Derivation of Passage-time Densities in PEPA Models using ipc: the Imperial PEPA Compiler

Jeremy T. Bradley[1] Nicholas J. Dingle[1] Stephen T. Gilmore[2] William J. Knottenbelt[1]

[1] Department of Computing, Imperial College London
180 Queen's Gate, London SW7 2BZ, United Kingdom.
{jb,njd200,wjk}@doc.ic.ac.uk
[2] Laboratory for the Foundations of Computer Science,
University of Edinburgh, Edinburgh EH9 3JZ, United Kingdom.
stg@dcs.ed.ac.uk

Abstract

We present a technique for defining and extracting passage-time densities from high-level stochastic process algebra models. Our high-level formalism is PEPA, a popular Markovian process algebra for expressing compositional performance models. We introduce ipc*, a tool which can process PEPA-specified passage-time densities and models, by compiling the PEPA model and passage specification into the DNAmaca formalism. DNAmaca is an established modelling language for the low level specification of Markov and semi-Markov chains, which can analyse models with up to* 100 *million states. We provide performance results for* ipc*/DNAmaca and comparisons with another PEPA tool, PRISM. Finally, we generate passage-time densities and quantiles for a case study of a high-availability web cluster.*

1. Introduction

Passage-time densities are key metrics for performance modellers of distributed computer and communication systems. Indeed, Service Level Agreements (SLAs) often use passage-time quantiles as contractual obligations, e.g. 98% of text messages must be delivered within 2.5 seconds. However, the ability to derive these passage-times formally and from a high-level model is only now becoming practical, due to the fact that passage-time calculations in large systems require correspondingly large amounts of computational effort. In this paper, we present ipc, a tool which can generate passage-times over one such high-level modelling paradigm, PEPA.

PEPA [16, 17] is a popular Markovian process algebra for specifying compositional performance models. To provide a sufficiently powerful analysis capability for our PEPA models, we will make use of DNAmaca [19], a tool for the numerical analysis of low-level Markov and, more recently, semi-Markov [3] specifications. DNAmaca originally specialised in performing large-system steady-state analysis [20, 21]. Subsequently, the capacity to perform transient and passage-time analysis on systems with large state-spaces has been incorporated into DNAmaca as well [2, 9, 15].

In this paper, we present ipc, the *Imperial PEPA compiler*, which compiles system-level passage-time requirements, along with the associated PEPA model, to a DNAmaca specification. ipc automatically derives and compiles in *stochastic probes*, which are small fragments of process algebra that specify the start and end points of passage times in PEPA models. ipc does not unfold the global state space, instead using the compositional nature of the PEPA model to construct an equivalent DNAmaca model. In this way, ipc performs the compilation in linear time, with respect to the size of the original PEPA description.

ipc is meant to complement current PEPA tools [7, 8, 12, 22], which already support steady-state and transient measures and rewards. As such, we will specifically focus on the passage-time aspect of PEPA model analysis. The rest of this paper is organised as follows: we introduce PEPA in Section 2, the current PEPA tool-base in Section 3 and the DNAmaca modelling language in Section 4. The ipc tool architecture is described in Section 5. In Section 6, we present a PEPA case study of a high-availability web server. Section 7 compares the performance of ipc/DNAmaca with the PRISM tool. Finally, in Section 8, we describe the derivation of passage-time densities and quantiles from our PEPA case study using stochastic probes.

2. PEPA

PEPA is a parsimonious stochastic process algebra that can describe compositional stochastic models. These models consist of components whose actions incorporate random exponential delays. The syntax of a PEPA component, P, is represented by:

$$P \quad ::= \quad (a, \lambda).P \;\big|\; P + P \;\big|\; P \underset{S}{\bowtie} P \;\big|\; P/L \;\big|\; A \qquad (1)$$

$(a, \lambda).P$ is a prefix operation. It represents a process which does an action, a, and then becomes a new process, P. The time taken to perform a is described by an exponentially distributed random variable with parameter λ. The rate parameter may also take a $\top$-value, which makes the action passive in a cooperation (see below).

$P_1 + P_2$ is a choice operation. A race is entered into between components P_1 and P_2. If P_1 evolves first then any behaviour of P_2 is discarded and vice-versa.

$P_1 \underset{S}{\bowtie} P_2$ is the cooperation operator. P_1 and P_2 run in parallel and synchronise over the set of actions in the set S. If P_1 is to evolve with an action $a \in S$, then it must first wait for P_2 to reach a point where it is also capable of producing an a-action. In an active cooperation, the two components then jointly produce an a-action with a rate that reflects the slower of the two components (usually the minimum of the two individual a-rates). In a passive cooperation, where P_1, say, can evolve with an $(a, \top)$-transition, the joint a-action inherits its rate from the P_2 component alone.

