mathematics

MDPI

Article

Analyzing Non-Markovian Systems by Using a Stochastic Process Calculus and a Probabilistic Model Checker

Gabriel Ciobanu

Faculty of Computer Science, Alexandru Ioan Cuza University, 700506 Iaşi, Romania; gabriel@info.uaic.ro

Abstract: The non-Markovian systems represent almost all stochastic processes, except of a small class having the Markov property; it is a real challenge to analyze these systems. In this article, we present a general method of analyzing non-Markovian systems. The novel viewpoint is given by the use of a compact stochastic process calculus developed in the formal framework of computer science for describing concurrent systems. Since phase-type distributions can approximate non-Markovian systems with arbitrary precision, we approximate a non-Markovian system by describing it easily in our stochastic process calculus, which employs phase-type distributions. The obtained process (in our calculus) are then translated into the probabilistic model checker PRISM; by using this free software tool, we can analyze several quantitative properties of the Markovian approximation of the initial non-Markovian system.

Keywords: process calculus; phase-type distribution; non-Markovian systems; model checker PRISM

MSC: 60E05; 60J20; 62E17; 62M09; 65C60; 68Q60; 68Q85

1. Introduction

The progress in computing technologies made in the last decades has influenced the sciences (including mathematics) in fundamental ways, but also offers new opportunities for deeper mathematical research. Thanks to the significantly computation power, mathematicians may extend their classical techniques, proofs and solutions by using an algorithmic approach that enables the consideration of more complex models with wider applicability. While providing powerful tools to mathematicians, the technologies also create new challenges and problems.

According to the properties of their states and transitions, the stochastic systems are separated into non-Markovian and Markovian systems. A *non-Markovian system* is one that does not have the Markov property. On the other hand, a *Markovian system* possess the Markov property. The Markov property says that the probability of a future state is only dependent on the current state and independent of any previous state (i.e., the past event does not affect the future event; this is the reason why Markov property is also known as the memoryless property).

Although the non-Markovian systems represent almost all of the stochastic processes (with the exception of a small class having the Markov property), it is not easy to describe and analyze them. We present in this paper a new approach inspired by the theory of concurrency [1] and use a stochastic process calculus and a probabilistic model checker. The task is challenging because the existing probabilistic and stochastic process calculi are not convenient for describing non-Markovian systems. The drawback of the existing probabilistic process calculi is that almost all of them were developed for Markovian systems described in terms of Markov chains. The Markov chains have a well-established mathematical theory [2]; this theory allows the performance analysis and facilitates the exact numerical value of passage time, transient and steady-state performance measures. On the other hand, the theory dealing with non-Markovian systems is much less developed,

check for updates

Citation: Ciobanu, G. Analyzing Non-Markovian Systems by Using a Stochastic Process Calculus and a Probabilistic Model Checker. *Mathematics* **2023**, *11*, 302. https://doi.org/10.3390/math11020302

Academic Editor: Alexander Zeifman

Received: 17 November 2022
Revised: 15 December 2022
Accepted: 27 December 2022
Published: 6 January 2023

and in general, the performance measures are not derived analytically (only approximated). These performance measures are generally determined either by using non-Markovian approaches which require specific (discrete events) simulation techniques, or by using a Markovian approximation for the behavior of the non-Markovian system (and then analyze this Markovian approximation). Due to the fact that phase-type distributions are known to be able to approximate any distribution arbitrarily well [3,4], the non-Markovian systems can be described in terms of the phase-type distributions. Phase-type distributions are also adequate for such an approximation because of their strong properties: they are closed under finite convolutions, minimum, maximum and convex mixtures (contrasting the exponential distributions that are closed only under minimum).

The main contribution of this article is given by a novel methodology of analyzing non-Markovian systems via a concise stochastic process calculus using phase-type distributions and a model checker to verify several properties. As a consequence, we overcome an important impediment in dealing with non-Markovian systems: the lack of appropriate software tools. As far as we know, currently, a similar approach does not exist.

To summarize, we tackle the problem of efficiently analyzing non-Markovian systems by examining three main possibilities currently available:

- using non-Markovian process calculi (poor theory, few tools),
- using phase-type approximations for non-Markovian systems (strong theory, no tools),
- replacing non-Markovian distributions with exponential ones (strong theory, many tools, inaccurate approximation).

Among these options, the second one appeared to offer us a good balance between mathematical tractability and accuracy (i.e., the stochastic properties of the system are captured in a relatively faithful manner). However, it lacks a suitable software tool. To solve this situation, by using the process calculus employing phase-type distributions for transition durations, we provide a gradual description of how the processes of this calculus can be translated into a free advanced software, namely a probabilistic model checker called PRISM [5]. This approach makes possible the approximation of the non-Markovian systems up to any desired level of accuracy, and the use of all the automated facilities of PRISM to analyze (the approximation of) these systems.

The method presented in this article is applicable to a large number of concrete examples. Just to emphasize the method, we prefer to focus on its general steps and to illustrate it just by a rather theoretical example. The potential practical applications are mentioned by some fields in which this general method could be applied.

The structure of the paper is as follows. We briefly recall non-Markovian systems and phase-type distributions in Section 2. Then, we describe the syntax and semantics of the stochastic process calculus (named PHASE) designed to describe non-Markovian systems by using phase-type distributions. This process calculus is then translated faithfully into the language of the model checker PRISM [5]. This probabilistic model checker is used as a software tool to analyze various properties of the phase-type approximations of the non-Markovian system described by our process calculus. In Section 4, it is described how the process calculus PHASE can be implemented in PRISM, and in Section 5, an overview of the popular tools fitting phase-type distributions is presented. Finally, the whole approach is illustrated by considering a non-Markovian system, describing it in PHASE calculus, and analyzing it using PRISM.

2. Non-Markovian Models and Phase-Type Distributions

In order to present a Markovian system, let us consider the time spent by a person in (the employment of) a company. We assume that the person evolves through successive states, spending a period of time in each state. In a (continuous time) Markov chain model, the time spent in each state has an exponential distribution. When leaving a state, the person either moves on to the next state or leaves the company (by dismissal or death). Thus, we have successive labelled states together with a final state expressing that the

person left the company; this is an absorbing state (an absorbing state is a state that, once entered, cannot be left; a state which is not absorbing is called transient).

In a (more realistic) non-Markovian system, we may assume the same except the fact that the time spent in each state has an arbitrary distribution. Since we abandon the assumption of exponentially distributed times in each state, such a system no longer has the Markov property (and so, it is no longer a Markov chain). Let us consider a continuous-time Markovian process with a number of states, such that the first states are transient states and the final state is an absorbing state. The process has an initial probability of starting in any of the transitions between states.

Two non-Markovian systems are depicted in Figure 1 in terms of states and labelled transitions; each label indicates an action (a, b, c, d) and a distribution (both exponential and nonexponential distributions could be be used). The distributions involved in these examples are normal, uniform, log-normal and Weibull distributions (these are some of the non-Markovian distributions most frequently encountered in practice).

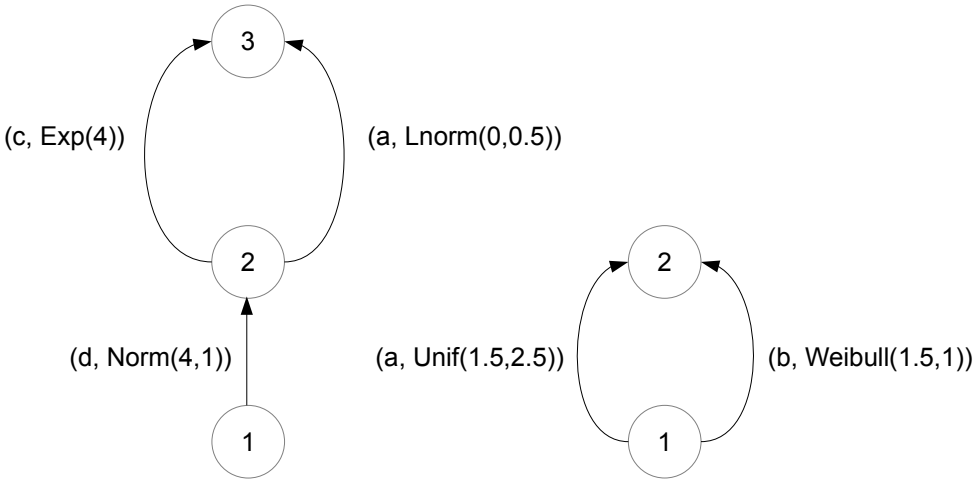

Figure 1. Examples of non-Markovian systems.

