Published in IET Software
Received on 9th January 2009
Revised on 27th May 2009
doi: 10.1049/iet-sen.2009.0002

In Special Issue on Performance Engineering

ISSN 1751-8806

Transient calculations on process algebra derived Markov chains

A. Clark S. Gilmore

LFCS, University of Edinburgh, Edinburgh, UK
E-mail: a.d.clark@ed.ac.uk

Abstract: The process of obtaining transient measures from a Markov chain as implemented in the software, ipclib is described. The software accepts models written in PEPA, Bio-PEPA or as a Petri net. In the case of the process algebras, a rich query specification language particularly well suited for the derivation of passage-time quantiles is provided. Such measurements are obtained from the derived Markov chain through a process known as uniformisation. The authors detail how the process algebra and query specification language allow one to ensure that the passage-time calculation is valid and then the entire process through to the final calculation of the cumulative distribution and probability density functions of the passage in question. The authors also show a more generic transient measure for which the full probability distributions at specific times are required.

1 Introduction

When analysing a performance model of a system a common query is over the duration of a given passage within the model. In a client–server style model, this is often the response time, that is the time that a client must wait for their request to be satisfied by the server. In other models we are simply analysing the time taken between two events or circumstances. Where the model in question is a continuous time Markov chain (CTMC) or a representation from which a CTMC may be derived, the average passage duration may be computed from the steady-state probability distribution obtained by solving the CTMC.

In general CTMCs are well suited to the analysis of average behaviour, derivable from the steady-state probability distribution. This paper is concerned with transient analysis of models that compute measures that are not time independent. Essentially we wish to analyse the probability distribution of a model a given time after some event or from the initial configuration. Here we describe software support for the computation of passage-time quantiles computed via a technique known as uniformisation. A passage-time quantile is simply a point taken along the cumulative distribution function (cdf) of the passage in question. The cdf maps time (usually along the x-axis) against the probability of completing the passage at or before that time. This allows the modeller to answer such questions as: 'What is the probability that a request is responded to within 4 seconds?' It is also possible to plot the probability density function (pdf) where the cdf is the integral of the pdf. The pdf maps time against the probability density of completing the passage at exactly that time.

This paper describes the calculation of passage-time quantiles/densities from a model which may be translated into a CTMC. The technique used is known as uniformisation – though it is sometimes called randomisation. We show how uniformisation has been integrated into a generic software tool for analysing models written in higher-level formalisms. Specifically we are chiefly interested in models written in the stochastic process algebra PEPA [1] and its derivative biological modelling language Bio-PEPA [2].

1.1 Related work

Hydra [3] is another software tool used for extracting response-time profiles, in that case from Petri nets. The focus in Hydra is on analysing very large models, for response time this obviously depends on rate values involved (see Section 5.4) but it has been used to solved

models with state-space sizes of the order of 10^6 and even 10^7. One technique that is used is pre-compilation, the model is input by the user and the Hydra software generates a program in standard C to generate the state-space and solve the generator matrix. This program is then compiled and run. From the outset the ipc software has been linked with the Hydra software; the ipc software can generate a Hydra model description from a PEPA model (and hence also from a Bio-PEPA model) or ipc can read in Hydra model description files. Recently some of the techniques described in this paper have been implemented in the Hydra software, most notably the uniformiser now produces both the cdf and pdf of a passage at the same time and the absorbing state is used to terminate the uniformisation computation (see Section 5.1).

The Möbius [4] project is aimed at delivering a multi-paradigm modelling software tool. As such there is some similarity with the ipc software, although we are mostly concerned with process algebras and in particular PEPA whereas the Möbius software is not limited to process algebras. In addition, Möbius, has several analysis techniques including discrete-event simulation. We have discrete-event simulation and continuous simulation methods available in the PEPA Eclipse Plug-in [5], but not in the ipclib suite. The Möbius framework excels in the specification of reward structures over the input model. These allow the definition of instant-of-time or interval-of-time rewards. Our framework here does not provide a comparable reward specification language. Additionally working with Sanders of the Möbius project, Diener completed a masters thesis [6] comparing several uniformisation techniques.

The MRMC (Markov Reward Model Checker) [7] project has produced several results for transient analyses of Markov chains. Here the problem is stated as time-bounded reachability – and amounts to determining the probability of reaching a set of target states within a given time bound. In [8] the authors report on the efficient detection of steady state during transient analysis. When performing a transient analysis over a large time span – suppose the user has asked for the time series analysis for a given time bound – it may be that the model reaches steady state or an equilibrium before the end of the time bound. When this happens the software can stop calculating for further time points since they will all have the same probability distribution. However, for a passage-time analysis this is inappropriate since we modify the Markov chain to have an absorbing state such that we measure only the first passage and not subsequent passages. Although we focus on passage analysis our software can also be used for such transient analysis (see Section 6) and there such optimisation is certainly appropriate.

The Prism [9] model checker implements a uniformisation algorithm in order to model check transient properties of CTMCs. The software can be used to check properties written in Continuous Stochastic Logic (CSL) [10]. Our ipc software is capable of translating process algebra models into an input suitable for the Prism model checker, thus allowing the user familiar with CSL the opportunity to specify their queries in their known setting. By using ipc to generate the Prism model the users benefit from the advantages of compositional modelling and in particular with respect to the constraints imposed on passages discussed here in Sections 3 and 4.

Structure: The paper is organised as follows in Section 2 we first give a brief introduction to PEPA, the process algebra in which our example models are written. We then give two example models which we will use to illustrate the techniques implemented in our software. In Section 3 we detail the constraints imposed on the passage-time query in question by the use of the uniformisation algorithm. The constraints refer to the transitions within the Markov chain in relation to the specified passage. Then in Section 4 we explain how the passage is specified by the user where the user is concerned with the model at the level of the process algebra but the uniformisation routine expects a Markov chain and source and target sets of states. We demonstrate how the use of a compositional process algebra allows us to ensure that the query the user specifies is always translated into a valid passage-time query which does not violate any of the imposed constraints. In addition we discuss some optimisations that are made possible through the use of a process algebra to derive the underlying Markov chain. Section 5 provides a full description of the uniformisation technique in order to obtain passage-time quantiles from a Markov chain for a passage specified as a set of source and a set of target states and provides results for our first example. Our second example is then analysed in Section 6, which is concerned with the calculation of more general transient measures in which probability distributions at specified times are calculated. In particular, this allows the production of a time series analysis. We finish by detailing our implementation in Section 7 and concluding in Section 8.

2 Example models

We illustrate the tools and techniques in this paper by referring to two examples. We draw one example from the area of computer service analysis and the other from systems biology. We must first briefly introduce our favoured process algebra, PEPA.

2.1 PEPA

We use the popular process algebra PEPA to compositionally describe our models. A model described by a PEPA description has an underlying Markov chain representation, although the user is hidden from the details of the underlying states. Each defined sequential component is a description of a small stateful process and each such is combined using the cooperation combinator with the

doi: 10.1049/iet-sen.2009.0002

restriction that the two must synchronise over the specified action labels. This may mean that some states are unreachable so composition does not always increase the state space but in general the state-space size does increase rapidly. The PEPA language has the following combinators

$$P ::= (a, \lambda).P \mid P + P \mid P \underset{L}{\bowtie} P$$

The process term $(a, \lambda).P$ describes a process that may perform the action a at rate λ to become the process P. The rate may be a numerical rate or the special rate $\top$, which means that the operation is performed passively by this process that must be subsequently synchronised with over this activity. The other process involved in the synchronisation determines the rate of the activity. The process $P_1 + P_2$ depicts competitive choice between the processes P_1 and P_2. The operator $\underset{L}{\bowtie}$ is cooperation/synchronisation between two components over the given set of actions L. The special $\underset{*}{\bowtie}$ synchronisation specifies that the two components synchronise over all activities that both may perform.

A model is represented by a series of definitions that describe the sequential behaviour of named components. These named components are then combined together in a main system equation, which represents the interaction between the various components in a model. Full details of the PEPA stochastic process algebra can be found in [1].