P/L is a hiding operator where actions in the set L that emanate from the component P are rewritten as silent τ actions (with the same appropriate delays). The actions in L can no longer be used in cooperation with other components.

A is a constant label and allows, amongst other things, recursive definitions to be constructed.

3. Survey of PEPA Tools

There are a number of methods and tools available for solving PEPA models. One way in which a PEPA model can be solved is to use the PEPA Workbench [12] to generate the state space of the model and the infinitesimal generator matrix of the underlying Markov chain. The PEPA Workbench writes this matrix in the concrete syntax of the Maple computer algebra system [23] so this can be solved conveniently in the high-level mathematical computing environment which Maple offers. This method gives us an option which is not supported by any of the other PEPA tools, which is to solve the model symbolically in terms of the symbolic rates used in the model, instead of solving it only for a particular set of concrete values of these rates. Unfortunately this facility is practical only for very small models. For models of even moderate size it is necessary to use concrete values for the rates.

Other PEPA tools now encompass a number of well-engineered direct solution methods. The Möbius multi-formalism modelling framework supports PEPA as one of its input languages [8]. Möbius provides efficient sparse matrix-based implementations of steady-state and transient solvers as well as a simulator.

A different approach to the representation of the infinitesimal generator matrix of the CTMC is taken by the PRISM probabilistic symbolic model checker [22]. PRISM stores the matrix as a multi-terminal binary decision diagram (MTBDD) which offers compact storage for state spaces of significant size. PRISM supports PEPA as one of its input languages and offers a range of numerical solution procedures: Power, Jacobi, forwards and backwards Gauss-Seidel, JOR and forwards and backwards SOR.

Our contribution here is to allow additional solution procedures and passage-time analysis capabilities to be accessed via ipc, a tool written in the Haskell lazy functional programming language [18]. Its purpose is to compile a PEPA model into the input language of Knottenbelt's DNAmaca analyser [19]. The possible steady-state solution methods offered by DNAmaca include direct methods (Gaussian Elimination, Grassmann), classical iterative methods (Gauss-Seidel, fixed SOR, dynamic SOR), Krylov subspace techniques (BiCG, CGNR, CGS, BiCGSTAB, BiCGSTAB2, TFQMR) and decomposition-based methods (AI (Aggregation-Isolation), AIR).

4. The DNAmaca Modelling Formalism

DNAmaca is a modelling language for Markov and semi-Markov chains. As many previous publications already exist [2, 3, 9, 15, 19, 20, 21] describing the mathematical foundation for the calculation of steady-state, transient and passage-time distributions in such models, we will not dwell on the complete details here; rather in Section 4.1, we will briefly describe the theory behind the *uniformization* technique [14, 26], used by the HYDRA release [10, 11] of DNAmaca to calculate passage-time quantities in Markov models.

The DNAmaca interface language, to which ipc compiles, is described in [19]. Section 8 will use this to describe the construction of the stochastic probe, used to measure passage-time quantities in a PEPA model.

4.1. Passage-time Calculation

PEPA models reduce to an underlying continuous-time Markov chain (CTMC), so we consider an n state CTMC with $n \times n$ generator matrix $Q = q_{ij}$. Solving the linear system $\pi Q = 0$ subject to $\sum \pi_i = 1$ gives us the steady state vector, π. We calculate passage-time densities from many source states $\vec{i}$ to many target states $\vec{j}$ by means of an efficient uniformization-based analysis.

Uniformization [14, 26] transforms a CTMC into one in which all states have the same mean holding time $1/q$, by allowing *invisible* transitions from a state to itself. After normalisation of the generator matrix rows with an associated Poisson process of rate q, we obtain a one-step DTMC transition matrix P, given by:

$$P = Q/q + I \tag{2}$$

where $q > \max_i |q_{ii}|$ (to ensure that the DTMC is aperiodic).

While uniformization is normally used for transient analysis, it can also be employed for the calculation of response-time densities and quantiles [24, 25]. We add an extra, absorbing state to our uniformized chain, which is the sole successor state for all target states (thus ensuring we calculate the *first* passage-time density). We denote by P' the one-step transition matrix of the modified, uniformized chain. Remembering that the time taken to traverse a path with n hops in this chain will have an Erlang distribution with parameters n and q, the density of the time taken to pass from a set of source states $\vec{i}$ into a set of target states $\vec{j}$ is given by:

$$f_{\vec{i}\vec{j}}(t) = \sum_{n=1}^{\infty} \frac{q^n t^{n-1} e^{-qt}}{(n-1)!} \sum_{k \in \vec{j}} \pi_k^{(n)} \tag{3}$$