The time associated with each transition is indicated by the following distributions: $Norm(4,1)$ is a normal distribution with a mean of 4 and a standard deviation of 1, $Exp(4)$ is an exponential distribution with a rate of 4, $Lnorm(0,0.5)$ is a log-normal distribution with a mean of 0 and a standard deviation of 0.5 on the log scale, $Unif(1.5,2.5)$ is a uniform distribution with an upper bound 2.5 and a lower bound 1.5, while $Weibull(1.5,1)$ is a Weibull distribution with scale 1 and shape of 1.5.

It is known that general distributions can be approximated by phase-type distributions. A phase-type distribution is defined as the distribution of the absorption time in a Markov chain, namely the distribution of time from the starting state until the absorption in the final state. The distribution can be represented by a random variable describing the time until absorption of a Markovian process with one absorbing state. The name of phase-type distribution comes from the time taken from the starting state until absorption in the final state through a number exponentially distributed phases. These phases are equivalent to the states of the underlying Markov chain (we use the term phase when referring to distributions, and state when referring to Markov chains).

The method of phases uses an exponential time for the transitions of a continuous time Markov chain; such a transition corresponds to one phase. This approach provides algorithmically tractable solutions in closed matrix forms, such as formulas for densities, Laplace transforms and moments. Thus, phase-type distribution can approximate complex problems and solve them in an algorithmic and computational way. In this way, phase-type distributions represent a useful instrument for describing real-world phenomena in an tractable way.

The phase-type distribution is essentially a probability distribution constructed by a convolution or mixture of exponential distributions. It is in fact a class of distributions,

including exponential distribution, Erlang distribution, geometric distribution and Coxian distributions (among others). The properties of the phase-type distributions provide a good balance between generality and mathematical tractability. The most important property is that any (positive) probability distribution can be arbitrarily close approximated through a phase-type distribution; this is a consequence of the result that the class of phase-type distributions is dense in the set of non-negative probability distributions [3].

More information about the phase-type distributions can be found in [4].

3. A Stochastic Process Calculus Using Phase-Type Distributions

To integrate properly the phase-type distributions into a stochastic process calculus, we design a compact process calculus inspired by PEPA [6], PEPA$^{\infty}_{ph}$ [7] and IMC [8]. For convenience, we consider the phase-type representations in which the probability of the starting state is equal to 1 (i.e., there is a single initial state); this means that they can be specified fully in terms of their infinitesimal generator matrix [9]. We denote by $PH(A)$ the phase-type distribution whose generator is A; the distribution $PH(A)$ describes the time until absorption for a continuous-time Markov chains of size $ord(A)$ (the order of A) denoted by $CTMC(A)$, where state $ord(A)$ is absorbing and all the other states are transient. The rate of a transition from state i to state j is given by $A(i,j)$, where $1 \leq i,j \leq ord(A)$ and $i \neq j$; the element $A(i,i)$ with $1 \leq i \leq ord(A)$ is the sum of the rates of all the transitions originating in state i.

The process calculus PHASE uses only the following operators: the *sequential operator*, the *choice operator*, and the *parallel operator*. The syntax of PHASE is given in Table 1, where P_{seq} is a sequential process, P_{par} is a parallel process, α is an action, $(\alpha, PH(A))$ is a phase-type transition, $\{L\}$ is a set of actions and n is a natural number with $n \geq 2$.

Table 1. Syntax of the process calculus PHASE.

P_{seq}	::=	$(\alpha, PH(A)).P_{seq} \mid (\alpha_1, PH(A_1)).P^1_{seq} + \ldots + (\alpha_n, PH(A_n)).P^n_{seq}$
P_{par}	::=	$P_{seq} \mid P^1_{par} \underset{\{L\}}{\S} P^2_{par}$

The expression $(\alpha, PH(A)).P_{seq}$ specifies that action α is performed after a delay distributed according to $PH(A)$, and then behaves like P_{seq}. The expression $(\alpha_1, PH(A_1)).P^1_{seq} + \ldots + (\alpha_n, PH(A_n)).P^n_{seq}$ specifies a race among the transitions $(\alpha_1, PH(A_1))$ with $1 \leq i \leq n$; in this race for execution it is selected the transition with the shortest time delay, and its corresponding action is performed (the other transitions are discarded).

Considering the non-Markovian systems presented in Figure 1, the PHASE process corresponding to the system located on the left-hand side of the figure can be described by

$$P1_1 = (d, PH(A_1)).\left((c, PH(A_2)).P1_3 + (a, PH(A_3)).P1_3\right)$$

and the PHASE process corresponding to the right-hand side of the figure by

$$P2_1 = (a, PH(A_4)).P2_2 + (b, PH(A_5)).P2_2\ ,$$

where $PH(A_m)$ (with $m = 1..5$) are some phase-type distributions approximating the non-Markovian distributions of Figure 1, and $P1_1, P1_3, P2_1, P2_2$ represent some PHASE processes in the corresponding states indicated by subscripts.

Additionally, the parallel expression $P^1_{par} \underset{\{L\}}{\S} P^2_{par}$ indicates that the processes P^1_{par} and P^2_{par} must synchronize whenever performing a transition whose action is included in the *cooperation set* $\{L\}$. More exactly, for each action $\alpha \in \{L\}$, whenever P^1_{par} finishes a transition $(\alpha, PH(A_1))$, then P^1_{par} is blocked until P^2_{par} ends the corresponding transition $(\alpha, PH(A_2))$, and vice-versa. Thus, this operator indicates the cooperation of the involved processes along some shared transitions. By using our process calculus PHASE, we can put together the two systems of Figure 1 by synchronizing them using their common action a. Thus, the new non-Markovian system obtained by synchronizing the two systems of Figure 1 can be described in PHASE by a process $Sys = P1_1 \underset{\{a\}}{\S} P2_1$.

On the other hand, the transitions with actions not included in $\{L\}$ can go ahead unaffected by this cooperation. It is worth to mention that there are not defined associativity rules for the parallel composition; this means that the order in which the processes are composed should be explicit (by using parentheses). Both the choice and the parallel operators are commutative.

For defining the operational semantics of PHASE, we differentiate transition durations from the occurrence of actions, and describe phase-type distributions in terms of their associated CTMC. In this way, we distinguish a *Markovian transition* from an *action transition*. The first one (Markovian transitions) are denoted by either $\langle r \rangle$ or $\stackrel{r}{\Rightarrow}$, indicating a temporal delay given by an exponential distribution with a rate r. The action transitions are denoted by either α or $\stackrel{\alpha}{\rightarrow}$; they indicate the instant occurrence of action α. Using this distinction, every sequential expression $(\alpha, PH(A)).P_{seq}^{fin}$ is translated into the following equivalent form in which $o = ord(A)$:

$$\begin{array}{rcl} Int_1 & = & \langle A(1,1)\rangle.Int_1 \oplus \langle A(1,2)\rangle.Int_2 \oplus \ldots \oplus \langle A(1,o)\rangle.Int_o \\ Int_2 & = & \langle A(2,1)\rangle.Int_1 \oplus \langle A(2,2)\rangle.Int_2 \oplus \ldots \oplus \langle A(2,o)\rangle.Int_o \\ \vdots & & \\ Int_{o-1} & = & \langle A(o-1,1)\rangle.Int_1 \oplus \langle A(o-1,2)\rangle.Int_2 \oplus \ldots \oplus \langle A(o-1,o)\rangle.Int_o \\ Int_o & = & \alpha.P_{seq}^{fin} \ . \end{array}$$

In this equivalent form, $\oplus$ indicates an internal choice among Markovian transitions; the behavior of this operator is the same as that of the choice operator for Markovian transitions in classical process calculi, such as PEPA.