2.2 Layer-7 two-way architecture

Fig. 1 depicts a layer-7 two-way service architecture. A layer-7 two-way architecture allows the service to provide a common front to several different kinds of request. Each request is sent to an administrator who analyses the request and then dispatches it to be serviced by the appropriate background server. Each background server awaits such specific requests and returns them to the administrator who forwards the response to the appropriate client who invoked the original request. Fig. 2 shows the PEPA model description of a particular layer-7 two-way architecture configuration. By analysing such a system we can analyse the response times of different kinds of requests as well as in general. We could also modify the configuration of the service to enable us to decide, which will perform better under different conditions. For example, we can vary the rates of requests or the capacity of each kind of server, which in reality would result from server re-deployments or upgrades. We will see the passage-time analysis of this model in Section 5.2. In the next section we discuss how this process algebra model allows us to derive a Markov chain in a form suitable for the calculation of passage-time quantiles measuring the time from the client's request to their received response.

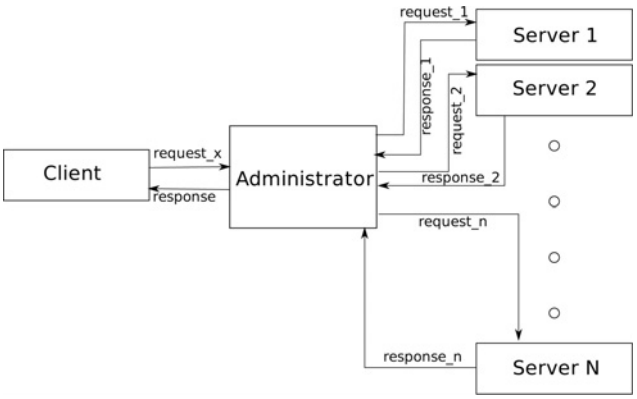

Figure 1 *Structure of a layer-7 two-way architecture*

2.3 Michaelis–Menten

We show here the PEPA model of a chemical reaction between an enzyme E and a substrate S to produce an intermediate compound $E{:}S$ and finally a product P. The reactions are modelled with Michaelis–Menten semantics.

The four chemical species react over three reaction channels: r_1 converting E and S to $E{:}S$, the backward reaction r_{-1} taking $E{:}S$ to E and S and the reaction r_2 converting the compound $E{:}S$ into product P and releasing the enzyme E. The reaction rates are governed by kinetic laws involving rate constants (k_1, k_{-1} and k_2) and the molecular counts of the species involved.

The reaction is initiated with a quantity of enzyme and a quantity of substrate and over the course of time the reactions will convert the substrate into the product (although many forward steps to the intermediate compound via reaction r_1 may be rapidly undone by the reverse reaction r_{-1}). However, reaction r_2 is not reversible in this model and so once product is produced by this reaction it cannot be destroyed by any other reaction.

Crucially, the rates of the reactions are functions of the model state, not constants. Thus in the initial state only reaction r_1 can fire (to form the compound). When some compound ($E{:}S$) is present then reactions r_{-1} and r_2 can fire, otherwise not. This switching behaviour is controlled

$$\begin{array}{lcllcl}
 & & r_request_1 & = & 0.1 & r_response_1 & = & 0.1 \\
 & & r_request_2 & = & 0.3 & r_response_2 & = & 0.3 \\
 & & r_request_3 & = & 0.01 & r_response_3 & = & 0.8 \\
 & & r_dispatch & = & 4.0 & r_response & = & 1.0 \\
\hline
Client & = & (request_1, r_request_1).Wait & & & & & \\
 & + & (request_2, r_request_2).Wait & & & & & \\
 & + & (request_3, r_request_3).Wait & & & & & \\
Wait & = & (response, \top).Client & & & & & \\
 & & & & & & & \\
Admin & = & (request_1, \top).(dispatch_1, r_dispatch).Respond & & & & & \\
 & + & (request_2, \top).(dispatch_2, r_dispatch).Respond & & & & & \\
 & + & (request_3, \top).(dispatch_3, r_dispatch).Respond & & & & & \\
Respond & = & (response_{int}, \top).(response, r_response).Admin & & & & & \\
Server_X & = & (dispatch_X, \top).(response_{int}, r_response_X).Server_X & & & & &
\end{array}$$

$$Client \underset{\mathcal{L}}{\bowtie} (Admin \underset{\mathcal{M}}{\bowtie} (Server_1 \parallel Server_2 \parallel Server_3))$$

$$where$$

$$\mathcal{L} = \{request_X, response\}$$

$$\mathcal{M} = \{dispatch_X, response_{int}\}$$

Figure 2 *PEPA model of a layer-7 two-way architecture*

by the kinetic laws, which use the number of molecules of each species as a multiplier (giving a zero rate when the species is not present, and thereby switching off certain reactions). This model reaches a deadlock state in which there is no free substrate and no free compound. In this state reaction r_1 cannot fire (because it requires the presence of the substrate) and reactions r_{-1} and r_2 cannot fire (because they require the presence of the compound).

The Bio-PEPA species definitions indicate which chemical species participate in each reaction and whether they are reactants consumed by the reaction or products produced by the reaction. The symbol $\downarrow$ is used for the former (because the molecular count of that chemical species goes down) whereas the symbol $\uparrow$ is used for the latter (because the molecule count of that chemical species goes up). The model equation of a Bio-PEPA model specifies which species participate in which reactions and whether they are initially present (denoted by a 1 in the model equation) or initially absent (denoted by a 0 in the model equation). The Bio-PEPA model of this system is shown in Fig. 3.

The PEPA model shown in Fig. 4 is derived from the Bio-PEPA input. It is presented in reagent-centric style [11], meaning that it indicates the direction of the reaction as a change from higher to lower quantities of the compound. If the reaction increases the quantity of the chemical species X ($\uparrow$ in Bio-PEPA) then the derived PEPA model will have a change from X_{low} to X_{high}, meaning that the molecular count of X is higher after the reaction. If the reaction decreases the quantity of the chemical species X ($\downarrow$ in Bio-PEPA) then the derived PEPA model will have a change from X_{high} to X_{low}, meaning that the molecular count of X is lower after the reaction. With only the direction of change known (from high to low, or vice-versa), it is possible to derive exactly the system of ordinary differential equations that specifies the continuous limit of this reaction behaviour [12]. Our software converts the derived PEPA model into a CTMC, which can then be solved for the steady-state distribution and related measures. However, in this case this is inappropriate since eventually all of the substrate is consumed and only enzyme and product remain, at this point the model is in a deadlocked state. We instead use

$$
\begin{aligned}
r_1 &= [k_1 \times E \times S] \\
r_{-1} &= [k_{-1} \times E{:}S] \\
r_2 &= [k_2 \times E{:}S]
\end{aligned}
$$

$$
\begin{aligned}
E &\stackrel{def}{=} r_1\downarrow + r_{-1}\uparrow + r_2\uparrow \\
S &\stackrel{def}{=} r_1\downarrow + r_{-1}\uparrow \\
E{:}S &\stackrel{def}{=} r_1\uparrow + r_{-1}\downarrow + r_2\downarrow \\
P &\stackrel{def}{=} r_2\uparrow
\end{aligned}
$$

$$
\left(E(1) \underset{\{r_1, r_{-1}, r_2\}}{\bowtie} \left(S(1) \underset{\{r_1, r_{-1}\}}{\bowtie} E{:}S(0)\right)\right) \underset{\{r_2\}}{\bowtie} P(0)
$$

Figure 3 *Bio-PEPA model for Michaelis–Menten reactions*

Species			*Rates*		
E_{high}	$\stackrel{def}{=}$	$(r_1, r_1).E_{low}$			
E_{low}	$\stackrel{def}{=}$	$(r_{-1}, r_{-1}).E_{high}$			
	$+$	$(r_2, r_2).E_{high}$			
S_{high}	$\stackrel{def}{=}$	$(r_1, r_1).S_{low}$			
S_{low}	$\stackrel{def}{=}$	$(r_{-1}, r_{-1}).S_{high}$	r_1	$=$	$k_1 \times E_{high} \times S_{high}$
$E{:}S_{high}$	$\stackrel{def}{=}$	$(r_{-1}, r_{-1}).E{:}S_{low}$	r_{-1}	$=$	$k_{-1} \times E{:}S_{high}$
	$+$	$(r_2, r_2).E{:}S_{low}$	r_2	$=$	$k_2 \times E{:}S_{high}$
$E{:}S_{low}$	$\stackrel{def}{=}$	$(r_1, r_1).E{:}S_{high}$			
P_{low}	$\stackrel{def}{=}$	$(r_2, r_2).P_{high}$			
P_{high}	$\stackrel{def}{=}$	Stop			

$$E_{high} \underset{*}{\bowtie} S_{high} \underset{*}{\bowtie} E{:}S_{low} \underset{*}{\bowtie} P_{low}$$

Figure 4 *Translated PEPA model for Michaelis–Menten reactions*

transient analysis of the computed Markov chain to enable a time-series plot allowing us to compute the time taken for all (or a portion of) the substrate to be consumed. We perform this in Section 6.