where

$$\pi^{(n+1)} = \pi^{(n)} P' \qquad \text{for } n \geq 0$$

with

$$\pi_k^{(0)} = \begin{cases} 0 & \text{for } k \notin \vec{i} \\ \pi_k / \sum_{j \in \vec{i}} \pi_j & \text{for } k \in \vec{i} \end{cases} \tag{4}$$

and in which $\boldsymbol{\pi}$ is any non-zero solution to $\pi = \pi P$. The corresponding passage-time cumulative distribution function is given by:

$$F_{\vec{i}\vec{j}}(t) = \sum_{n=1}^{\infty} \left\{ \left(1 - e^{-qt} \sum_{k=0}^{n-1} \frac{(qt)^k}{k!} \right) \sum_{k \in \vec{j}} \pi_k^{(n)} \right\}. \tag{5}$$

Truncation is employed to approximate the infinite sum in Eq. (3) (and Eq. (5)), terminating the calculation when the Erlang term drops below a specified threshold value. Concurrently, when the convergence criterion

$$\frac{||\pi^{(n+1)} - \pi^{(n)}||_\infty}{||\pi^{(n)}||_\infty} < \epsilon \tag{6}$$

is met, for given tolerance ϵ, the steady state probabilities of P' are considered to have been obtained with sufficient accuracy and no further multiplications with P' are performed.

5. ipc Tool Architecture

ipc performs the translation from PEPA to a stochastic Petri net formalism [27], and also incorporates any extra logic necessary for expressing the passage-time or steady-state query (see Section 8).

The ipc compiler consists of:

1. `.pepa` file parser
2. PEPA normal form translator
3. component state space explorer
4. DNAmaca component linker and `.mod` file generator
5. PEPA-passage specifier
6. command line parser for passage-time and steady-state queries

The `.pepa` file format allows, for instance, arbitrary numbers of sequential prefixes and also arbitrary numbers of summation and cooperation operations attributed to a single constant label. The normal form in question strictly enforces the binary summation and cooperation, insisting on constant labels after each operation; it therefore also has to take care of the unique labelling of all components states.

The component linker takes the DNAmaca description of the individual PEPA components and creates shared transitions with appropriate preconditions and actions to represent the cooperation over shared actions.

The PEPA-passage specifier augments the input PEPA model with process algebra probes and adds the requisite passage specification command to the DNAmaca file; this is explained in more detail in Section 8.

6. Case study: High-availability Web Server

In this section, we present the description and analysis of a PEPA model which we will use:

- to compare PEPA tool chains ipc/DNAmaca and PRISM in generating steady state solutions
- to demonstrate ipc/DNAmaca's capability to automate the generation of passage-time densities

Our model is of a high-availability web server. A typical application scenario for such a system is a web-based current events news feed which must meet strict quality-of-service requirements on availability and response-time. Regardless of whether the underlying technology is web-based or not, such systems require careful performance engineering to achieve peak efficiency [4]. In part, the strict QoS requirements are met by skewing the prioritisation for *fast* reads over writes so that writes are buffered and only processed at times of low read load. The consequence of this is that there is no guarantee that a reader will see the latest version of the site although high availability is maintained.

The system is built of a cluster of servers, each of which can fail independently and be repaired independently. If all of the servers fail then a special recovery mechanism can restart them all. We now proceed to describe the components of the system.

6.1. The server model

The server receives read requests each of which incurs a read lookup before the server is available to serve the next request (states Server and $\mathit{Server\_read}$ below). A successful write request requires that none of the servers are in the process of performing a read access, so that all servers can be simultaneously updated (therefore no $\mathit{s\_write}$ action is allowed in the $\mathit{Server\_read}$ state). The server may fail and while failed (the $\mathit{Server\_fail}$ state) read requests are not intercepted (no $\mathit{s\_read\_request}$ activities) whereas write requests ($\mathit{s\_write}$) are absorbed without action. It is assumed that server resynchronisation occurs during recovery. The write costs are different in the functioning and failure states (the costs are quantified by variables r_{s_w} and $r_{s_{fsw}}$ respectively). Failed servers may be repaired individually ($\mathit{s\_fail\_recover}$) or collectively ($\mathit{s\_fail\_recover\_all}$).

$$
\begin{array}{rcl}
\mathit{Server} & \stackrel{\mathit{def}}{=} & (\mathit{s\_read\_request}, \top).\mathit{Server\_read} \\
 & + & (\mathit{s\_fail}, r_{s_f}).\mathit{Server\_fail} \\
 & + & (\mathit{s\_write}, r_{s_w}).\mathit{Server} \\
\mathit{Server\_read} & \stackrel{\mathit{def}}{=} & (\mathit{s\_read\_lookup}, r_{s_{rl}}).\mathit{Server} \\
\mathit{Server\_fail} & \stackrel{\mathit{def}}{=} & (\mathit{s\_fail\_recover}, r_{s_{fr}}).\mathit{Server} \\
 & + & (\mathit{s\_fail\_recover\_all}, \top).\mathit{Server} \\
 & + & (\mathit{s\_write}, r_{s_{fsw}}).\mathit{Server\_fail}
\end{array}
$$

