erlang distributions.

It is worth noticing that the simulation of the stochastic π-calculus with general distribution has been studied in [16]. The simulation works by recomputing the statistical distribution of transitions after a transition has been chosen, to take into account the elapsed time. Based on this method, an implementation of the simulation of non-Markovian processes for the BlenX language, which is close to the stochastic π-calculus, has recently been proposed [30]. Another method is proposed in [31], which addresses the simulation of non-Markovian chemical reactions, by precisely identifying each reacting molecules and by computing a priori the reaction times, and then selecting the lowest. These approaches are quite different from the one proposed in this paper as they apply only for simulation purposes, whereas our construction allows the direct use of analysis tools, such as model-checking tools (e.g., PRISM).

The method presented in this paper allows the model-checking of stochastic π-Calculus with Erlang distribution, under the restriction imposed by the PRISM translation (Section 5). It is achieved by a translation of a model using Erlang distribution into a Markovian model. Whereas the model-checking is still manageable for a few transitions having an Erlang distribution, it suffers from a state space explosion when using large stochasticity absorption factors (Section 6). However, one can notice that the CTMCs obtained are very structured, suggesting that methods such as symmetry reductions [32] or abstractions of sequences of transitions [33] may provide more efficient probabilistic model-checking.

Little work has been done to apply probabilistic model-checking directly to models with general distributions. The model-checking of semi-Markovian chains is studied in [34], where time spent in states follows a general distribution. Despite some good results for the checking of a few properties, they raise a negative conclusion, that is, the model-checking becomes computably highly complex in the scope of general distributions. An efficient model-checking of stochastic automata with general distributions against a simple probabilistic temporal logic is provided in [35].

In [36], the authors propose reducing the gap between stochastic and time extensions of Petri nets. They define a new way to describe the structure of the nets, Discrete Phase Type Timing, and use it to build both a functional model in the sense of a time Petri net [13] and a stochastic model. The so-called discrete phase type timing can represent either probabilistic or nondeterministic choice over an interval.

8 Conclusion

In this paper, we presented a technique for tuning temporal features within the stochastic π-calculus. This tuning is done through a stochasticity absorption factor, which reduces the variance around the average duration of transitions, allowing then the specification of transitions within firing time intervals, given a confidence coefficient. The stochasticity absorption is achieved by replacing the exponential distribution for firing actions by the Erlang distribution. Estimators permitting the computation of stochastic parameters for transitions from a desired firing time interval have been provided in this paper. We claim that such an approach makes it possible to have more accurate specifications of the timed behavior of a stochastic system.

We presented a translation of the Erlang distributed stochastic π-calculus into the exponential distributed one. In this way, such tuned models can be simulated and analyzed using the large panels of standard tools, which assume an exponential distribution of transitions. The probabilistic model-checking of the stochastic π-Calculus with Erlang distribution is achieved using PRISM, by a construction of the stochasticity absorption factors within PRISM models obtained from a translation of the stochastic π-Calculus [10]. It has been raised that, while the simulation of the tuned models is still efficient, the model-checking may suffer from a state space explosion. Future work may study the use of current state-of-the-art techniques of state space reduction in CTMCs to overcome this explosion.

The applicability of our approach to the modeling of a biological system has also been shown, giving its usefulness for modeling computational processes in general. The tuning of temporal parameters within stochastic models provides a precise method for models in which both stochasticity and time are important features.

References

[1] R. Milner, *Communication and Concurrency.* Prentice-Hall, Inc., 1989.

[2] M. Abadi and A.D. Gordon, "A Calculus for Cryptographic Protocols: The Spi Calculus," *Information and Computation,* vol. 148, pp. 36-47, 1999.

[3] C. Priami, "Stochastic π-Calculus," *The Computer J.,* vol. 38, no. 7, pp. 578-589, 1995.

[4] C. Priami, A. Regev, E. Shapiro, and W. Silverman, "Application of a Stochastic Name-Passing Calculus to Representation and Simulation of Molecular Processes," *Information Processing Letters,* vol. 80, no. 1, pp. 25-31, 2001.