3 Passage time constraints

In this section we detail the constraints imposed on the form of the passage within the Markov chain by the uniformisation method of obtaining passage-time quantiles. In order to perform uniformisation on a (derived) CTMC we must specify two sets of states: the set of source states $\mathcal{S}$ and the set of target states $\mathcal{T}$. The passage set is the set of all states that lie on some path between the source and target sets including the source set but not including the target set. The intuition is that a source state is reached by an event which begins the passage, hence the sojourn time of a source state is included in the time taken to complete the passage. A target state is reached by an event which ends the passage, therefore the passage is completed when a target state is entered and its sojourn time should not be included in the time to complete the passage.

In order for our uniformisation calculation to be valid, it must be the case that no state within the passage set may transition to a state within the source set. The reason that this is invalid is that we must obtain the probability distribution of the source states at the beginning of the passage. If there is no transition from the passage set to one of the source states then this can be computed using the steady-state probability distribution of the embedded Markov chain to calculate the frequency with which each source state is entered relative to the frequency that the source set in general is entered. Where such a transition does exist then this calculation will not provide the probability distribution we require since it is possible to enter particular source states without beginning the passage of interest.

Although this would appear to restrict the set of models for which passage times may be calculated any model that violates this may be turned into an equivalent model that

doi: 10.1049/iet-sen.2009.0002

does not by duplicating the source states that may be returned to from within the passage set. The cost of this transformation is to increase the state-space size; however, the increase is linear. In fact, we must increase the state space by at most the size of the source set.

Note also that we cannot transition from a state within the passage set to a state outside both the passage set and the target set. Such a state either has a path back to the passage set and hence to the target set and is hence within the passage set (and as such may not transition into the source set). Alternatively the state is (or has a path to) a deadlocked state. Such deadlocked states are not allowed if there are multiple source states, since then the steady-state of the embedded Markov chain cannot be computed. If there is only one source state this is not required, however, we view such a deadlocked state as within the passage set. This also means that there is a non-zero probability that the passage of interest is never completed at all. More strictly then the passage set is defined as all those states that are reachable from the source set without passing through the target set.

4 Compositional models

The technique of uniformisation is defined over the underlying CTMC and not in terms of the higher-level formalism used to define the CTMC. However, for passage-time quantiles we must specify to the uniformisation routines the source and target states and this must be done in the higher-level formalism since the user should not be required to understand the derived state space of the model. In this section we concentrate on the process algebra PEPA and how our software with its query specification language allow a valid passage-time analysis to be presented to the uniformisation routine without user intervention.

When designing a query specification language which will be used for passage-time queries, one must consider two factors. Firstly can the compositional nature of both the model and the query be used to ensure that the constraints as specified in Section 3 are not violated? Secondly can the compositional nature be used to optimise the calculation of the results?

Two techniques are commonly used to describe the states along the passage of interest. Firstly state-based techniques require the user to specify the states of interest using the local states of the separate processes or tokens as a filter on the entire state space. Alternatively action or event sequences that result in the source or target states are described. Measurement specification is still an active area of research and a recent paper by the current authors describes a specification language which blends the use of activity observations with state-based filters [13].

4.1 Stochastic probes

To describe passages we insert probe components that observe activities performed by the rest of the model, via passive synchronisation, and change their local state accordingly. We then use the states of the probe components as a filter on the full state space of the model. Probe components can be manually composed and added to the model or specified in our (regular-expression-like) language, which is then automatically converted into PEPA components and inserted into the model.

The major rule that we must respect is that no state along the passage may have a transition outside of the passage or a transition which targets a source state.

For the purposes of this paper we will assume that the model performs a 'start' activity when and only when the model enters a source state. Similarly the model performs a 'stop' activity when and only when the model enters a target state. Elsewhere [14] we describe how such start and stop signals can be generated automatically when stochastic probes are added to the model in such a way that the behaviour of the model is not affected. We can therefore add a simple passage probe, which is running whenever the model is within the passage set.

Because the passage-probe may only be stopped by the model entering a state within the target set, it is not possible to transition out of the passage set. In the case that this was possible for the original model, such a state (outside of the passage set) is duplicated as the same state description but with the passage-probe component either running or not. This ensures that the first property is maintained. The passage probe can be defined and composed with the main system by

$$\begin{aligned} PassageStopped &= (start, \top).PassageRunning \\ &+ (stop, \top).PassageStopped \\ PassageStopped &= (start, \top).PassageRunning \\ &+ (stop, \top).PassageStopped \end{aligned}$$

$$PassageStopped \underset{L}{\bowtie} System$$

$$\text{where } L = \{start, stop\}$$

We can, in a similar fashion, add a second probe called a source probe, which is in the running state only when the model is in a state directly following the start of the passage. Any source states that may be revisited during the passage are duplicated in a similar manner to the above, and hence the duplicated source state is not considered to be within the source set for the purposes of the passage-time calculations. The definition of the source probe depends upon the alphabet of activities performed by the entire model; for one in which the alphabet is just $\{start, stop, a, b\}$ the source-probe is defined and composed

with the main system by

$$\begin{aligned} \mathit{SourceStopped} &= (\mathit{start}, \top).\mathit{SourceRunning} \\ &+ (\mathit{stop}, \top).\mathit{SourceStopped} \\ &+ (a, \top).\mathit{SourceStopped} \\ &+ (b, \top).\mathit{SourceStopped} \\ \mathit{SourceRunning} &= (\mathit{start}, \top).\mathit{SourceStopped} \\ &+ (\mathit{stop}, \top).\mathit{SourceStopped} \\ &+ (a, \top).\mathit{SourceStopped} \\ &+ (b, \top).\mathit{SourceStopped} \end{aligned}$$

$$\begin{aligned} &\mathit{SourceStopped} \underset{L}{\bowtie} \mathit{System} \\ \text{where}\quad &L = \{\mathit{start}, \mathit{stop}, a, b\} \end{aligned}$$

Astute readers should note that the first line of the *SourceRunning* definition is not required since the 'start' activity should not be performed once already in a source state but it is included for completeness. By adding simple probe components, which may be specified with an even simpler probe language, we can ensure that any necessary state duplications are made automatically during state-space generation. We also do not duplicate states where this is not necessary to stay within the constraints of the measurement.

Finally one might ask: What happens if the initial state is along the passage? In this case some states along the passage are duplicated during state-space generation, which may impact on how long it takes to solve the model. However, the results will not be affected. More specifically the states that lie on a path from the initial state to the target set will be duplicated and become transient states (states which are not visited in the long run). Since a passage-time measurement is not dependent on the initial state of the system, the user can always re-write the system equation to avoid this, but we stress that it will make no difference to the results obtained only how quickly they may be obtained.

4.2 Optimisation

As will be described in Section 5 in order to make sure that only the probability of completing the first passage and not subsequent passages are computed, the Markov chain is modified so that all target states have only one outward transition whose destination is an added absorbing (deadlocked) state. This means that there are some states in the modified Markov chain which are unreachable from the source set. These are the states that are not contained in the passage set or the target set, we will call this set $\mathcal{U}$.