6.2. Server groups

The recovery of the servers is co-ordinated by a server group manager. The responsibility of this component is to witness server failures and recoveries from failures. When the server number, S, is reached and all servers have failed, the server group is restarted with all failures recovered simultaneously (as a result of a high-priority repair).

$$
\begin{array}{rcl}
\mathit{Server\_group}_0 & \stackrel{\mathit{def}}{=} & (\mathit{s\_fail}, \top).\mathit{Server\_group}_1 \\
\mathit{Server\_group}_i & \stackrel{\mathit{def}}{=} & (\mathit{s\_fail}, \top).\mathit{Server\_group}_{i+1} \\
 & + & (\mathit{s\_fail\_recover}, \top) \\
 & & .\mathit{Server\_group}_{i-1} : 1 \leq i < S \\
\mathit{Server\_group}_S & \stackrel{\mathit{def}}{=} & (\mathit{s\_fail\_recover\_all}, r_{s_{gsfra}}) \\
 & & .\mathit{Server\_group}_0
\end{array}
$$

Taking, for example, $S = 4$, the process instantiation expression for the server cluster is as shown below:

$$
\begin{array}{rcl}
\mathit{Servers} & \stackrel{\mathit{def}}{=} & (\mathit{Server} \underset{\mathcal{L}}{\bowtie} \mathit{Server} \underset{\mathcal{L}}{\bowtie} \mathit{Server} \underset{\mathcal{L}}{\bowtie} \mathit{Server}) \\
 & & \underset{\mathcal{L}'}{\bowtie} \mathit{Server\_group}_0
\end{array}
$$

where $\mathcal{L} = \{\mathit{s\_write}, \mathit{s\_fail\_recover\_all}\}$ and $\mathcal{L}' = \mathcal{L} \cup \{\mathit{s\_fail\_recover}\}$.

6.3. Buffered writes

The write buffer manages the promotion of buffered writes to server write actions. To ameliorate the relative infrequency of all the servers being available simultaneously to perform an $\mathit{s\_write}$, write requests are necessarily buffered until the buffer capacity, B, is reached. To take advantage of a $\mathit{s\_write}$ action when it occurs, the entire write buffer is executed and emptied.

$$
\begin{array}{rcl}
\mathit{Write\_buffer}_0 & \stackrel{\mathit{def}}{=} & (\mathit{b\_write}, \top).\mathit{Write\_buffer}_1 \\
\mathit{Write\_buffer}_i & \stackrel{\mathit{def}}{=} & (\mathit{b\_write}, \top).\mathit{Write\_buffer}_{i+1} \\
 & + & (\mathit{s\_write}, \top).\mathit{Write\_buffer}_0 \\
 & & : 1 \leq i < B \\
\mathit{Write\_buffer}_B & \stackrel{\mathit{def}}{=} & (\mathit{s\_write}, \top).\mathit{Write\_buffer}_0
\end{array}
$$

6.4. Web authors and browsers

Web authors (writers) issue web content to the system. Web browsers are the readers in our system. It is assumed that there are multiple writers and, comparatively, a much larger number of web browsers (readers). This accounts for the prioritisation of read access over writes.

We have a population of W writes and R reads in a given time period. After these have all been processed, all the writers and readers are simultaneously reset for the next time period. In this way, we maintain an irreducible system and we can easily measure when a fixed number of reads and writes have occurred (by passively observing $\mathit{rw\_reset\_all}$ actions), without having to set up further action counting-process components (e.g. $\mathit{Server\_group}$).

Writers may perform only buffered writes. They have no capacity to perform a server write directly.

$$\begin{array}{rcl} \mathit{Writer} & \stackrel{\mathit{def}}{=} & (b\_write, r_{w_{bw}}).\mathit{Writer\_writ} \\ \mathit{Writer\_writ} & \stackrel{\mathit{def}}{=} & (rw\_reset\_all, \top).\mathit{Writer} \end{array}$$

With three writers ($W = 3$) the process instantiation expression for the writers is as shown below:

$$\mathit{Writers} \stackrel{\mathit{def}}{=} \mathit{Writer} \underset{\mathcal{K}}{\bowtie} \mathit{Writer} \underset{\mathcal{K}}{\bowtie} \mathit{Writer}$$

where $\mathcal{K} = \{rw\_reset\_all\}$.