[5] C. Kuttler and J. Niehren, "Gene Regulation in the pi Calculus: Simulating Cooperativity at the Lambda Switch," *Trans. Computational Systems Biology,* pp. 24-55, Springer, 2006.

[6] R. Blossey, L. Cardelli, and A. Phillips, "Compositionality, Stochasticity and Cooperativity in Dynamic Models of Gene Regulation," *HFSP J.,* vol. 2, no. 1, pp. 17-28, Feb. 2008.

[7] L. Cardelli, E. Caron, P. Gardner, O. Kahramanogullari, and A. Phillips, "A Process Model of Actin Polymerisation," *Electronic Notes in Theoretical Computer Science,* vol. 229, pp. 127-144, 2009.

[8] D.T. Gillespie, "Exact Stochastic Simulation of Coupled Chemical Reactions," *The J. Physical Chemistry,* vol. 81, no. 25, pp. 2340-2361, 1977.

[9] A. Phillips and L. Cardelli, "Efficient, Correct Simulation of Biological Processes in the Stochastic pi-Calculus," *Computational Methods in Systems Biology,* Springer, 2007.

[10] G. Norman, C. Palamidessi, D. Parker, and P. Wu, "Model Checking Probabilistic and Stochastic Extensions of the π-Calculus," *IEEE Trans. Software Eng.,* vol. 35, no. 2, pp. 209-223, Mar./Apr. 2009.

[11] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker, "PRISM: A Tool for Automatic Verification of Probabilistic Systems," *Proc. 12th Int'l Conf. Tools and Algorithms for the Construction and Analysis of Systems,* 2006.

[12] R. Alur and D. Dill, "The Theory of Timed Automata," *Proc. REX Workshop Real-Time Theory in Practice,* pp. 45-73, 1992.

[13] P.M. Merlin, "A Study of the Recoverability of Computing Systems," PhD dissertation, 1974.

[14] P. Francois, V. Hakim, and E.D. Siggia, "Deriving Structure from Evolution: Metazoan Segmentation," *Molecular Systems Biology,* vol. 3, 2007.

[15] R. Milner, *Communicating and Mobile Systems: The π-Calculus.* Cambridge Univ. Press, 1999.

[16] C. Priami, "Stochastic π-Calculus with General Distributions," *Proc. Fourth Workshop Process Algebras and Performance Modelling,* pp. 41-57, 1996.

[17] A. Phillips, L. Cardelli, and G. Castagna, "A Graphical Representation for Biological Processes in the Stochastic pi-Calculus," *Trans. Computational Systems Biology,* pp. 123-152, Springer, 2006.

[18] M. Evans, N. Hastings, and B. Peacock, *Statistical Distributions,* third ed. Wiley-Interscience, 2000.

[19] A. Phillips, *SPiM,* http://research.microsoft.com/~aphillip/spim, 2009.

[20] A. Zaigraev and A. Podraza-Karakulska, "On Estimation of the Shape Parameter of the Gamma Distribution," *Statistics & Probability Letters,* vol. 78, no. 3, pp. 286-295, 2008.

[21] J. Mi and A. Naranjo, "Inferences about the Scale Parameter of the Gamma Distribution Based on Data Mixed from Censoring and Grouping," *Statistics & Probability Letters,* vol. 62, no. 3, pp. 229-243, 2003.

[22] D.J. Wilkinson, *Stochastic Modelling for Systems Biology.* Chapman & Hall/CRC, 2006.

[23] D. Best and D. Roberts, "Algorithm AS91: The Percentage Points of the Chi-Squared Distribution," *Applied Statistics,* vol. 24, no. 3, pp. 385-390, 1975.

[24] A.R. DiDonato and A.H. Morris Jr., "Computation of the Incomplete Gamma Function Ratios and Their Inverse," *ACM Trans. Math. Software,* vol. 12, no. 4, pp. 377-393, 1986.