In addition, we will never query the probability mass of any state within the unreachable set and the states in this set cannot affect the probability of other states and hence the passage-time quantiles we are computing. We must still derive this set as it will be used to calculate the seeding of the source states, that is the probability distribution at the beginning of the passage when all the probability mass is within the source set. However, once this is done these states may be removed for the rest of the computation. This would be done at the same time that the Markov chain is modified to have an absorbing state to which all target states must transition. As we will see the rest of the computation involves (possibly many) matrix multiplications in which the modified Markov chain is multiplied by a probability distribution. Therfore if the modified Markov chain can be made smaller by removing the states in the unreachable set $\mathcal{U}$, then it is a potentially large saving in both the space and time required for the computation.

Using the process algebra and probe setting described above, this set is easy to calculate; it is the set of all states in which the passage probe is in the 'stopped' state minus the target set. The target set itself may be computed using a probe that is similar to the source probe. The target probe can be defined and added to the main system by

$$\begin{aligned} \mathit{TargetStopped} &= (\mathit{stop}, \top).\mathit{TargetRunning} \\ &+ (\mathit{start}, \top).\mathit{TargetStopped} \\ &+ (a, \top).\mathit{TargetStopped} \\ &+ (b, \top).\mathit{TargetStopped} \\ \mathit{TargetRunning} &= (\mathit{stop}, \top).\mathit{TargetStopped} \\ &+ (\mathit{start}, \top).\mathit{TargetStopped} \\ &+ (a, \top).\mathit{TargetStopped} \\ &+ (b, \top).\mathit{TargetStopped} \end{aligned}$$

$$\begin{aligned} &\mathit{TargetStopped} \underset{L}{\bowtie} \mathit{System} \\ \text{where}\quad &L = \{\mathit{start}, \mathit{stop}, a, b\} \end{aligned}$$

Note that it is the same as the source probe with the roles of the 'start' and 'stop' activities reversed. This method, though, has the disadvantage that the target states may be unnecessarily duplicated since there was initially no restriction on a state looping back to a target state. That is, there is no reason why a state outside the passage set (including one within the target set itself) may not transition into a state within the target set, but this probe will distinguish such occurrences of the target states. This will only impact the calculation of the seeding of the source states since the duplicated target set states will be within the unreachable set. This is therefore entirely harmless if the source set has only one member since the seeding of the source states can be avoided when there is only one source state (it is the trivial seeding in which all of the probability mass is held by the single source state).

Regardless of how the target set and the unreachable set are identified, with a compositional approach to building the model it is straight-forward to see what kind of saving we can make by removing the unreachable set. This of course depends on the measurement but commonly we are measuring the response time as observed by a single component. Suppose there are two clients and one server. Each client may 'request' something from the server which will subsequently 'respond' to the client. The server will passively wait for all incoming requests and make a

doi: 10.1049/iet-sen.2009.0002

response after an exponentially distributed delay. In this case our measurement is begun with the first client, making a request and stopped when it receives its response. Since the state of the second client is independent of the state of the first client, there are as many states outside the passage-set and target-set as there are within. If we increase the number of clients this still means that the state space is split approximately equally by those states within the passage and target sets. So the unreachable set is roughly half the size of the model. It gets larger if we add states in which the clients can sojourn when they are not waiting on the server. In fact, the state space of the whole model can be divided into N equally sized sets where each set corresponds to a particular local state of the first client. A number of these sets, say $t < N$, are within the passage and target sets and some do not depend on the state of the first client. So in this example the unreachable set is $(N - t)/N$ times the size of the entire state space. This clearly may be well over half the size of the state space. Such a simple model can be complicated by competition for a scarce resource such as a server which only accepts a limited number of concurrent requests. This complicates the calculation somewhat, but in general we find that half is a good estimation as to how much of the state space is in the unreachable set and can therefore be removed from the modified Markov chain before any matrix multiplications take place.

5 Uniformisation

This section details the steps used to derive the cdf (and/or pdf) of a passage within a model represented as a CTMC in which we have specified a source and target-set and are comforming to the constraints laid out in Section 3. We first detail the prerequisites:

1. The CTMC may be represented by the generator matrix Q. The generator matrix is an $n \times n$ matrix where n is the number of states in the Markov chain. Each row corresponds to one state and the value in one cell of a row corresponds to the rate at which the Markov chain may transition from the state given by the row number to the state given by the column number. The diagonal values are given by subtracting from zero the sum of the other values in the row. Hence if we write $r(i, j)$ to mean the rate at which the Markov chain may transition from state i to j then the generator matrix Q is written as:

$$Q_{i,j} = \begin{cases} r(i,j), & i \neq j \\ 0 - (\sum_{k=1}^{n} r(i,k)), & \text{otherwise} \end{cases}$$

This requires that for any state i, the rate $r(i, i)$ is zero – this condition is usually stated by insisting that the model contains no self-loops.

2. The generator matrix may be solved to obtain the steady-state probability distribution π where π_i is the long-term probability of being in state i. This requires that the model be deadlock free.

3. The set of source states $\mathcal{S}$ and the set of target states $\mathcal{T}$. The probability at time t that we compute is the probability of moving from any of the states in $\mathcal{S}$ to any of the states in $\mathcal{T}$ within time t.

The steps in the computation of the pdf and the cdf of a particular passage within a CTMC are summarised as follows:

1. Uniformise the generator matrix to obtain the new matrix P by

$$P = Q/q + I$$

where Q is the generator matrix, I is the identity matrix and q is a rate value which is chosen to be greater than the magnitude of all of the rates within the generator matrix including the values along the diagonal. Therefore we have $q > \max_{ij} |Q_{ij}|$ which can be reduced to $q > \max_i |Q_{ii}|$ since the magnitude of the diagonal values in each row are the sums of the other values in the row which cannot be negative. Since q is of greater magnitude than any of the (negative) diagonal values dividing by q returns a negative number $x : -1 < x < 0$. This means that adding the identity matrix ensures that all rate values are positive.

The bottom graphs of Fig. 5 depict the probabilities of performing a n hops within time t for a fixed rate q. Note that these graphs do not depend on the passage being analysed only on the value of q.

2. Add to this uniformised matrix P an absorbing state. This state has no out-going edges to any state other than itself, which it loops to with probability 1.

3. Modify all target states (states in $\mathcal{T}$) to transition with probability one to the absorbing state. Call this new matrix P'. The reason for our absorbing state is to ensure that we compute the probability of the first passage and not subsequent completions of the passage. That is, if we are in state $i \in \mathcal{T}$ at time t then we know we are completing the passage at time t and it is not the case that we completed the passage at some earlier time and remained in or returned to state i.

4. Compute the probability distribution at the start of the passage, $\pi^{(0)}$. That is the probability, in the original Markov chain, of being in each particular source state given that we must be in one of the source states. This is done using the embedded Markov chain from the original Markov chain Q. We obtain $\pi^{(0)}$ by

$$\pi_k^{(0)} = \begin{cases} 0, & k \notin \mathcal{S} \\ \pi_k/\pi_{\mathcal{S}}, & k \in \mathcal{S} \end{cases}$$