Readers send read requests and then await the response from the server.

$$\begin{array}{rcl} \mathit{Reader} & \stackrel{\mathit{def}}{=} & (s\_read\_request, r_{r_{srr}}) \\ & & .(s\_read\_lookup, \top).\mathit{Reader\_read} \\ \mathit{Reader\_read} & \stackrel{\mathit{def}}{=} & (rw\_reset\_all, \top).\mathit{Reader} \end{array}$$

If we model three readers ($R = 3$), for instance, we get:

$$\mathit{Readers} \stackrel{\mathit{def}}{=} \mathit{Reader} \underset{\mathcal{M}}{\bowtie} \mathit{Reader} \underset{\mathcal{M}}{\bowtie} \mathit{Reader}$$

where $\mathcal{M} = \{rw\_reset\_all\}$. Readers and writers are reset by the dedicated component:

$$\mathit{RW\_reset} \stackrel{\mathit{def}}{=} (rw\_reset\_all, r_{w_{ra}}).\mathit{RW\_reset}$$

6.5. The system equation

The system is built compositionally by composing the behaviours of the simpler component to form the behaviour of the model as a whole:

$$\begin{array}{rcl} \mathit{Environment} & \stackrel{\mathit{def}}{=} & \mathit{Writers} \underset{\{rw\_reset\_all\}}{\bowtie} \mathit{Readers} \\ & & \underset{\{rw\_reset\_all\}}{\bowtie} \mathit{RW\_reset} \\ \mathit{Web\_cluster} & \stackrel{\mathit{def}}{=} & \mathit{Servers} \underset{\{s\_write\}}{\bowtie} \mathit{Write\_buffer}_0 \\ \mathit{System} & \stackrel{\mathit{def}}{=} & \mathit{Environment} \underset{\mathcal{N}}{\bowtie} \mathit{Web\_cluster} \end{array}$$

where $\mathcal{N} = \{b\_write, s\_read\_request, s\_read\_lookup\}$.

7. Tool Comparisons

In this section, we present a comparison of the use of the ipc/DNAmaca tool with the use of the PRISM solver. Our running example is the high-availability web server model presented in the previous section. This model is configurable by varying the numbers of servers, S, buffer capacity, B, number of readers, R, and writers, W.

Tab. 1 presents the results of our model-building and solution. We varied the parameters S, B, R and W as indicated in column 1 in the table. The state space of the model is given in column 2. Timings are given in column 3 and column 4. All measurements were made on a 2.0GHz Pentium IV processor machine with 1Gb of memory, running Red Hat Linux 7.2. The GNU `time` command version 1.7 was used to obtain the measurement data. The time reported is elapsed real (wall clock) time used by the process, measured in seconds.

We used PRISM Version 1.3.1 and DNAmaca version 0.95. For both tools the problem is to solve the model for its equilibrium probability distribution. We used the same accuracy for the numerical precision of the results and used a range of solution options for both tools. The PRISM tool has three solution engines (MTBDD, Sparse and Hybrid) and seven numerical procedures so there are twenty-one possible combinations of these[3]. In the table below, we report the *best* time recorded for all combinations of solver and engine. The times taken by ipc and the equivalent PEPA compiler of the PRISM tool are not included in the run-times presented. The run-times reported reflect the processing time for the native formats of the tools only. These results show

Parameters S,B,R,W	**States**	**PRISM** run-time	**DNAmaca** run-time
3, 3, 2, 2	1,376	2.02	3.12
4, 3, 3, 3	21,248	7.55	6.70
5, 4, 3, 3	69,440	21.12	17.73
6, 5, 3, 3	211,968	70.62	58.14
6, 5, 4, 4	1,369,728	303.03	381.94

Tab. 1. Run-time measurements for the web server model

that DNAmaca is competitive with PRISM on solving PEPA models of small to moderate size. Models with larger state spaces can be solved faster with parallel and distributed versions of DNAmaca, which are also available [11, 15, 21].

8. Extracting Passage-time Densities with ipc

In this section, we make use of the `\passage` pragma in DNAmaca to extract passage-time quantities from our webserver case study. The version of DNAmaca which extracts

3 In PRISM version 1.3.1 not all of the solution procedures are implemented for the MTBDD engine so the number of possibilities is less than 21.

passage-times from purely Markov systems (release version HYDRA) uses uniformization to calculate both densities and cumulative distributions (as described in Section 4).