As a result, $P_{seq}^{init} = (\alpha, PH(A)).P_{seq}^{fin}$ becomes $P_{seq}^{init} = Int_1$, while

$$P_{seq} = (\alpha_1, PH(A_1)).P_{seq}^1 + \ldots + (\alpha_n, PH(A_n)).P_{seq}^n \text{ becomes } P_{seq} = Int_1^1 + \cdots + Int_1^n.$$

The states $Int_1, \ldots, Int_o$ are correlated with the $CTMC(A)$ states, while the values $A(i,j)$ $(1 \leq i, j \leq o)$ are correlated with the transitions rates of $CTMC(A)$.

The operational semantics of PHASE is presented in Table 2. Each rule consists of some premises and a conclusion; the transition of the conclusion is valid whenever the transitions of the premises are valid. Because the operators $\oplus$, + and $\underset{\{L\}}{\S}$ are commutative, the rules from $CH1$ to $PAR5$ remain valid whenever we interchange $P1$ with $P2$.

The rule $SEQ1$ indicates the occurrence of actions (for action transitions), while $SEQ2$ indicates the time delay (for Markovian transitions). Rule $CH1$ indicates an internal choice for Markovian transitions; additionally, rule $CH2$ describes the race indicated by the choice operator between two phase-type distributions. Rule $CH3$ describes the race policy: while the quickest phase-type transition is executed, the others are discarded. The other rules deal with the parallel composition. $PAR1$ describes the situation when the processes do not interact. $PAR2$ treats the parallel composition between a Markovian transition and an action: since the involved action is not in the cooperation set $\{L\}$, its corresponding action transition is immediate (and precedes the Markovian transition). On the other hand, if the action is in $\{L\}$, then the process containing the action transition should wait until the other process executes a matching action transition; this is described by rule $PAR3$. Rule $PAR4$ says that in a synchronization, the transitions which are not part of the cooperation set L work independently, while rule $PAR5$ says that in a synchronization between action transitions, the matching transitions with actions in $\{L\}$ work simultaneously.

Since the PHASE semantics uses both action and Markovian transitions, it is possible to have a nondeterminism produced by the parallel operator (not caused by the choice operator). To illustrate such a nondeterminism, we use the process $P = (P_1 \underset{\emptyset}{\S} P_2) \underset{\{\alpha\}}{\S} P_3$, where $P_1 = (\alpha, PH(A_1)).P_1$, $P_2 = (\alpha, PH(A_2)).P_2$, and $P_3 = (\alpha, PH(A_3)).P_3$.

Table 2. Rules of the operational semantics for PHASE.

$$(SEQ1)\ \frac{}{\alpha.P \xrightarrow{\alpha} P} \qquad (SEQ2)\ \frac{}{\langle r \rangle.P \overset{r}{\Rightarrow} P} \qquad (CH1)\ \frac{P_1 \overset{r_1}{\Rightarrow} Q_1}{P_1 \oplus P_2 \overset{r_1}{\Rightarrow} Q_1}$$

$$(CH2)\ \frac{P_1 \overset{r_1}{\Rightarrow} Q_1 \qquad P_2 \overset{r_2}{\Rightarrow} Q_2}{P_1 + P_2 \overset{r_1}{\Rightarrow} Q_1 + P_2} \qquad (CH3)\ \frac{P_1 \xrightarrow{\alpha_1} Q_1 \qquad P_2 \overset{r_2}{\Rightarrow} Q_2}{P_1 + P_2 \xrightarrow{\alpha_1} Q_1}$$

$$(PAR1)\ \frac{P_1 \overset{r_1}{\Rightarrow} Q_1 \qquad P_2 \overset{r_2}{\Rightarrow} Q_2}{P_1 \underset{\{L\}}{\S} P_2 \overset{r_1}{\Rightarrow} Q_1 \underset{\{L\}}{\S} P_2}$$

$$(PAR2)\ \frac{P_1 \xrightarrow{\alpha_1} Q_1 \qquad P_2 \overset{r_2}{\Rightarrow} Q_2}{P_1 \underset{\{L\}}{\S} P_2 \xrightarrow{\alpha_1} Q_1 \underset{\{L\}}{\S} P_2}\ (\alpha_1 \notin \{L\})$$

$$(PAR3)\ \frac{P_1 \xrightarrow{\alpha_1} Q_1 \qquad P_2 \overset{r_2}{\Rightarrow} Q_2}{P_1 \underset{\{L\}}{\S} P_2 \overset{r_2}{\Rightarrow} P_1 \underset{\{L\}}{\S} Q_2}\ (\alpha_1 \in \{L\})$$

$$(PAR4)\ \frac{P_1 \xrightarrow{\alpha_1} Q_1 \qquad P_2 \xrightarrow{\alpha_2} Q_2}{P_1 \underset{\{L\}}{\S} P_2 \xrightarrow{\alpha_1} Q_1 \underset{\{L\}}{\S} P_2}\ (\alpha_1 \notin \{L\})$$

$$(PAR5)\ \frac{P_1 \xrightarrow{\alpha} Q_1 \qquad P_2 \xrightarrow{\alpha} Q_2}{P_1 \underset{\{L\}}{\S} P_2 \xrightarrow{\alpha} Q_1 \underset{\{L\}}{\S} Q_2}\ (\alpha \in \{L\})$$

Whenever the duration of transitions $tr_1 = (\alpha, PH(A_1))$ and $tr_2 = (\alpha, PH(A_2))$ are shorter than the duration of transition $tr_3 = (\alpha, PH(A_3))$, the transition tr_3 can synchronize with only one of the transitions tr_1 or tr_2, namely with the one which is available for cooperation. We should resolve all the instances of action nondeterminism in order to derive the performance measures for PHASE processes. For this, the competing transitions are assumed to be equally probable to be chosen, namely the winning action transition is obtained from a uniform distribution of all competitors. In the case of our P, the assumption that the competing transitions are equally probable to be chosen means that both tr_1 and tr_2 are selected for synchronization with probability 0.5. Note that there exist alternative solutions for dealing with such a non-determinism: for instance, employing priority levels and weights [10], or using schedulers.

When comparing our process calculus PHASE to the process calculus PEPA employing exponential distributions [6], a possible link is provided by the intermediate states and transitions used when defining the PHASE transitions and operators. On the other hand, when reasoning about the behavior of PHASE processes, these internal states and transitions could be ignored (viewed as technical details). Thus, when referring to the transitions and states of a sequential PHASE process P, we have in mind only transitions of the form $(\alpha, PH(A))$ and the states to which these transitions are connected (in P). The states reached by P are given by the set $ds(P)$, and the transitions appearing between the states of $ds(P)$ are given by the multiset $trm(P)$.

Examining the probabilistic model checker PRISM [5], a process P specified in PHASE can be translated into (the language of) PRISM by following the next steps:

1. Describe P in a form compatible with the language of PRISM;
2. Generate the states and transitions of P (ignoring the time of transitions and internal parallelism);
3. Implement the sequential and choice operators;
4. Implement the parallel operator.

The detailed presentation for each step is given below.

3.1. Step 1: Expressing PHASE Processes in a Form Closer to PRISM

PRISM is limited to sequential processes composed in parallel. This requires to describe a process P of PHASE in a form denoted by P_B consisting of sequential processes, parallel operators and parentheses only. This form is obtained by decomposing the subcomponents of a PHASE parallel process P_{par}; such a decomposition is realized recursively up to the moment when the sub-components are sequential. As a result, we get the multiset $SP = \{P_1, \ldots, P_m\}$ of sequential PHASE processes defining P_B, as well as the multiset of transitions $TR(P_B) = \biguplus_{1 \le i \le m} trm(P_i)$ executed by these sequential processes.