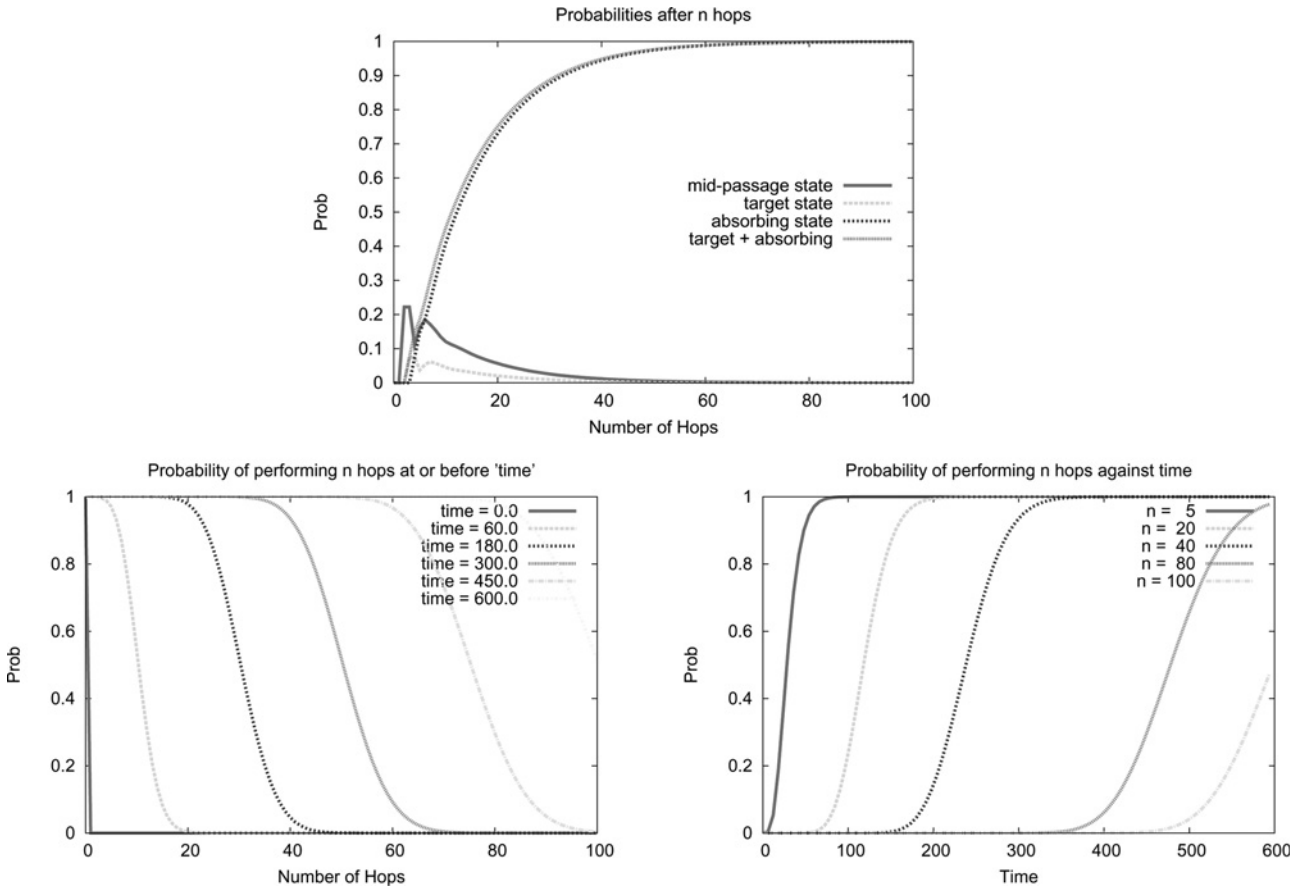

Figure 5 *Top graph shows the probability of completing after n hops, graph on the bottom left shows the probability of completing n hops within a given time t, whereas the graph on the bottom right shows the probability of performing a given number hops against t*

where π_k is the steady-state probability in the embedded Markov chain of being in state k and $\pi_{\mathcal{S}}$ is the steady-state probability in the embedded Markov chain of being in any one of the source states, that is $\sum_{k\in\mathcal{S}} \pi_k$. Where there is exactly one source state then the steady-state probability distribution need not be calculated and $\boldsymbol{\pi}^{(0)}$ is given by

$$\pi_k^{(0)} = \begin{cases} 0, & k \notin \mathcal{S} \\ 1, & k \in \mathcal{S} \end{cases}$$

since for the one source state j, $\pi_j = \pi_{\mathcal{S}}$.

5. Compute the probability distribution after n hops of the uniformised Markov chain; given by $\boldsymbol{\pi}^{(n)}$ where $\boldsymbol{\pi}^{(n+1)} = \boldsymbol{\pi}^{(n)}P'$. The top graph in Fig. 5 is an example passage calculation showing the probabilities of being in certain states after a given number of hops. In particular, the probability of being in a target state is the probability that the passage is completed in exactly n hops while the probability of being in the absorbing state is the probability that the passage is complete in less than n hops. The addition of these two probabilities gives the probability of completing within n hops.

6. To compute the cdf at each time t we finally adjust each hop value by the probability that n hops are performed within time t. Recall that each hop has the same average duration of $1/q$ so the probability that we may perform n hops in time t is an erlang distribution. An erlang distribution for a particular value of q is shown in the bottom two graphs of Fig. 5. So then our cdf at time t is: $\sum_{n=0}^{n=\infty} Er_t^{(n)}\boldsymbol{\pi}_{\mathcal{T}}^{(n)}$ where $Er_t^{(n)}$ is the probability that the nth hop will be performed at or before time t and $\boldsymbol{\pi}_{\mathcal{T}}^{(n)}$ is the probability of being in any of the target states after exactly n hops of the uniformised matrix, P'.

In the final step above we have computed the cdf of the passage by multiplying the probability of being in a target state after exactly n hops by the probability of performing n within the time t. This probability is given by $(1 - \mathrm{e}^{-qt}\sum_{k=0}^{n=1}(qt)^k/k!)$. For the pdf we substitute this for the probability of performing n hops at exactly time t. This is given by: $q^n t^{n-1}\mathrm{e}^{-qt}/(n-1)!$.

For completeness we provide the full formulae for computing the cdf and pdf of the passage, respectively given by

$$F_{\vec{ij}}(t) = \sum_{n=1}^{\infty}\left(\left(1 - \mathrm{e}^{-qt}\sum_{k=0}^{n-1}\frac{(qt)^k}{k!}\right)\sum_{k\in\vec{j}}\pi_k^{(n)}\right)$$

doi: 10.1049/iet-sen.2009.0002

and

$$f_{\vec{ij}}(t) = \sum_{n=1}^{\infty}\left(\frac{q^n t^{n-1} e^{-qt}}{(n-1)!}\sum_{k\in\vec{j}}\pi_k^{(n)}\right)$$

A summary of the mathematical relationships involved in this algorithm is given in Table 1.

5.1 Producing passage-time quantiles

The above description of an algorithm for computing passage-time quantiles is an ideal implementation with one important flaw; the final step calls for a summation to infinity. It is worth re-stating exactly the meaning of this summation. By the final step we have the means to compute an infinite number of 'hops' where each hop is a probability distribution from which can be extracted the probability of completing the passage in exactly n hops ($\pi_{\mathcal{T}}^{(n)}$). In addition, we know the probability of performing n hops in a given amount of time t since each hop has the same average duration which is exponentially distributed, giving an erlang distribution for completing n hops within t time ($Er_t^{(n)}$). For any given n multiplying these two values together gives the probability of completing the passage in the given time t using exactly n hops. Since we do not care how many hops it takes if we add together these probabilities for all values of n then we have the probability that the passage is completed within time t.

Clearly, summing all of these probability values from zero to infinity is impossible for a computer to do. However, there will be some value X for which all values $n_X > X$, the probability of completing the passage within the given time in exactly n_X hops is negligible. Hence at this point we may stop computing probability values. Here we show two conditions that suffice to find the value X.

Previously one method was to compute the probabilities for successive values of n and whenever the probability $((P_n((n)))$ was sufficiently low we assume that subsequent values will also be sufficiently low. This method has problems when the passage we must complete has separate paths that vary greatly in their length of hops. In this instance, it is possible for the probability to drop below the threshold value but later climb above it. In this case the given method would stop calculating the probability values before they have a chance to rise above the threshold once again.