8.1. Automated PEPA Passage-time Specification

To ease the description of passage-times in DNAmaca, ipc provides us with a method to specify passages which relates directly to the high-level PEPA model. Given start actions and stop actions for the passage, ipc first adds a simple PEPA fragment to the model, which synchronises with the system and passively observes the occurrence of these key actions. We call these fragments *stochastic probes* [1], as they effectively measure the system for the required passage. Probes are similar in nature to the *testing component* concept used in [13], which were used to aid transient analysis of PEPA models. In our context, stochastic probes are specified with the set of starting actions, L_s, and the set of terminating actions, L_f, and take the general form:

$$\begin{aligned} ProbeX &\stackrel{\text{def}}{=} \sum_{s \in L_s} (s, \top).ProbeX\_run \\ &\quad + \sum_{f \in L_f \setminus L_s} (f, \top).ProbeX \\ ProbeX\_run &\stackrel{\text{def}}{=} \sum_{s \in L_s \setminus L_f} (s, \top).ProbeX\_run \\ &\quad + \sum_{f \in L_f} (f, \top).ProbeX \end{aligned}$$

The probe, initially in state $ProbeX$, waits for any one of the start actions $s \in L_s$ before moving to state $ProbeX\_run$. In waiting for a start action, it must absorb any stop actions it comes across without changing state. If any of the start actions also appears in the stop action set, L_f, they are treated as start actions first before being treated as stop actions in the state $ProbeX\_run$. A corresponding strategy is used for stop and start actions in the $ProbeX\_run$ state. The new system is created from the synchronisation of the old system with the probe as follows:

$$System' \stackrel{\text{def}}{=} System \underset{L_s \cup L_f}{\bowtie} ProbeX$$

Now, on seeing the probe first enter state $ProbeX\_run$, the passage-time measure is started and on the next occurrence of state, $ProbeX$, the measure is stopped. To encode this, ipc automatically adds the following passage measurement to the DNAmaca `.mod` file:

```
\passage {
   \sourcecondition{ (ProbeX_run > 0) }
   \targetcondition{ (ProbeX > 0) }
}
```

Here `ProbeX_run` and `ProbeX` are variables which encode the relevant state in the underlying DNAmaca Petri net model. If there are many underlying source states to a passage, as there often will be, then the steady-state distribution is used to weight the different possible passages, to give an overall passage distribution at equilibrium [15].

8.2. Examples

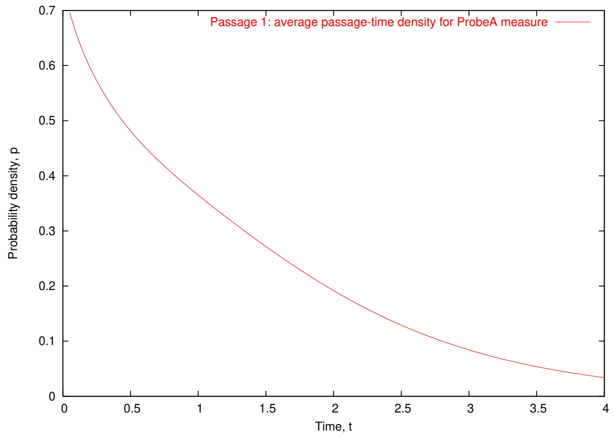

Fig. 1. Average passage-time density for the time taken to commit a write to the servers from the moment that it is buffered. (S = 5, B = 4, W = 3, R = 3: 69,440 states)

Fig. 1 shows the passage-time density representing the time taken for a write action to actually take place in the web server after it is first added to the write buffer. To achieve this ipc uses the PEPA fragment, $ProbeA$, which looks for a b_write action to signify a write update being placed into the write buffer. The fragment then stops measuring, on seeing the next s_write action which will represent the write buffer being flushed and the initial buffer write being committed.

$$\begin{aligned} ProbeA &\stackrel{\text{def}}{=} (b\_write, \top).ProbeA\_run \\ ProbeA\_run &\stackrel{\text{def}}{=} (b\_write, \top).ProbeA\_run \\ &\quad + (s\_write, \top).ProbeA \end{aligned}$$

$$System_1 \stackrel{\text{def}}{=} System \underset{\{b\_write,\, s\_write\}}{\bowtie} ProbeA$$

Fig. 2 represents the passage-time density between successive rw_reset_all actions; this is the time taken to complete an entire cycle of R reads and W writes. It is measured us-

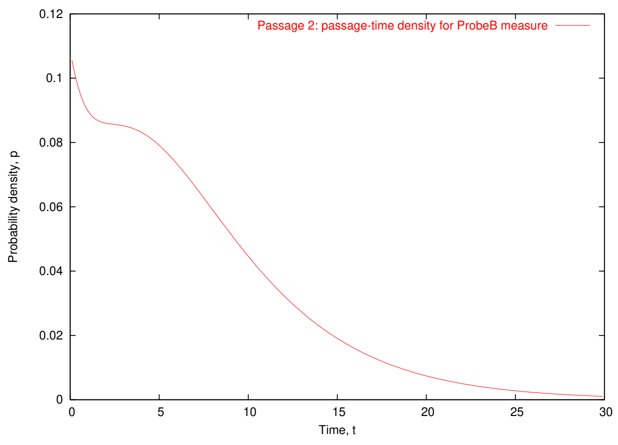

Fig. 2. Average passage-time density for R reads and W writes. (S = 5, B = 4, W = 3, R = 3: 69,440 states)