3.2. Step 2: Generating the States and Transitions in PRISM

This step is dealing with the states reached by the sequential processes of P_B and with the transitions between these states. Looking at PRISM, the states of a process are described by using additional local variables assigned to the process; in this way, each state is mapped to a valuation over these variables. The local variables should be part of a module, indicating also that they belong to a specific process. Therefore, we provide a PRISM name to each process $P_i \in SP$ by using an injective function $f : SP \to ID_{PRISM}$, where ID_{PRISM} is the set of valid PRISM names. We represent each P_i by using a single variable; this fact (having only a variable for each process) simplifies the behaviour of P_i by identifying 'module' with 'variable'. Consequently, the function f provides the name of both the variable corresponding to P_i and that of the module in which this variable is used. In this way, the possible states of P_i can be presented as numerical values for these variable by defining an injective function $g_i : ds(P_i) \to \mathbb{N}$. Thus, it is obtained the following module whose initial state is exactly P_i:

module $f(P_i)$
$f(P_i) : \left[0..\max_{S \in ds(P_i)}(g_i(S))\right]$ init $g_i(P_i)$;
endmodule .

In this way, the sequential processes in PHASE are expressed as PRISM modules by using their states: if a process P_i is in state S, then module $f(P_i)$ (using the variable $f(P_i)$) is in state $g_i(S)$. Regarding the transitions performed by P_i during its evolution, we give a PRISM name to the transitions in $TR(P_B)$ in a similar way (as for states) by using an injective function $h : TR(P_B) \to ID_{PRISM}$. Thus, every transition tr is presented as $(\alpha, PH(A))$ from state S_0 to state S_1 of P_i as follows:

$$[\alpha]\ (f(P_i) = g_i(S_0))\ \&\ (h(tr)\_\text{won=true}) \mathrel{-}> 1 : (f(P_i)' = g_i(S_1))\ .$$

This means that whenever P_i is in a state S_0 (i.e., $f(P_i) = g_i(S_0)$) and the race for execution is won by the transition tr (namely $h(tr)$_won=true), this transition is activated and action α is executed. Thus, P_i is moving to state S_1 (i.e., $f(P_i)' = g_i(S_1)$), and transition tr is included in module $f(P_i)$.

As a result of this step, the states and the transitions of the sequential processes P_i in PHASE are translated into PRISM. However, the PHASE transition $tr = (\alpha, PH(A))$ whose delay time is phase-type distributed (according to $PH(A)$) is translated into PRISM as a transition $tr' = (\alpha, 1)$ whose delay time is exponentially distributed. A solution to this problem is presented in the next step. Moreover, the phase-type distributions determining the transitions duration and the interactions between the sequential processes in P_B are treated in Steps 3 and 4. Thus, these steps complete the description of P_B.

3.3. Step 3: Implementing the Choice and Sequential Operators

It is required to split phase-type transitions into Markovian and action transitions (as indicated by the semantics of PHASE) to be able to tackle the behaviour of the sequential and choice operators. In order to implement this in PRISM, a module is created for any PHASE transition tr. Such a module becomes active whenever P_i enters in state S_0,

and its stochastic behaviour matches the Markov chain $CTMC(A)$ (transition by transition). However, since PRISM does not admit a dynamic creation of new modules (namely, all the modules should be specified for the initial system), it is not possible to create new modules when a transition tr is activated (and discard it after performing). Alternatively, a single module for tr is defined, reusing it whenever it is required. On the other hand, we should be sure that P_i performs the action α after (the module corresponding to) tr reaches the absorbing state (i.e., after the delay time associated with tr has expired). Considering $o = ord(A)$, the obtained module is:

module $h(tr)$
$h(tr) : [0..o]$ init 1;
$h(tr)$_won : bool init false;
...
$[\,] (f(P_i) = g_i(S_0))$ & $(h(tr) = j)$ -> $A(j,1) : (h(tr)' = 1) + \ldots +$
$+A(j,j-1) : (h(tr)' = j-1) + A(j,j+1) : (h(tr)' = j+1) + \ldots +$
$+A(j,o-1) : (h(tr)' = o-1) + A(j,o) : (h(tr)' = o)$ & ($h(tr)$_won′=true);
...
endmodule .

In this module, we have $1 \leq j \leq o-1$, and + describes a choice between exponentially distributed transitions (in PRISM) representing the precise equivalent of the PHASE operator $\oplus$ (as mentioned, PRISM handles only exponentially distributed transitions). The values taken by the variable $h(tr)$ provide the states of module $h(tr)$; these values (excepting 0) correspond exactly to the states of $CTMC(A)$. To be accurate, we omit the self loops from $CTMC(A)$; however, since self loops have no stochastic effect, they can be ignored. Additionally, the variable $h(tr)$_won is considered in order to manage properly the status of transition tr: if $h(tr)$_won=false, the process P_i is not in state S_0, and so module $h(tr)$ is inactive; if $h(tr)$_won=true, the delay time has elapsed and process P_i should perform action α.

However, this translation is not complete because it does not treat the following points:

- The module $h(tr)$ becomes inactive after the process P_i leaves state S_0 (inactive means that no transition is enabled). This module can reactivate itself later if the guard of at least one of its transitions is satisfied; if not, the module keeps its current state (given by the values of the local variables). However, when process P_i re-enters state S_0 (later), the module $h(tr)$ is not reset to its initial state, leading to an incorrect behavior.
- The phase-type transitions of a sequential process occurs usually in choice expressions; accordingly, the module $h(tr)$ should consider the context of transition tr.

In order to get a correct PRISM implementation of the sequential and choice operators, it is worth noting that a choice applied to a single transition is equivalent to the sequential operator; this means that we may analyze only the choice operator (avoiding the sequential operator). To implement properly the choice operator to ensure that (phase-type) transitions are reset correctly, there are required some changes of the modules capturing the duration of these transitions. First, we define the state S_0 of P_i by using the choice operator: $S_0 = (\alpha_1, PH(A_1)).S_1 + \ldots + (\alpha_n, PH(A_n)).S_n$. Then, let us assume that $tr_k = (\alpha_k, PH(A_k))$ and $o_k = ord(A_k)$ for $1 \leq k \leq n$. In these circumstances, we impose the rule that a module $h(tr_k)$ is active only when none of the phase-type transitions from S_0 won the race for execution (according to the race condition for choice). For this, we define a Boolean property $f(P_i)_g_i(S_0)$_race_on described in PRISM by:

formula $f(P_i)_g_i(S_0)$_race_on = ($h(tr_1)$_won=false) & ... & ($h(tr_n)$_won=false) .

This property is true when the race between the PHASE transitions tr_k ($1 \leq k \leq n$) is still going on (and false otherwise). This property is added to the guards for all transitions of the module $h(tr_k)$; the old PRISM guard

$$(f(P_i) = g_i(S_0)) \text{ \& } (h(tr_k) = j)$$

is replaced by the new guard

$$(f(P_i) = g_i(S_0)) \ \& \ (h(tr_k) = j) \ \& \ f(P_i)\_g_i(S_0)\_\text{race\_on}.$$

Thus, the modules for transitions involved in such a choice operation are inactivated immediately the choice outcome is decided. The reset of these modules after the race for execution is completed can be performed at any moment after the end of the race until P_i returns to state S_0. The reset is applied as soon as the choice is settled. This seems to be a natural approach, also easy to implement: in module $h(tr_k)$, we replace the updates $(h(tr_k)' = o_k)$ with the update $h(tr_1)' = 1) \ \& \ \ldots \ \& \ (h(tr_n)' = 1)$. Note that both expressions $h(tr_k)' = o_k$ and $h(tr_k)$_won′=true accomplish the same objective: to indicate that transition tr_k won the race.