Table 1 Relationships between the probability values

Probability of	Value	Name
completing the passage in exactly n hops	$\sum_{k\in\mathcal{S}}\pi_k^{(n)}$	$\pi_{\mathcal{T}}^{(n)}$
performing n hops within time t	$(1-e^{-qt}\sum_{k=0}^{n-1}(qt)^k/k!)$	$Er_t^{(n)}$
completing the passage in n hops by time t	$\pi_{\mathcal{T}}^{(n)}Er_t^{(n)}$	$P_n(t)$
completing the passage by time t	$\sum_{n=0}^{n=\infty}P_n(t)$	cdf

Our method is to monitor the probability of being in the absorbing state after n hops – we designate this value $\text{Abs}^{(n)}$. When this value climbs to within a suitable threshold of 1 then there is no probability left to flow through the target states. Hence the probability of being in a target state for all values of n greater than the current value must be below the threshold value, since this probability is multiplied by the probabilty of performing n hops within time t we know that all subsequent probabilities will be below the threshold.

This method performs well; however, for small values of t we find that we compute more hop values than are required. This is because for small values of t it is unlikely that we are able to perform a large number of hops. However, if the passage is long then it may be that the probability of being in the absorbing state does not climb to within the threshold of one until n is large – where by 'large' we mean 'larger than the number of hops we could hope to perform within the time t'. Therefore we also monitor the value of $P_n(t)$ – the probability of performing n hops within time t – whenever this value falls below the threshold, we know that any subsequent values of n will yield negligible probability at time t [since $P_n(t)$ is involved in the product] and hence we have determined a suitable value of X.

Our algorithm may be summed up by a recursive function as

$$\text{cdf}(n,t) = \begin{cases} 0, & \text{Abs}^{(n)} > (1-\text{threshold}) \\ 0, & Er_t^{(n)} < \text{threshold} \\ (\pi_{\mathcal{T}}^{(n)} * Er_t^{(n)}) & \\ \quad +(\text{cdf}(n+1,t)), & \text{otherwise} \end{cases}$$

and similarly for the pdf function

$$\text{pdf}(n,t) = \begin{cases} 0, & \text{Abs}^{(n)} > (1-\text{threshold}) \\ 0, & Er_t^{(n)} < \text{threshold} \\ (\pi_{\mathcal{T}}^{(n)} * Erp_t^{(n)}) & \\ \quad +(\text{pdf}(n+1,t)), & \text{otherwise} \end{cases}$$

where $Erp_t^{(n)}$ is the probability of performing the nth hop at exactly time t and is given by: $q^n t^{n-1} e^{-qt}/(n-1)!$. Since there is a lot of shared computation our implementation computes both the cdf and the pdf of the passage together.

5.2 Analysis results

The passage time results applied to our layer-7 example model are shown in Fig. 6. In the model there were three kinds of requests, so we have analysed the response time of each kind of request separately as well as the response-time in general. These four analyses were specified with the stochastic probe definitions:

$request_1$:start, *response*:stop

$request_2$:start, *response*:stop

$request_3$:start, *response*:stop

($request_1$|$request_2$|$request_3$):start, *response*:stop

These definitions were then translated automatically by our software into PEPA components and attached to the model. The software then derives the state space which is guaranteed to adhere to the constraints of the passage-time calculation and the software already knows the source and target sets of states.

5.3 Technical points

We have shown how to compute passage-time quantiles from continuous time Markov chains. However, we have left the actual numerical computation as given, although this is non-trivial. For the cdf, we must compute

$$F_{\vec{ij}}(t) = \sum_{n=1}^{\infty}\left((1-\mathrm{e}^{-qt}\sum_{k=0}^{n-1}\frac{(qt)^k}{k!})\right)\sum_{k\in\vec{j}}\pi_k^{(n)})$$

Notice in particular that we must compute $k!$ for what may be large values of k. In addition, we must compute qt^k, also for potentially large values of k. The large values here are in the order of the number of hops, this value may be quite high – a value in the order of thousands is not uncommon (in [15] this number is said to be of the order of qt). Hence we can expect to encounter a problem with overflow. Even if some arbitrary precision library is used (at a performance cost) computing the cdf in this way is inefficient. Our first observation is that

$$\frac{(qt)^k}{k!} \text{ is equal to: } \prod_{i=1}^{i=k}\frac{qt}{i}$$

which allows us to avoid the computation of the large power and factorial values.

Now for each value of n we must compute $\sum_{k=0}^{N}\frac{(qt)^k}{k!}$. We need not compute each term separately. We can instead compute the infinite list of values by the recursive function

$$\text{sumvalues}(n, \text{current}) = \text{current} : \text{rest}$$
$$\text{where rest} = \text{sumvalues}\left(n+1, \text{current} + \sum_{k=0}^{N}\frac{(qt)^k}{k!}\right)$$

Because our implementation is in the lazy programming language Haskell we need not worry about the computation of an infinite list since we will only ever examine a finite number of elements from it. For a strict language this laziness can be easily simulated. We now observe that even this computation does a large amount of re-computation. Namely the successive values of $\sum_{k=0}^{N}\frac{(qt)^k}{k!}$ recompute all previous values. However, we can use the same trick:

$$\text{prodvalues}(k, \text{current}) = \text{current} : \text{rest}$$
$$\text{where rest} = \text{prodvalues}\left(k+1, \text{current} \times \frac{qt}{k}\right)$$

This means that we can now update our sumvalues function to take advantage of this. It now becomes a list transformation function which takes in the list of product values computed by the above prodvalues function.

$$\text{sumvalues}(\text{current}, (n, p) : \text{rest}) = (n, \text{current}) : \text{restsum}$$
$$\text{where restsum} = \text{sumvalues}(\text{current} + p, \text{rest})$$

We can also factor out the code to calculate the probability of being in the absorbing state and/or a target state after exactly n hops. Since otherwise we will recompute these values for each time value we desire. Once we have factored out all the common computation we have a set of infinite lists that

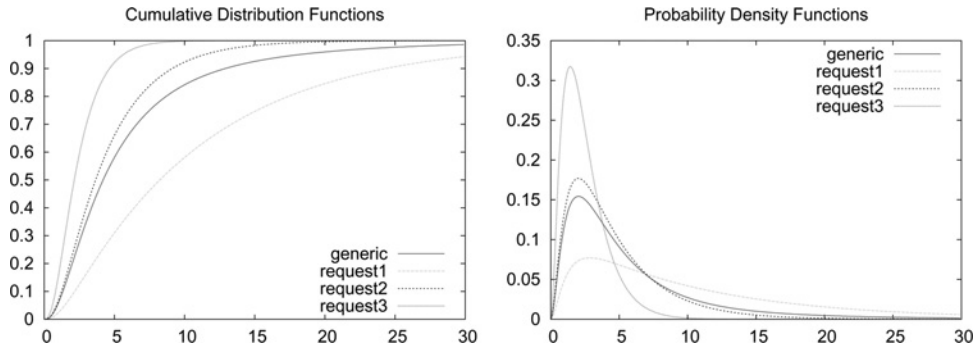

Figure 6 *Results of analysing passages within the layer-7 model*

doi: 10.1049/iet-sen.2009.0002

map n from $n = 0$ to $n = \infty$ to values used in the computation of the cdf and pdf. We need only operate on these lists for the values of t.

5.4 High rates

In this section we detail why the use of very high rates within the model can render the uniformisation technique unusable. Consider the very simple Markov chain shown in Table 2.

This model can be solved for the steady-state solution very quickly because the number of states is very small. Using the steady-state solution we can calculate the average time it takes to complete some passage, for example from the source state zero to the target state three. This gives an average passage time of just over 20 time units. However, if we wish to calculate passage-time quantiles for the same passage we must uniformise the chain. Unfortunately, this involves dividing through by the magnitude of the largest rate, which is slightly larger than 50 000. We already know that the passage takes on average 20 time units, so the number of hops we will have to compute is at least: 50 000 × 20, rather a lot for such a simple passage.

The same problem occurs whenever we have large rates that are involved in the passage, but the passage takes much longer than the reciprocal of the largest rates. This may occur because some small rates are involved in the passage and/or because of looping within this model. The authors have encountered this problem in the modelling of biochemical reactions, in particular those with a Michaelis–Menten semantics which involves the enzyme and substrate in very fast equilibrium with their complex.