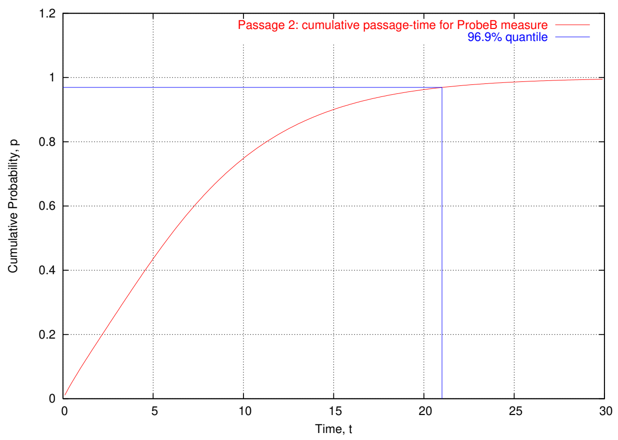

Fig. 3. Average cumulative passage-time distribution for R reads and W writes and quantile measure. (S = 5, B = 4, W = 3, R = 3: 69,440 states)

ing the PEPA fragment:

$$\begin{aligned} ProbeB &\stackrel{def}{=} (rw\_reset\_all, \top).ProbeB\_run \\ ProbeB\_run &\stackrel{def}{=} (rw\_reset\_all, \top).ProbeB \end{aligned}$$

$$System_2 \stackrel{def}{=} System \underset{\{rw\_reset\_all\}}{\bowtie} ProbeB$$

Fig. 3 displays the equivalent measure in cumulative distribution form (an option in the DNAmaca passage syntax). From the cumulative distribution, we can derive useful quality-of-service or quantile information, such as, the system will complete R reads and W writes in 21 time units, 96.9% of the time (also shown in Fig. 3).

9. Conclusion

In this paper, we have introduced the ipc tool as a means of compiling PEPA models into DNAmaca specifications and thereby access the rich passage-time analysis and large state-space capability that DNAmaca has to offer.

The main thrust of the paper has been to show how ipc automatically constructs PEPA probes to specify passages across process algebra models. These can then be passed to DNAmaca to provide the modeller with passage-time densities, cumulative distributions and passage-time quantiles.

We have also demonstrated that the ipc/DNAmaca tool chain compares favourably to current tool technologies, as used by PRISM, for the steady-state analysis of large PEPA models.

As future developments and to provide further analysis possibilities, we are looking to using ipc to compile full PML$_\mu$ (Probabilistic Modal Logic for PEPA models [5, 6]) specifications into DNAmaca, as an alternative way of expressing passage start and termination points.

Acknowledgements

Stephen Gilmore is supported by the *DEGAS (Design Environments for Global ApplicationS)* project IST-2001-32072 funded by the FET Proactive Initiative on Global Computing.

References

[1] J. T. Bradley. *Towards Reliable Modelling with Stochastic Process Algebras*. PhD thesis, Department of Computer Science, University of Bristol, Bristol BS8 1UB, UK, October 1999.

[2] J. T. Bradley, N. J. Dingle, P. G. Harrison, and W. J. Knottenbelt. Distributed computation of passage time quantiles and transient state distributions in large semi-Markov models. In *Performance Modelling, Evaluation and Optimization of Parallel and Distributed Systems*, Nice, April 2003. IEEE Computer Society Press.

[3] J. T. Bradley, N. J. Dingle, W. J. Knottenbelt, and P. G. Harrison. Performance queries on semi-Markov stochastic Petri nets with an extended Continuous Stochastic Logic. In *PNPM'03, Proceedings of Petri Nets and Performance Models*, 2003. (To appear).

[4] A. Carmona, L. Domingo, R. Macau, R. Puigjaner, and F. Rojo. Performance experiences of the Barcelona Olympic games computer system. In G. Haring and G. Kotsis, editors, *Computer Performance Evaluation, Modeling Techniques and Tools*, volume 794 of *Lecture Notes in Computer Science*, pages 52–75. Springer, 1994.

[5] G. Clark. Formalising the specification of rewards with PEPA. In M. Ribaudo, editor, *Process Algebra and Performance Modelling Workshop*, pages 139–160. CLUT, Torino, July 1996.

[6] G. Clark, S. Gilmore, J. Hillston, and M. Ribaudo. Exploiting modal logic to express performance measures. In B. R. Haverkort, H. C. Bohnenkamp, and C. U. Smith, editors, *TOOLS 2000, Proceedings of 11th Int. Conference on Computer Performance Evaluation: Modelling Techniques and Tools*, volume 1601 of *Lecture Notes in Computer Science*, pages 247–261, Schaumburg, IL, March 2000. Springer-Verlag.