Since in PRISM local variables are read globally but modified locally, the modules can use but not change the values of the local variables defined in other modules. This means that the above update does not work properly (as desired). It is necessary to turn the local variable $h(tr_k)$ into a global one, such that the competitors of tr_k can updated it whenever they win the race. For this, we remove the declaration

$$h(tr_k) : [0..o_k] \text{ init } 1;$$

from the module $h(tr_k)$, and declare as global the variable $h(tr_k)$ in any other module by

$$\text{global } h(tr_k) : [0..o_k] \text{ init } 1 \ .$$

Unlike the (defined locally) functions $g_1, \ldots, g_m$, the function h is defined globally (note that naming conflicts could appear whenever local functions $h_1, \ldots, h_m$ would be used instead of h). In this way, whenever the transition tr_k wins the race for execution, the modules corresponding to the losing transitions are blocked and reset (as desired). Moreover, whenever the process P_i (re)enters the state S_0, the modules $h(tr_1), \ldots, h(tr_n)$ remain inactive due to the fact that $f(P_i)_g_i(S_0)$_race_on is false and because $h(tr)$_won is true. To avoid such a faulty behavior, we reset the variable $h(tr_k)$_won to false at any time when the transition tr_k is performed; this makes sure that $f(P_i)_g_i(S_0)$_race_on is true. To implement this, we include in module $h(tr_k)$ the following transition:

$$[\alpha_k] \ (f(P_i) = g_i(S_0)) \ \& \ (h(tr_k)\text{\_won=true}) {-}{>} \ 1 : (h(tr_k)\text{\_won'=false}) \ .$$

Thus, $h(tr)$_won becomes false once P_i finishes transition tr_k; this is obtained by synchronizing module $h(tr_k)$ with module $f(P_i)$ over their common action α_k (as presented in Step 4).

3.4. Step 4: Implementing the Parallel Operator

This final step translates the interactions between the sequential (PHASE) processes from SP such that the action transitions to be indeed immediate; now the action transitions are represented by exponentially distributed (PRISM) transitions with rates equal to 1. However, PRISM does not support immediate transitions; then, we introduce a module named inst_sync such that for any action α which can be performed by one of the processes P_i $(1 \leq i \leq m)$, the module inst_sync contains a transition of the form $[\alpha]$ true $-> infty$: true; where $infty$ is a large number (e.g., a value at least a few orders of magnitude larger than any value associated with the transitions from $TR(P_B)$).

Due to the multiplicative law for computing synchronization rates in PRISM, synchronization of module inst_sync with other modules pushes the immediacy of actions. More precisely, all the transitions of form $[\alpha] \ldots -> \ldots$ executed by either any module $f(P_i)$ or any interaction between the modules of SP have a rate of $1 \cdot \ldots \cdot 1 \cdot infty = infty$—this means that their duration is equal to $1/infty \approx 0$.

The parallel operator is translated into PRISM in a rather simple way, as a consequence of the separation between transition durations and occurrence of actions, and that actions are performed instantaneously. Considering together all the modules, in order to obtain fully the translation of P in PRISM, we use the parallel operators available in PRISM (having

the same waiting condition like the parallel operator in PHASE). Consequently, we replace each process P_i from P_B with the following parallel composition of modules:

$$f(P_i) \;||\; \left(\underset{tr \in trm(P_i)}{|||} h(tr) \right).$$

For indicating that modules $h(tr)$ interact by means of shared variables (not actions), we use the operator $|||$ expressing a parallel composition without any synchronization over actions. On the other hand, the operator $||$ denotes a parallel composition over the common actions of the involved participants; this operator is involved in both assuring the proper reset of the modules $h(tr)$ and separating the duration of transitions from the actions occurrence. Furthermore, all the instances of the $\underset{\{L\}}{\S}$ operator are replaced by its counterpart $||[L]||$ in PRISM, denoting the resulting PRISM expression by P'_B. Similar to the PHASE parallel operator using $\underset{\{L\}}{\S}$, the parallel operator using $||[L]||$ describes the synchronization over the actions in $\{L\}$. Therefore, the PRISM expression P'_B is connected to the module inst_sync, resulting $P' = P'_B \;||$ inst_sync. The module inst_sync ensures that the actions in P' are immediate. It is worth noting that immediate actions are not permitted in PRISM; therefore, the duration of a PHASE transition $tr = (\alpha, PH(A))$ is distributed according not to $PH(A)$, but to the convolution of $PH(A)$ (namely the delay associated with tr) and $Exp(infty)$ for producing action α as 'immediate' action (the error introduced by $Exp(infty)$ can be reduced to an imperceptible level by selecting an appropriate large value for $infty$).

Finally, we notify the PRISM model checker about the completion of this translation by using the construct `system` P' `endsystem`. To indicate that this construct describes a CTMC, the keyword `ctmc` is placed at the beginning of P'. In this way, the PRISM implementation of the PHASE process P is finished.

4. Software Tools for Phase-Type Distributions

Distribution fitting is the procedure of selecting a certain distribution that best fits to a set of data generated by some random process. Distribution fitting also checks if the distribution of a sample of data differs notably from a theoretical distribution. Once the distribution type is identified and the parameters to be estimated are fixed, a best-fit distribution is usually defined as one with the maximum likelihood parameters. In this section, we review four fitting tools for phase-type distributions, namely EMpht, jPhase, HyperStar and PhFit.

The tool EMpht [11] is one of the oldest software program for fitting PH distributions. It consists of a C program which implements the parameter estimation algorithm [12], and a MATLAB program which plots the resulting PH approximations. EMpht accepts as input both discrete samples and continuous distributions; the distributions are either prespecified (i.e., uniform, normal, log-normal, Weibull, Wald, or PH) or user-defined. The fitting procedure is based on expectation-maximization (EM), and the user must choose the type of PH distribution fitting the input data (namely hyperexponential, general hypoexponential, Coxian, general Coxian, etc), or user-defined. The last feature is particularly powerful, as it allows the user to place constraints on the structure of the underlying CTMC in terms of the initial probability vector (e.g., the only two possible initial states are 1 and 3), and the infinitesimal generator matrix (e.g., there is no direct transition between states 2 and 5). An interesting additional option provided by this tool is its support for right-censored and interval-censored sample data (censoring is a condition in which the value of an observation is only partially known; in right censoring, a data point is above a certain value but it is unknown by how much, while in interval censoring, a data point is somewhere on an interval between two values).

The tool jPhase [13] contains three Java packages for operating with PH distributions, namely jPhase (for representing the distributions), jPhaseGenerator (for drawing random variates from the distributions), and jPhaseFit (for fitting distributions). The fitting module offers several algorithms, some of them relying on expectation–maximization, while the

other employing moment matching (MM) techniques. The expectation–maximization procedures accept sets of samples as input, and output PH distributions which are either general [12], hyperexponential [14], or hyper-Erlang [15]. In contrast, the moment matching procedures require only the first three moments of the distribution to be fitted, and return PH distributions whose first moments (approximately) match those given as input. Furthermore, as output the user can choose between 2-phase [16] and n-phase representations [17], depending on the desired goodness of fit for the moments. In addition to fitting and simulation capabilities, jPhase also supports basic operations (convolution, minimum, maximum, convex mixture) over PH representations, and can compute both the probability density functions (PDF) and cumulative distribution functions (CDF) of PH distributions (technically, a probability density function is the derivative of a cumulative distribution function).

The tool HyperStar [18] is a recent Java application for PH fitting focusing on being user-friendly. Unlike EMpht and jPhase, Hyperstar follows a cluster-based approach to generating PH distributions [19]: after supplying sample data as input, the user selects the regions of the empirical PDF that must be fit particularly well, and the algorithm produces a hyper-Erlang distribution according to the instructions of the user. More specifically, this procedure divides the sample into clusters, centred around the points indicated by the user, and then fits each cluster with an Erlang distribution. Besides its accessibility, another attractive feature of HyperStar is that it has an Expert mode in which the user can customize the main elements of the fitting algorithm (e.g., whether the detection of the important regions of the empirical PDF should be done manually or automatically). The tool also comes with an optional Mathematica interface which allows the user to develop her/his own fitting procedures, while benefiting from HyperStar's intuitive and informative visual user interface.