[25] R Development Core Team, *R: A Language and Environment for Statistical Computing,* R Foundation for Statistical Computing, http://www.R-project.org, 2009.

[26] R. Kohavi and F. Provost, "Glossary of Terms," *Applications of Machine Learning and the Knowledge Discovery Process* vol. 30, editorial for the special issue, 1998.

[27] S. Amari and R. Misra, "Closed-Form Expressions for Distribution of Sum of Exponential Random Variables," *IEEE Trans. Reliability,* vol. 46, no. 4, pp. 519-522, Dec. 1997.

[28] S. Nadarajah, "A Review of Results on Sums of Random Variables," *Acta Applicandae Mathematicae,* vol. 103, no. 2, pp. 131-140, 2008.

[29] S. Favaro and S. Walker, "On the Distribution of Sums of Independent Exponential Random Variables via Wilks' Integral Representation," *Acta Applicandae Mathematicae,* vol. 109, no. 3, pp. 1035-1042, 2010.

[30] I. Mura, D. Prandi, C. Priami, and A. Romanel, "Exploiting Non-Markovian Bio-Processes," *Electronic Notes in Theoretical Computer Science,* vol. 253, no. 3, pp. 83-98, 2009.

[31] M.A. Gibson and J. Bruck, "Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels," *The J. Physical Chemistry A,* vol. 104, no. 9, pp. 1876-1889, 2000.

[32] M. Kwiatkowska, G. Norman, and D. Parker, "Symmetry Reduction for Probabilistic Model Checking," *Proc. 18th Int'l Conf. Computer Aided Verification,* pp. 234-248, 2006.

[33] J.-P. Katoen, D. Klink, M. Leucker, and V. Wolf, "Abstraction for Stochastic Systems by Erlang's Method of Stages," *Proc. 19th Int'l Conf. Concurrency Theory,* pp. 279-294, 2008.

[34] G. López, H. Hermanns, and J.-P. Katoen, "Beyond Memoryless Distributions: Model Checking Semi-Markov Chains," *Proc. Joint Int'l Workshop Process Algebra and Probabilistic Methods, Performance Modelling and Verification,* pp. 57-70, 2001.

[35] J. Bryans, H. Bowman, and J. Derrick, "Model Checking Stochastic Automata," *ACM Trans. Computational Logic,* vol. 4, no. 4, pp. 452-492, 2003.

[36] A. Bobbio and A. Horváth, "Petri Nets with Discrete Phase Type Timing: A Bridge between Stochastic and Functional Analysis," *Electronic Notes in Theoretical Computer Science,* vol. 52, no. 3, pp. 209-226, 2002.

Loïc Paulevé received the MS degree in computer science from the École Normale Supérieure de Cachan, Brittany extension, France, in 2008. He is currently a PhD student at the Research Institute for Communications and Cybernetics of Nantes (IRCCyN-UMR CNRS 6597). His research interests include modeling and formal checking of biological complex systems.

Morgan Magnin received the PhD degree from the University of Nantes in 2007, about discrete and dense-time extensions of Petri nets. He is an associate professor at the École Centrale de Nantes and he carries out his research activity at the Research Institute for Communications and Cybernetics of Nantes (IRCCyN-UMR CNRS 6597). His work focuses on the modeling and the verification of dynamic concurrent systems. He has a special interest in studying how formal methods can be applied to biological systems (e.g., gene regulatory networks).

Olivier Roux received the PhD degree in 1985 and the HDR in 1992. He has been a professor of computer science at the École Centrale de Nantes since 1997. He was previously an associate professor at the Université de Nantes and an invited professor at Laval University (Québec, Canada). Deeply interested in models of time in computer science, he is the founder and former head of the research team MOVES at IRCCyN, in the domain of formal methods for modeling and verifying dynamical complex systems, and he has been particularly involved in bioinformatics for almost a decade. He created the international MOVEP summer school about modeling and verifying parallel processes (nine occurrences of the school have already taken place in France, Belgium, and Germany) in 1994. He is currently also "directeur scientifique adjoint" at the French Ministry of Higher Education and Research.