[7] G. Clark, S. Gilmore, J. Hillston, and N. Thomas. Experiences with the PEPA performance modelling tools. In *UKPEW'98, Proceedings of the 14th UK Performance Engineering Workshop*, Edinburgh, July 1998.

[8] G. Clark and W. Sanders. Implementing a stochastic process algebra within the Möbius modeling framework. In L. de Alfaro and S. Gilmore, editors, *Proceedings of the first joint PAPM-PROBMIV Workshop*, volume 2165 of *Lecture Notes in Computer Science*, pages 200–215, Aachen, Germany, Sept. 2001. Springer-Verlag.

[9] N. J. Dingle, P. G. Harrison, and W. J. Knottenbelt. Response time densities in Generalised Stochastic Petri Net models. In *Proceedings of the 3rd International Workshop on Software and Performance (WOSP'2002)*, pages 46–54, Rome, July 2002.

[10] N. J. Dingle, P. G. Harrison, and W. J. Knottenbelt. Uniformization and hypergraph partitioning for the distributed computation of response time densities in very large Markov models. *Submitted to the Journal of Parallel and Distributed Computing*, 2002.

[11] N. J. Dingle, W. J. Knottenbelt, and P. G. Harrison. HYDRA: HYpergraph-based Distributed Response-time Analyser. In *Proceedings of the 2003 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA'03)*, Las Vegas, NV, June 2003.

[12] S. Gilmore and J. Hillston. The PEPA workbench: A tool to support a process algebra-based approach to performance modelling. In G. Haring and G. Kotsis, editors, *Proceedings of the 7th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, volume 794 of *Lecture Notes in Computer Science*, pages 353–368. Springer-Verlag, Vienna, May 1994.

[13] S. Gilmore and J. Hillston. Feature interaction in PEPA. In C. Priami, editor, *Process Algebra and Performance Modelling Workshop*, pages 17–26. Università Degli Studi di Verona, Nice, September 1998.

[14] W. Grassman. Means and variances of time averages in Markovian environments. *European Journal of Operational Research*, 31(1):132–139, 1987.

[15] P. G. Harrison and W. J. Knottenbelt. Passage-time distributions in large Markov chains. In M. Martonosi and E. d. S. e Silva, editors, *Proceedings of ACM SIGMETRICS 2002*, pages 77–85, Marina Del Rey, USA, June 2002.

[16] J. Hillston. Compositional Markovian modelling using a process algebra. In *Proceedings of the 2nd International Workshop on Numerical Solution of Markov Chains*. Kluwer Academic Press, Raleigh, January 1995.

[17] J. Hillston. *A Compositional Approach to Performance Modelling*, volume 12 of *Distinguished Dissertations in Computer Science*. Cambridge University Press, 1996. ISBN 0 521 57189 8.

[18] S. P. Jones, editor. *Haskell 98 Language and Libraries: The Revised Report*. Cambridge University Press, 2003.

[19] W. J. Knottenbelt. Generalised Markovian analysis of timed transitions systems. MSc thesis, University of Cape Town, South Africa, July 1996.

[20] W. J. Knottenbelt. *Parallel Performance Analysis of Large Markov Models*. PhD thesis, Imperial College, London, United Kingdom, February 2000.

[21] W. J. Knottenbelt and P. G. Harrison. Distributed disk-based solution techniques for large Markov models. In *NSMC'99, Proceedings of the 3rd Intl. Conference on the Numerical Solution of Markov Chains*, pages 58–75, Zaragoza, September 1999.

[22] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. In J.-P. Katoen and P. Stevens, editors, *Proc. 8th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'02)*, volume 2280 of *LNCS*, pages 52–66. Springer, 2002.

[23] MapleSoft Corporation. Maple computer algebra system. Web site, Apr. 2003. http://www.maplesoft.com/.

[24] B. Melamed and M. Yadin. Randomization procedures in the computation of cumulative-time distributions over discrete state Markov processes. *Operations Research*, 32(4):926–944, July–August 1984.

[25] J. K. Muppala and K. S. Trivedi. Numerical transient analysis of finite Markovian queueing systems. In U. N. Bhat and I. V. Basawa, editors, *Queueing and Related Models*, pages 262–284. Oxford University Press, 1992.

[26] A. Reibman and K. Trivedi. Numerical transient analysis of Markov models. *Computers and Operations Research*, 15(1):19–36, 1988.

[27] M. Ribaudo. Stochastic Petri net semantics for stochastic process algebras. In *PNPM'95, Proceedings of 6th Int. Workshop on Petri Nets and Performance Models*, pages 148–157, Durham, North Carolina, October 1995.