The tool PhFit [20] is a Java application for fitting PH distributions which distinguishes itself in terms of flexibility. Firstly, the input provided by the user can be either a sample, a prespecified distribution (Weibull, log-normal, uniform, Pareto-like), or a user-defined distribution. Second, the tool employs a segmentation-based approach for building PH representations which involves splitting the distribution to be fit into a body section and a tail section. The body and the tail fit then by using an acyclic PH (ACPH) and a hyperexponential distribution, respectively. Thirdly, the user has the possibility of selecting the error measure to be minimized during the fitting process (it can be either the CDF/PDF area difference between the input and the output distribution, or the cross entropy between the aforementioned distributions).

To summarize, the main characteristics of the tools are in Table 3 below:

Table 3. Characteristics of the fitting tools for phase-type distributions.

Software Tool	**EMpht [11]**	**jPhase [13]**	**HyperStar [18]**	**PhFit [20]**
Tool type	C application	Java library	Java application	Java application
Input format	sample, PDF	sample, moments	sample	sample, CDF
Fitting method	EM	EM, MM	cluster-based	nonlinear optimization
PH distributions	any	any	hyper-Erlang	ACPH-hyper mixture
Visual interface	no	yes	yes	yes

It is difficult to declare a winner when it comes to pick the best tool for obtaining phase-type approximations. Each of the tools has its own unique advantages and weaknesses. EMpht is very easy to use and offers many options that are valuable for advanced modeling, but it lacks a proper visual interface. jPhase is remarkably comprehensive (it implements six different fitting techniques), yet it does not support user-defined constraints over the structure of the resulting PH representations. HyperStar provides a delicate balance between accessibility and the degree of user control over the fitting procedure; however, its focus on hyper-Erlang distributions is somewhat restrictive and can lead to large representations. Finally, PhFit may not be as complex as its competitors, but its segmentation-based mechanism is especially suitable for handling ill-behaved, heavy-tailed distributions.

5. An Example of Analyzing a Non-Markovian Approximation

An example is presented to illustrate our approach of approximating and analyzing a non-Markovian system by means of process calculus PHASE and model checker PRISM.

We consider the non-Markovian systems presented in Figure 1. Using our process calculus PHASE, we can put these two systems together by synchronizing them using their common action a. We denote by $P1_n$ the PHASE process corresponding to the non-Markovian system located on the left-hand side of Figure 1 in state n (n could be either 1, 2 or 3); similarly, by $P2_1$ and $P2_2$ are denoted the right-hand side non-Markovian system in state 1 and 2, respectively. Thus, by approximating the distributions of the five transitions with some phase-type distributions denoted by $PH(A_m)$, the new non-Markovian system obtained by synchronizing the initial systems of Figure 1 can be described in PHASE by a process $Sys = P1_1 \underset{\{a\}}{\S} P2_1$, with

$$\begin{aligned} P1_1 &= (d, PH(A_1)).P1_2 \\ P1_2 &= (c, PH(A_2)).P1_3 + (a, PH(A_3)).P1_3 \\ P2_1 &= (a, PH(A_4)).P2_2 + (b, PH(A_5)).P2_2\ , \end{aligned}$$

where $PH(A_1) \approx Norm(4,1)$, $PH(A_2) \approx Exp(4)$, $PH(A_3) \approx Lnorm(0,0.5)$, $PH(A_4) \approx Unif(1.5,2.5)$, and $PH(A_5) \approx Weibull(1.5,1)$.

As discussed in Section 4, there exist various software tools able to generate the involved distributions $PH(A_1)$ ... $PH(A_5)$. Based on the tools review presented in the previous section, our option is for EMpht [12]. This option is motivated by the fact that EMpht allows to have structural constraints over the PH representations (namely, to have only one initial state) and because it provides the fitting algorithm for the distributions appearing in our example. Reasonably large PH representations are used in order to get accurate approximations: 1 phase ($PH(A_2)$), 10 phases ($PH(A_3)$ and $PH(A_5)$), 15 phases ($PH(A_1)$), and 20 phases ($PH(A_4)$). Only for $PH(A_2)$, since it models an exponential distribution, we simply build the matrix A_2 by hand instead of relying on EMpht. The match produced by EMpht between these PH distributions and the initial distributions is good. It is worth noting the special case of uniform distribution, approximated by a (normal-like) Erlang distribution.

We have now all the necessary ingredients to implement our PHASE process Sys in PRISM. Essentially, the PHASE process is translated into a PRISM one by following the steps previously presented.

Since the model Sys is expressed in a form compatible with PRISM (as required by Step 1), we can move to Step 2. Thus, we define the functions f, g_1, g_2 and h in Table 4.

Table 4. The definition for functions f, g_1, g_2 and h.

Function	f		g_1			g_2				
Argument	$P1$	$P2$	$P1_1$	$P1_2$	$P1_3$	$P2_1$	$P2_2$			
Value	P1	P2	1	2	3	1	2			
Function	h									
Argument	$(d, PH(A_1))$		$(c, PH(A_2))$			$(a, PH(A_3))$		$(a, PH(A_4))$	$(b, PH(A_5))$	
Value	tr1		tr2			tr3		tr4	tr5	

Based on these functions, we present the modules corresponding to $P1$ and $P2$:

```
module P1
P1 : [1..3] init 1;
[d] (P1=1)&(tr1_won=true) -> 1 : (P1'=2);
[c] (P1=2)&(tr2_won=true) -> 1 : (P1'=3);
[a] (P1=2)&(tr3_won=true) -> 1 : (P1'=3);
endmodule
```

```
module P2
P2 : [1..2] init 1;
[a] (P2=1)&(tr4_won=true) -> 1 : (P2'=2);
[b] (P2=1)&(tr5_won=true) -> 1 : (P2'=2);
endmodule .
```

We proceed to Step 3 and create a separate module for each phase-type transition in $P1$ and $P2$, together with the global variables and formulae associated with each module:

```
global tr1 : [0..15] init 1;
global tr2 : [0..1] init 1;
global tr3 : [0..10] init 1;
global tr4 : [0..20] init 1;
global tr5 : [0..10] init 1;

formula P1_1_race_on = (tr1_won=false);
formula P1_2_race_on = (tr2_won=false)&(tr3_won=false);
formula P2_1_race_on = (tr4_won=false)&(tr5_won=false);

module tr1
tr1_won : bool init false;

[] (P1=1)&(tr1=1)&P1_1_race_on -> ...;
  . . .
[] (P1=1)&(tr1=15)&P1_1_race_on -> ...;

[d] (P1=1)&(tr1_won=true) -> 1 : (tr1_won'=false);
endmodule
  . . .
module tr5
tr5_won : bool init false;

[] (P2=1)&(tr5=1)&P2_1_race_on -> ...;
  . . .
[] (P2=1)&(tr5=10)&P2_1_race_on -> ...;

[b] (P2=1)&(tr5_won=true) -> 1 : (tr5_won'=false);
endmodule .
```

Having the modules that implement the structure and the transitions of $P1$ and $P2$, we go now to Step 4. We begin by building the module inst_sync, which is essential in forcing the immediacy of actions:

```
module inst_sync
[a] true -> 10000 : true;
[b] true -> 10000 : true;
[c] true -> 10000 : true;
[d] true -> 10000 : true;
endmodule .
```

Combining all the modules defined so far, we get the following PRISM expression for *Sys*:

```
system
((P1 || (tr1 ||| tr2 ||| tr3)) |[a]| (P2 || (tr4 ||| tr5))) || inst_sync
endsystem .
```

The keyword `ctmc` is inserted at the beginning of the PRISM model, and so the PRISM implementation of the PHASE description of Sys is ready to be executed in PRISM.