6 Transient measures

This paper is mostly concerned with passage-time measurements. Other transient measures can also be calculated via uniformisation. Most generally we can calculate a probability distribution at a given time from a given set of source states. Commonly the set of source states may be the singleton set containing only the initial state of the model. Calculating a series of probability distributions at given time intervals allows the generation of a time-series plot, which plots the population of a given component type (or state) against time. Our example model

Table 2 Simple Markov chain exhibiting high rates within a measured passage

	0	1	2	3
0		0.1		
1			50 000	
2		50 000		0.1
3	0.1			

of a chemical reaction given in Section 2.3 is analysed using this more general style of transient analysis. Recall that the model eventually becomes deadlocked, thus meaning many other kinds of analysis over the derived Markov chain are inappropriate. The resulting time-series plot is shown in the graph of Fig. 7. This shows that initially the populations of the enzyme (E) and the substrate (S) drop rapidly, but after a brief time the population of the enzyme recovers together with an increasing product (P) population. The opposite happens to the complex ($E{:}S$), which initially becomes high in concentration but decreases as more product is produced.

To compute such a probability distribution we use much of the same steps as in passage-time calculations. However, we need not modify the uniformised chain to contain an absorbing state since in this case we do wish to calculate the probability of returning to the source states. However we must now calculate, for each time point we desire t, for each hop n the probability that we have performed exactly n hops at time t.

Because we cannot add an absorbing state, we lose our novel method of terminating the hop calculations and must rely on the probability of completing an ever increasing number of hops within the time t.

In Section 5 we concluded with the full formulae for calculating the cdf and pdf of the passage in question. In the same spirit of completeness, we provide here the full formula for calculating the probability distribution of a given Markov chain at time t from the initial state:

$$\pi(t) = \sum_{n=0}^{\infty} \left(\frac{(qt)^n \exp^{-qt}}{n!} \pi^{(n)} \right)$$

Here once again $\pi^{(n)}$ is the probability distribution as given

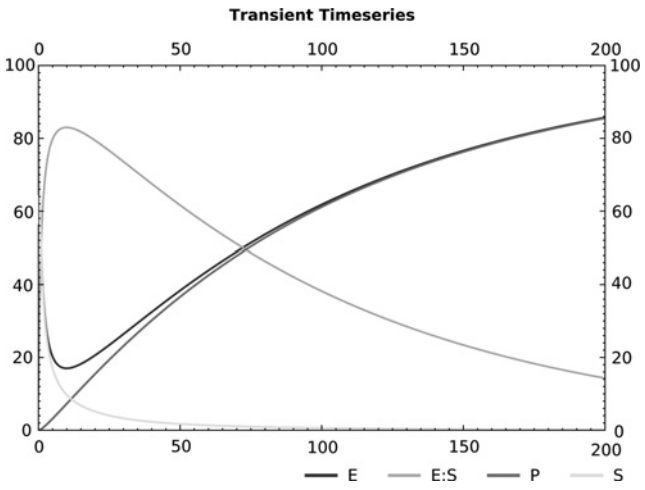

Figure 7 *Example time-series plot, plotting the concentration of four species as a function of time from the start of the experiment*

by the nth hop value and $(qt)^n \exp^{-qt}/n!$ is the probability that at time t there have been exactly n hops performed. Note that the sum is from $n = 0$ since at a given time there is a possibility that no hops have been performed. For passage-time calculations this is not necessary since the probability of completing the passage within zero hops is zero.

To conclude this section we clear up some terminology which we use. A passage-time calculation involves computing the probability that the model transitions into a target state a given time after first entering a source state. Sometimes the set of source states is the singleton set containing the initial state of the model. This is commonly used in 'time-to-failure' queries such as: 'What is the probability that the server has failed within one week of operation?' When we say transient analysis we usually refer to the more general set of transient analyses, which call for the computation of a probability distribution (even if only one state's probability is examined) a given time after a set of source states have been entered. Such analyses are used commonly for biochemical modelling but are also used to answer such queries as: 'What is the probabilty that the system is operational x minutes after a server crash?' In this query we wish to include the possibility that within the x minutes the server was repaired but crashed again. It is therefore not a passage-time calculation but a more general transient analysis.

7 Implementation

The techniques described in this paper have been fully implemented in the International PEPA Compiler (ipc) based on the ipclib [16]. This is a compiler for the PEPA [1], is open source software and may be downloaded from: http://www.dcs.ed.ac.uk/pepa/tools/ipc/.

The input language may be a PEPA model, a Bio-PEPA model which may then be converted into a PEPA model or a Petri net specified via the Hydra [3] model file syntax or the XML format used by the PIPE [17] software.

The diagram in Fig. 8 shows the input and outputs of the ipc software tool built on top of, and released together with the ipclib library. The user may start at any level, with all subsequent levels in the downward direction automatically derived from the level above. So the user may start with a Bio-PEPA model, which is translated into a PEPA model or write the PEPA model by hand. In either case they then provide a probe specification that is automatically converted into a PEPA component and added to the model to give a second PEPA model and a set of conditions on local component states that define the source and target sets. Or the user could manually input these source and target conditions. Alternatively, the user could input a Petri net but in this case they must specify the source and target conditions manually since probes are not appropriate for

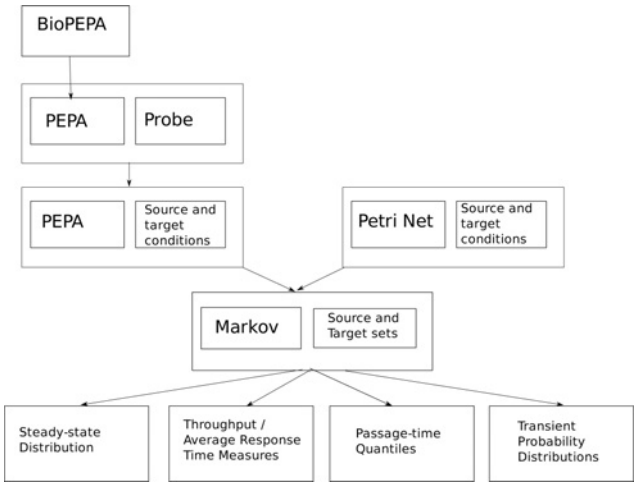

Figure 8 *Inputs and outputs of the International PEPA Compiler*

Petri nets. Whether we now have a PEPA model or a Petri net, this is compiled into a Markov chain and the conditions are evaluated for all states of the Markov chain to give the source and target sets of states. From this point all that is left is to derive the performance measure that the user desires, whether it be a steady-state distribution or a transient measure such as the calculation of passage-time quantiles. Of course, where the users's query does not involve a passage then the target set need not be specified. For a transient measure the source set must still be specified, although this is commonly the initial state. For steady-state distributions and queries that can be derived from it (such as throughput and average response-time), there is no need to even specify the set of source states.

7.1 Comparison with existing techniques

The naïve approach that we briefly illustrated in Section 5.1 is to compute for successive values of n until such values drop below a threshold. This method may be summed up by

$$\mathrm{cdf}(n,t) = \begin{cases} 0, & (\pi_{\mathcal{T}}^{(n)} * Er_t^{(n)}) < \text{threshold} \\ (\pi_{\mathcal{T}}^{(n)} * Er_t^{(n)}) & \\ \quad +(\mathrm{cdf}(n+1,t)), & \text{otherwise} \end{cases}$$

As we mentioned above this algorithm suffers from a problem if the input passage has multiple paths to completion of varying lengths. In this case the value at some n may drop below the threshold but may later rise above the threshold again. The simple solution would stop after the first time it drops below the threshold.