In the following paragraphs, we denote by Sys_{NM} the initial non-Markovian system presented in Figure 1. To analyze the behavior of the PRISM implementation of Sys, and see how this behavior matches that of Sys_{NM}, we compare the results produced by PRISM for Sys with those of a simulation for Sys_{NM} obtained by using an implementation of Sys_{NM} in R. The simulation is based on the generation of 1 million traces for Sys_{NM}, recording for each trace the moments at which the events took place.

Figure 2 presents the distribution of the time (the unit of time is arbitrary) elapsed until the systems terminate. Regarding Sys, this means the time taken by processes $P1$ and $P2$ to reach their final states (when no deadlock appears as a consequence of a failed synchronization over a). We can conclude that the distributions for Sys_{NM} and Sys match well, being close each other (without being identical).

Regarding the differences between the distributions for Sys_{NM} and Sys, our opinion is that they are caused by a deficient approximation of the duration for transition $tr4$ rather than some deficiencies in translating the PHASE process into PRISM. Regarding the deadlocks caused by the synchronization over a of $P1$ and $P2$ (namely the situations when $tr4$ became enabled and $tr3$ did not, or vice-versa), the percentage of these deadlocks is given by a value of 0.105986 for Sys_{NM} and a value of 0.118744 for Sys. Moreover, the probability of successful synchronization over a is calculated; it was indicated a value of 0.003007 for Sys_{NM} and a value of 0.003618 for Sys. The accuracy of such an analysis differs in general between a simple approximation (involving only PRISM) and a more elaborate one (involving PHASE and PRISM).

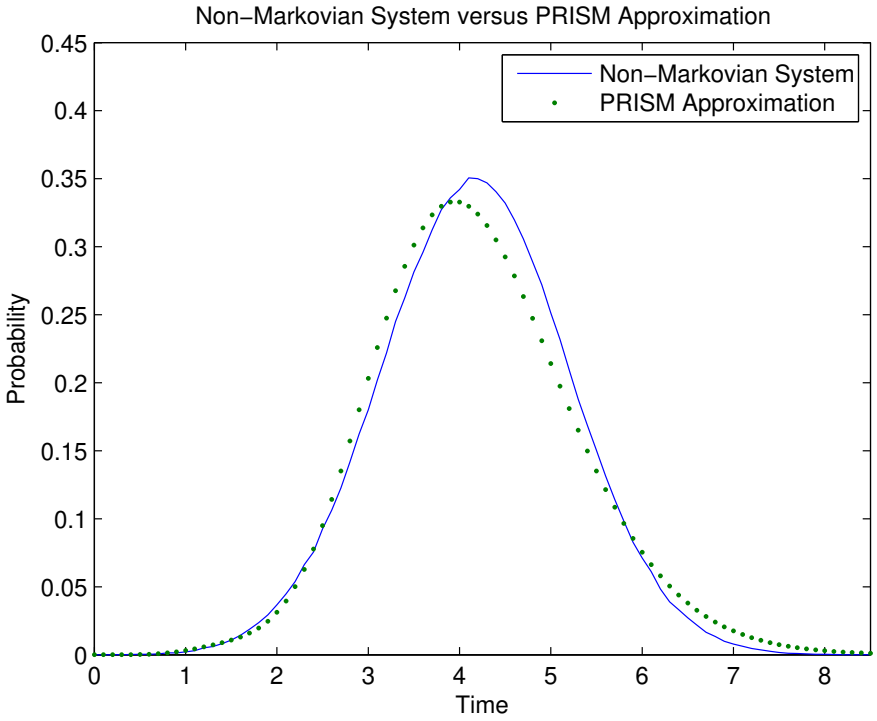

Figure 2. The matching of the distributions of Sys_{NM} and Sys.

Considering all these aspects, the performance measures for Sys_{NM} and Sys demonstrate the accuracy and precision of our approximation for non-Markovian systems by using a stochastic process calculus employing phase-type distributions and its faithful implementation in the probabilistic model checker PRISM.

In terms of system modeling, the results support the validity of the translation of a PHASE model to a PRISM model, with the goal of using advanced software tools for analyzing non-Markovian systems in a satisfactory way.

6. Conclusions

Stochastic approaches are widespread and popular in science these days; they develop various methods to describe the behavior and evolution of complex systems in many fields of engineering and science. In particular, in electromagnetics, viscoelasticity, fluid mechanics, electrochemistry, biological population models, optics and signal processing [21]. Most of these applications involve non-Markovian systems. For instance, fractional Brownian motion is a non-Markovian process generalizing the standard Brownian motion [22]; functionals of such a process are important in practical applications of non-equilibrium dynamics, and plays an important role in stochastic dynamical systems exhibiting a long range dependence between states of the system. Stochastic processes can be used to model the behavior of non-Markovian systems from a pure statistical point of view (in statistical physics, for instance). Stochastic approaches enhanced already the existing models to interpret better some physical phenomena [23]. For instance, spread of an infectious disease is highly connected with non-Markovian dynamics; the non-Markovian systems play an important role in the complex dynamics of the infectious diseases. The simulations and analysis of the epidemiological models related to the diseases transmission dynamics are important to control the transmission rate in order to either eradicate them or to increase the treatment rate.

Additionally, the stochastic approach can be used for the direct computation of quantitative observables in models of the dynamics for complex molecular systems used for the prediction of thermodynamic and kinetic properties of experimental interest [24]. Regarding protein-folding kinetics, non-Markovian models not only simulate the molecular dynamics accurately but also emphasize the anomalies in accelerated protein kinetics [25].

The non-Markovian quantum dynamics of open systems reveal quantum properties related to quantum coherence, correlations and entanglement [26]. Quantum correlations, quantum coherences and other non-Markovian quantum processes are important for improving several protocols within communication, teleportation, cryptography, metrology, and so providing the future progress in quantum technology. The open quantum systems dynamics reflect features of the environment which allows a new perspective for applications. A general approach to the construction of non-Markovian quantum theory is proposed in [27], and a class of solvable models of non-Markovian quantum dynamics is suggested (these models describe open quantum systems with general form of nonlocality in time).

Currently, analyzing the behaviors of non-Markovian systems represents a real challenge; a possible impediment is related to the almost complete lack of software tools. The approach presented in this article is to use a process calculus involving phase-type distributions, followed by a translation of this process calculus into a probabilistic model checker offering automated simulations and verification of several properties. The approach is based on the fact that phase-type distributions are appropriate for approximating (to an arbitrary degree of accuracy) the transition durations of non-Markovian systems [4]. There already exist some Markovian process calculi using phase-type distributions for transition durations (e.g., [7,28]); however, they face the lack of tool support. Few of these calculi (for instance, [7]) are compatible with only some subclasses of non-Markovian systems. Others (see [28]) do not use some patterns of interaction between processes (for instance, synchronizing over used-defined sets of shared actions).

Argued as a reasonable solution, we come up with a stochastic process calculus for describing easily non-Markovian systems by means of phase-type distributions, and then translating this process calculus into an existing model checker that allow us to analyze the translated system by using a rather complex and freely available software tool.

Stochastic process calculi have numerous features making them suitable for modeling complex systems. Their syntax is easy to use, and the number of operators is small. The intuitive nature of the operators facilitates the understanding of their semantics; it is not difficult to be translated into a visual graphlike representation in which the vertices are the states of the system, while the directed edges between the vertices are the transitions

between states. Moreover, since the process calculi were initially created for examining the behaviour of concurrent systems, they are compositional by design. More specifically, the parallel operator offers the user a straightforward manner of expressing the dynamics of a system in terms of both its components and the interactions that take place between them; this greatly reduces the complexity and duration of the modeling activity. Finally, they are equally effective in capturing complex behaviours, thus eliminating any need for employing different approaches (i.e., having one approach for the human element, another for the computational system, and yet another for the interplay between the previous two). Additionally, stochastic process calculi come equipped with a variety of powerful (quantitative) logics for investigating the temporal and probabilistic properties of model behaviour. There exist some complex software tools (freely available) which implement these logics, making the verification of quantitative properties feasible even for large models (i.e., having up to 10^{12} states and transitions).