As an improvement on this technique the Markovian response-time analyser Hydra [3, 18–20] monitors the value of the erlang distribution with a q rate parameter and n hop parameter. The Hydra solution can therefore be

doi: 10.1049/iet-sen.2009.0002

summarised by

$$\mathrm{cdf}(n, t) = \begin{cases} 0, & Er_t^{(n)} < \text{threshold} \\ (\pi_T^{(n)} * Er_t^{(n)}) & \text{otherwise} \\ \quad +(\mathrm{cdf}(n+1, t)), & \end{cases}$$

Therefore this solution will compute the same values as our solution in all cases because our solution contains the same condition. However, our solution is a further refinement that allows us to avoid needless computation for some values of n. In particular, where the t-range – that is the times for which we should compute the passage-time quantiles – specified is too large. Suppose the user has specified a t-range of 1 – 1000 but the passage has a probability very close to one of completing by time 500. Because there is a large probability of completing the passage by time 500 this means that there is a large probability of completing the passage within a number of hops X and that X hops are very likely to be performed within 500 time units. This means that for time values over 500 there will be a possibility to perform more than X hops and the Hydra solution will continue to compute probabilities for these hop values. However, our solution would recognise that such values cannot add anything to the cdf because you are very like to have completed the passage before X hops. In the case of the cdf this could be mitigated by incorporating the naïve solution but this is not as effective as for computing the pdf.

Our solution has a further, related, advantage; the user need not specify the upper bound on the t-range at all. The user need only give the start of the t-range and the steps in which we wish to increase the value of t. This is because using our technique we can calculate the value X at which performing more than X number of hops will not significantly add to the probability of completing the passage (because there is a probabilty within the threshold of being in the absorbing state by X hops). We can then use this to work out the upper bound on the t-range by calculating the value of t such that performing X hops within time t is significantly likely. In order that the user need not specify a t-range at all we default to a starting time of zero and a time step of the calculated stop-time divided by one hundred. The user may then override any of the start time, the stop time, the time increments and the number to divide the t-range by in order to obtain the time increments.

8 Conclusions

In this paper we have detailed a software library ipclib and associated tool ipc for the derivation of performance measures from process algebra models that may be translated into continous time Markov chains. Once the Markov chain has been derived the common steady-state probability distribution and associated performance measures such as throughput and average passage duration may be computed. In this paper, though, we have focused on the transient measure facilities provided by ipc. In particular, the calculation of passage-time quantiles is provided for via the compositional approach to model and query specification. In particular, the stochastic probes specification language allows the compiler to automatically prepare the Markov chain in a suitable way to avoid an invalid passage-time computation. It is very encouraging that the constraints on a valid passage-time calculation are not only checked but where there is a violation automatic state duplication occurs in order to amend the passage into a suitable form. Moreover, states not required within the uniformised Markov chain may be readily determined and thus removed in order to optimise the query.

We have also given a detailed account of the calculation of passage-time quantiles and densities from continuous-time Markov chains. We have shown the process from non-uniformised Markov chain through the uniformisation, modification with an absorbing state, the calculation of the hop probability distributions and the calculations involved in producing the final cdf and pdf. Two important properties that allow the otherwise infinite calculation to terminate were identified which have the desired conservative feature – meaning that we never terminate too early producing an erroneous answer. However, the combination of the two provides early detection to avoid some needless calculations. Finally, we feel that our paper provides a good introduction to the topic of uniformisation in general.

9 Acknowledgments

The authors are supported by the EU FET-IST Global Computing 2 project SENSORIA (Software Engineering for Service-Oriented Overlay Computers (IST-3-016004-IP-09)). The ipc/Hydra tool chain has been developed in co-operation with Jeremy Bradley, Will Knottenbelt and Nick Dingle of Imperial College, London.

10 References

[1] HILLSTON J.: 'A compositional approach to performance modelling' (Cambridge University Press, 1996)

[2] CIOCCHETTA F., HILLSTON J.: 'Bio-PEPA: a framework for the modelling and analysis of biological systems', *Theoreti. Comput. Sci.*, 2008, **410**, (33-34), pp. 3065–3084

[3] DINGLE N.J., KNOTTENBELT W.J., HARRISON P.G.: 'HYDRA: HYpergraphbased Distributed Response-time Analyser'. Proc. 2003 Int. Conf. on Parallel and Distributed Processing Techniques and Applications, 2003, vol. 1, pp. 215–219

[4] CLARK G., COURTNEY T., DALY D., DEAVOURS D., DERISAVI S., DOYLE J.M., SANDERS W.H., WEBSTER P.: 'The Möbius modeling tool'.

PNPM'01: Proc. Ninth Int. Workshop on Petri Nets and Performance Models (PNPM'01), USA, 2001, p. 241

[5] TRIBASTONE M., DUGUID A., GILMORE S.: 'The PEPA Eclipse Plug-in', *Perform. Eval. Rev.*, 2009, **36**, (4), pp. 28–33

[6] DIENER J.D.: 'Empirical comparison of uniformization methods for continuous-time Markov chains', in STEWART W.J. (ED.): 'Computations with Markov chains' (Kluwer Academic Publishers, 1995), pp. 547–570

[7] KATOEN J.P., KHATTRI M., ZAPREEV I.S.: 'A Markov reward model checker'. Quantitative Evaluation of Systems (QEST), 2005, pp. 243–244

[8] KATOEN J.P., ZAPREEV I.S.: 'Safe on-the-fly steady-state detection for timebounded reachability'. Quantitative Evaluation of Systems (QEST), 2006, pp. 301–310

[9] KWIATKOWSKA M., NORMAN G., PARKER D.: 'PRISM 2.0: a tool for probabilistic model checking'. Proc. First Int. Conf. on Quantitative Evaluation of Systems (QEST'04), 2004, pp. 322–323

[10] AZIZ A., SANWAL K., SINGHAL V., BRAYTON R.: 'Model-checking continuoustime Markov chains', *ACM Trans. Comput. Logic*, 2000, **1**, (1), pp. 162–170

[11] CALDER M., GILMORE S., HILLSTON J.: 'Modelling the influence of RKIP on the ERK signalling pathway using the stochastic process algebra PEPA'. Transactions on Computational Systems Biology VII, 2006, (*LNCS*, **4230**)

[12] CALDER M., GILMORE S., HILLSTON J.: 'Automatically deriving ODEs from process algebra models of signalling pathways'. Proc. Computational Methods in Systems Biology (CMSB 2005), 2005, pp. 204–215

[13] CLARK A., GILMORE S.: 'State-aware performance analysis with eXtended Stochastic Probes'. Proc. Fifth European Performance Engineering Workshop (EPEW 2008), 2008, (*LNCS*, **5261**), pp. 125–140

[14] ARGENT-KATWALA A., BRADLEY J., CLARK A., GILMORE S.: 'Location-aware quality of service measurements for service-level agreements'. Proc. Third Int. Conf. on Trustworthy Global Computing (TGC'07), 2008, (*LNCS*, **4912**), pp. 222–239

[15] BAIER C., HAVERKORT B.R., HERMANNS H., KATOEN J.P.: 'Model checking continuous-time Markov chains by transient analysis'. CAV'00: Proc. 12th Int. Conf. on Computer Aided Verification, 2000, pp. 358–372

[16] CLARK A.: 'The ipclib PEPA Library'. Proc. Fourth Int. Conf. Quantitative Evaluation of SysTems (QEST), 2007, pp. 55–56

[17] DINGLE N.J., KNOTTENBELT W.J., SUTO T.: 'PIPE2: a tool for the performance evaluation of generalised stochastic Petri Nets', *SIGMETRICS Perform. Eval. Rev.*, 2009, **36**, (4), pp. 34–39

[18] KNOTTENBELT W.J.: 'Generalised Markovian analysis of timed transition systems'. Master's thesis, Department of Computer Science, University of Cape Town, 1996

[19] DINGLE N.J.: 'Parallel computation of response time densities and quantiles in large markov and semi-Markov models'. PhD thesis, Department of Computing, Imperial College London, University of London, 2004

[20] BRADLEY J., DINGLE N., GILMORE S., KNOTTENBELT W.: 'Extracting passage times from PEPA models with the HYDRA tool: a case study'. Proc. 19th Annual UK Performance Engineering Workshop, 2003, University of Warwick, pp. 79–90

doi: 10.1049/iet-sen.2009.0002