Based on our achievements presented in [29,30], this article offers a general method of analyzing non-Markovian systems by using a stochastic process calculus employing phase-type distributions and a probabilistic model checker as a software tool. The whole approach is rather practical, intended to solve a challenging problem. It is not a theoretical one as those developed in [8,28], but rather a practical one proposing a way to easily analyze non-Markovian systems by using an advanced software tool (model checker PRISM). The choice of PRISM for translating the PHASE processes is based on its expressive power and the fact that PRISM can implement properly the PHASE calculus; moreover, PRISM is one of the most powerful and well-supported model checkers freely available. With respect to performance measures, PRISM allows the specification and verification of steady-state measures (i.e., the probability that the system is in state S in the long run), transient measures (i.e., the probability that the system is in state S at time t), and passage-time measures (i.e., the probability that the system reaches state S before time t), by using an extended version of Continuous Stochastic Logic (CSL) [31]. Moreover, PRISM includes features such as statistical model checking (i.e., a discrete event simulation in which a number of model executions are generated to perform an approximate verification of some CSL formulas) and interactive simulation (i.e., a user-guided simulation).

As an open problem related to our work, it would be useful to derive analytically the error bounds for performance measures computed over Markovian (i.e., phase-type and exponential) approximations of non-Markovian systems, as is done in statistical model checking (e.g., [32]). Such a theoretical contribution would be helpful in applying an incremental modeling approach, allowing the user to start with a basic PHASE model and then gradually refine the model (mainly the phase-type representations by increasing the number of phases in each representation), until the modeling user is finally satisfied with the error bounds for each performance measure of interest.

Funding: This research received no external funding.

Acknowledgments: Many thanks to Armand Rotaru for his notable contributions during our past collaboration.

References

1. Milner, R. *Communicating and Mobile Systems: The Pi-Calculus*; Cambridge University Press: Cambridge, UK, 1999.
2. Norris, J.R. *Markov Chains*; Cambridge University Press: Cambridge, UK, 1998.
3. Cox, D.R. A use of complex probabilities in the theory of stochastic processes. *Math. Proc. Camb. Philos. Soc.* **2008**, *51* 313–319. [CrossRef]
4. Nelson, R. *Probability, Stochastic Processes, and Queueing Theory*; Springer: New York, NY, USA, 1995.
5. Kwiatkowska, M.; Norman, G.; Parker, D. PRISM 4.0: Verification of probabilistic real-time systems. *Lect. Notes Comput. Sci.* **2011**, *6806*, 585–591.
6. Hillston, J. *A Compositional Approach to Performance Modelling*; Cambridge University Press: Cambridge, UK, 1996.
7. El-Rayes, A.; Kwiatkowska, M.; Norman, G. Solving infinite stochastic process algebra models through matrix-geometric methods. In Proceedings of the PAPM'99, Zaragoza, Spain, 6–10 September 1999; Prensas Universitarias de Zaragoza: Zaragoza, Spain, 1999; pp. 41–62.

8. Hermanns, H. *Interactive Markov Chains—The Quest for Quantified Quality*; Springer: Berlin, Germany, 2002.
9. Neuts, M.F. *Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach*; Dover Publications: Mineola, NY, USA, 1995.
10. Bernardo, M.; Gorrieri, R. Extended Markovian process algebra. *Lect. Notes Comput. Sci.* **1996**, *1119*, 315–330.
11. Olsson, M. The EMpht-Programme. Technical Report. Chalmers University of Technology. 1998. Available online: http://home.imf.au.dk/asmus/dl/EMusersguide.ps (accessed on 16 November 2022).
12. Asmussen, S.; Nerman, O.; Olsson, M. Fitting phase-type distributions via the EM algorithm. *Scand. J. Stat.* **1996**, *23*, 419–441.
13. Riaño, G.; Pérez, J.F. jPhase: An object-oriented tool for modeling phase-type distributions. In Proceedings of SMCTools 2006, Pisa, Italy, 10 October 2006; ACM: New York, NY, USA, 2006; Article Number 5.
14. Khayari, R.; Sadre, R.; Haverkort, B. Fitting world-wide web request traces with the EM-algorithm. *Perform. Eval.* **2003**, *52*, 175–191. [CrossRef]
15. Thümmler, A.; Buchholz, P.; Telek, M. A novel approach for phase-type fitting with the EM algorithm. *IEEE Trans. Dependable Secur. Comput.* **2006**, *3*, 245–258. [CrossRef]
16. Telek, M.; Heindl, A. Matching moments for acyclic discrete and continuous phase-type distributions of second order. *Int. J. Simul.* **2002**, *3*, 47–57.
17. Bobbio, A.; Horvath A.; Telek M. Matching three moments with minimal acyclic phase type distributions. *Stoch. Model.* **2005**, *21*, 303–326. [CrossRef]
18. Reinecke, P.; Krauss, T.; Wolter, K. Phase-type fitting using HyperStar. *Lect. Notes Comput. Sci.* **2013**, *8168*, 164–175.
19. Reinecke, P.; Krauss, T.; Wolter, K. Cluster-based fitting of phase-type distributions to empirical data. *Comput. Math. Appl.* **2012**, *64*, 3840–3851. [CrossRef]
20. Horvath, A.; Telek, M. PhFit: A general phase-type fitting tool. *Lect. Notes Comput. Sci.* **2002**, *2324*, 82–91.
21. Tarasov, V.E. *Fractional Dynamics: Applications of Fractional Calculus to Dynamics of Particles, Fields and Media*; Springer: New York, NY, USA, 2011.
22. Mandelbrot, B.B.; Van Ness, J.W. Fractional Brownian motions, fractional noises and applications. *SIAM Rev.* **1968**, *10*, 422–437. [CrossRef]
23. Ruiz-Castro, J.E.; Acal, C.; Aguilera, A.M.; Roldán, J.B. A complex model via phase-type distributions to study random telegraph noise in resistive memories. *Mathematics* **2021**, *9*, 390. [CrossRef]
24. Vroylandt, H.; Goudenege, L.; Monmarche, P.; Pietrucci, F.; Rotenberg, B. Likelihood-based non-Markovian models from molecular dynamics. *Proc. Natl. Acad. Sci. USA* **2022**, *119*, 13. [CrossRef]
25. Ayaz, C.; Tepper, L.; Brunig, F.; Kappler, J.; Daldrop, J.O.; Netz, R. Non-Markovian modeling of protein folding. *Proc. Natl. Acad. Sci. USA* **2021**, *118*, 31. [CrossRef]
26. Burgarth, D.; Facchi, P.; Ligabò, M.; Lonigro, D. Hidden non-Markovianity in open quantum systems. *Phys. Rev. A* **2021**, *103*, 012203. [CrossRef]
27. Tarasov, V.E. General Non-Markovian Quantum Dynamics. *Entropy* **2021**, *23*, 1006. [CrossRef]
28. Wolf, V. Equivalences on Phase Type Processes. Ph.D. Thesis, University of Mannheim, Mannheim, Germany, 2008.
29. Ciobanu, G.; Rotaru, A. PHASE: A stochastic formalism for phase-type distributions. *Lect. Notes Comput. Sci.* **2014**, *8829*, 91–106.
30. Ciobanu, G.; Rotaru, A. Phase-type approximations for non-Markovian systems: A case study. *Lect. Notes Comput. Sci.* **2014**, *8938*, 323–334.
31. Baier, C.; Katoen, J.-P.; Hermanns, H. Approximate symbolic model checking of continuous-time Markov chains. *Lect. Notes Comput. Sci.* **1999**, *1664*, 146–161.
32. Younes, H.; Simmons, R. Probabilistic verification of discrete event systems using acceptance sampling. *Lect. Notes Comput. Sci.* **2002**, *2404*, 223–235.